Programmation Logique : Prolog

Valentin NOEL et Romain Pascual

15 juin 2017

1 Problème 1 : Traversées de rivière

Les problèmes des traversée de rivière supposent généralement que l'on doit faire passer des individus d'un côté à l'autre de la rivière en ayant un nombre limité de personnage sur le bateau tout en interdisant à certains personnages de rester ensemble d'un même côté de la rivière.

Ici le problème est le suivant : "Lulu doit faire passer le chou, la chèvre et le loup de l'autre côté de la rivière et il n'a qu'une place sur son bateau. Si la chèvre et le chou sont ensemble sur une rive quand Lulu s'éloigne, la chèvre mange le chou. Et si le loup et la chèvre sont ensemble quand Lulu s'éloigne, le loup mange la chèvre."

On cherche à modéliser le problème en Prolog. La solution est donnée dans le fichier "rivière.pl"

La modélisation est la suivante : on utilise un tuple $(Lulu, Chou, Chevre, Loup)$ de booléens tel que chaque variable vaut 0 si le personnage est du premier côté de la rivière et 1 s'il est du second. On cherche donc à transformer le tuple $(0, 0, 0, 0,)$ en $(1, 1, 1, 1)$. Pour cela on commence par définir les états interdits (le loup et la chèvre d'un même côté ou encore la chèvre et le chou du même côté). Ensuite on modélise les déplacements possibles : déplacement du chou, de la chèvre, du loup ou de Lulu en interdisant les déplacements qui conduisent à un état interdit.

On cherche ensuite la solution à l'aide d'un parcours en largeur implémenté par des listes.

Le problème est finalement résolu par analogie avec une réflexion sur les graphes où les nœuds représentent les différents états licites et les arcs les déplacements du bateau. La "meilleure" solution est alors obtenue à l'aide d'un parcours en largeur du graphe. Le code du parcours en largeur à été trouvé sur internet et adapté à la situation.

Pour exécuter le code, il suffit d'utiliser la commande `solution` ou la commande `solution_ecrite` dans Prolog après avoir chargé le fichier "rivière.pl". Dans le premier cas, on obtient la liste des situations depuis celle ou tout le monde est sur la première berge jusqu'à celle ou tout le monde est sur la seconde berge. Avec la deuxième commande, on obtient la liste écrite des déplacements de Lulu, comme le montre la sortie ci-dessous :

```
romain@romain-N551JK:~/Documents/Perso/RepsGit/ProjetProlog$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [riviere].
true.

?- solution.
[situation(0,0,0,0),situation(1,0,1,0),situation(0,0,1,0),situation(1,1,1,0),
situation(0,1,0,0),situation(1,1,0,1),situation(0,1,0,1),situation(1,1,1,1)]
true.

?- solution_ecrite.
Lulu se déplace avec la chevre de la berge 1 à la berge 2.
Lulu se déplace seul de la berge 2 à la berge 1.
Lulu se déplace avec le chou de la berge 1 à la berge 2.
Lulu se déplace avec la chevre de la berge 2 à la berge 1.
Lulu se déplace avec le loup de la berge 1 à la berge 2.
Lulu se déplace seul de la berge 2 à la berge 1.
Lulu se déplace avec la chevre de la berge 1 à la berge 2.
Tout le monde a traversé
true .

?-
```

2 Problème 2 : Le compte est bon

"Le compte est bon est un jeu (télévisé) qui consiste à trouver une combinaison arithmétique de nombres afin d'obtenir un certain total."

Ici on suppose une entrée avec six nombres $(n_i)_{1 \leqslant i \leqslant 6}$ et un objectif N compris strictement entre 100 et 1000. Par ailleurs les nombres n_i sont à choisir parmi $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100$.

Par exemple, avec le tirage $1, 2, 3, 4, 5, 6$ et l'objectif $N = 150$, une combinaison gagnante est

$$(3 * (1 + 2)) - 4) * 5 * 6 = 150$$

Entre deux combinaisons gagnantes, celle qui utilise le moins de nombres sera préférée.

L'implémentation nécessite en premier lieu un certain nombre de prédicats pour pouvoir manipuler les listes (sélection d'un élément, calcul de la longueur, etc.). Le compte est bon est modélisé comme une liste d'opération arithmétiques, on extrait deux nombres, sur lesquels on applique une opération arithmétique ($+$, $-$, $*$ ou $*/$). Le résultat est alors rajouté à la liste des nombres. On obtient ainsi tous les résultats possibles à partir de l'ensemble de départ. Il suffit alors de s'arrêter lorsque l'on a construit l'objectif recherché. Pour obtenir la "meilleure" solution, c'est-à-dire celle utilisant le moins d'opération possibles, on se contente d'énumérer toutes les solutions et d'en choisir une parmi celles ayant le moins d'opérations. Par ailleurs, un certains nombres de prédicats sont utilisés pour vérifier que les contraintes sur les nombres utilisés et sur l'objectif sont remplis. Enfin, on trouvera des prédicats permettant une mise en forme minimale des solutions. Dans le cas d'une solution approchée, celle si est construite en testant tous les objectifs, en s'éloignant progressivement dans les deux directions.

Pour exécuter le code, il suffit d'appeler la commande `solution(+Liste, +Nombre)` avec pour paramètres les nombres que l'on peut utiliser et l'objectif à atteindre pour obtenir l'ensemble des solutions possibles. On utilisera la commande `meilleure_solution(+Liste, +Nombre)` pour afficher une meilleure solution (en nombre d'opérations). Enfin la commande `meilleure_solution_approchee(+Nombres, +But, +Erreur_toleree)` permet de chercher la meilleure solution approchée restant entre 100 et 1000 (exclus) avec pour borne supérieure de l'erreur la valeur de `Erreur_toleree`.

Notons que le grand nombre de cas à tester peut causer un certain délais avant d'obtenir le résultat. Comme on peut le voir sur la capture d'écran ci-dessous ou la valeur C représente le temps d'exécution du code en secondes.

```
romain@romain-N551JK:~/Documents/Perso/RepsGit/ProjetProlog$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [compte_est_bon].
true.

?- solution([100,2,75,3,1,10],888).
Le compte est bon.
100 - 2 = 98
75 + 3 = 78
78 + 1 = 79
79 * 10 = 790
790 + 98 = 888

true .

?- meilleure_solution([100,2,75,3,1,10],888).
Le compte est bon.
2 + 10 = 12
75 - 1 = 74
74 * 12 = 888
true .

?- get_time(A), meilleure_solution([100,2,75,3,1,10],888), get_time(B), C is B - A.
Le compte est bon.
2 + 10 = 12
75 - 1 = 74
74 * 12 = 888
A = 1497561367.207689,
B = 1497561390.4030046,
C = 23.19531559944153 .

?- get_time(A), meilleure_solution_approchee([1,2,3,4,5,6],964,10), get_time(B), C is B - A.
Le compte n'est pas bon, la meilleure solution trouvée est :
1 + 3 = 4
4 * 2 = 8
8 * 4 = 32
32 * 5 = 160
160 * 6 = 960
A = 1497561449.7442534,
B = 1497561651.5325816,
C = 201.78832817077637 .

?-
```