Programmation Logique : Prolog

Valentin NOEL et Romain Pascual

17 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 où 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.

Le code se trouve dans le fichier "compteEstBon.pl". Pour l'exécuter, 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 les captures d'écran ci-dessous où 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 .

?-
```

3 Problème 3 : Logique propositionnelle

3.1 Méthode des Tableaux

Il s'agit d'implémenter un prédicat qui étant donnée une formule ϕ indique si ϕ est prouvable par la méthode des tableaux (i.e s'il existe un tableau à partir de la négation de ϕ dont toutes les branches sont closes).

Les formules sont ici manipulées à l'aide d'arbres construits en s'appuyant sur les connecteurs logiques : la négation est d'arité 1 donc induit une branche tandis que la conjonction, la disjonction et l'implication dont d'arité 2 donc induisent deux branches. Les variables sont manipulées sous la forme d'entiers naturels. On construits alors simplement les littéraux avec les entiers relatifs : un entier négatif correspond à la négation d'une variable.

Les tableaux sont aussi manipulés comme des arbres, les nœuds sont alors α ou β suivant le type de branches. Les formules ne sont présentes que sur les feuilles, et sous la forme de listes. Ainsi si la feuille d'une branche est $[1, 2, et(1, 3)]$ cela signifie que l'on considère l'ensemble de formules $1, 2, 1 \wedge 3$.

On trouve donc un certain nombres de prédicats pour manipuler les listes (concaténation, recherche d'un élément, ajout en te de liste, etc.).

Ensuite le prédicat `formule(+F)` permet de construire les formules à l'aide des différents connecteurs logiques.

Enfin, on trouve les prédicats sur les tableaux (pour savoir si un tableau est développé, clos, etc.)

La construction d'un tableau à partir d'un ensemble de formules est réalisée de la manière suivante : on prend un liste de formule, on cherche une formule à développer, on la développe et on remplace la formule non développée par ses sous-formules. Ce processus est itéré tant qu'il existe une formule non développée puis on reconstruit

le tableau complet. Pour vérifier si une formule ϕ est prouvable par tableau, il suffit donc de construire le tableau associé à la $\neg\phi$ et à vérifier qu'il est clos.

Le code se trouve dans le fichier "tableaux.pl". On trouvera en en-tête quatre formules issues de l'exercice 1 du TD 3 pouvant être utilisés pour tester le code. La commande `solution(+F)` affiche à l'écran si la formule F est prouvable par tableaux. La commande `solution_avec_tableau` affiche aussi le tableau développé afin de pouvoir simplement contrôler le résultat. On peut voir un exemple d'exécution sur la capture d'écran suivante :

```
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).

?- [tableaux].
true.

?- solution(imp(imp(et(1,2),3), ou(imp(1,3), imp(2,3)))).
La formule est prouvable par tableaux.
true.

?- solution(ou(imp(1,2),imp(2,1))).
La formule est prouvable par tableaux.
true.

?- solution(imp(ou(1,et(2,3)),et(ou(1,2),ou(1,3)))).
La formule est prouvable par tableaux.
true.

?- solution(imp(ou(1,2),imp(3,imp(-2,-1)))).
La formule n'est pas prouvable par tableaux.
true.

?- solution_avec_tableau(ou(imp(1,2),imp(2,1))).
La formule est prouvable par tableaux.
alpha(alpha(alpha([-1,2,-2,1])))
true.

?-
```

3.2 Algorithme DPLL

Il s'agit d'implémenter un prédicat qui teste la satisfiabilité d'une formule booléenne ϕ en appliquant l'algorithme DPLL. On suppose ici que la formule est déjà mise sous forme normale conjonctive. La formule est alors un ensemble de clause et chaque clause est une disjonction de littéraux. Une clause est alors implémentée sous la forme d'une liste de littéraux et une formule sous la forme d'une liste de liste. Ici on oblige les variables à être représentées avec des entiers naturels. Les littéraux sont alors simplement représentés avec les entiers relatifs : un entier positif représente un variable et un entier négatif la négation de cette variable.

Après avoir défini un certain nombres de prédicats pour manipuler les listes, on implémente des prédicats pour manipuler les littéraux, les clauses et les formules (trouver les littéraux dans une formule, supprimer des littéraux dans une clause ou dans une formule, obtenir la liste des variables d'une formule, etc).

Il s'agit alors de définir les prédicats propres à la résolution avec l'algorithme DPLL :

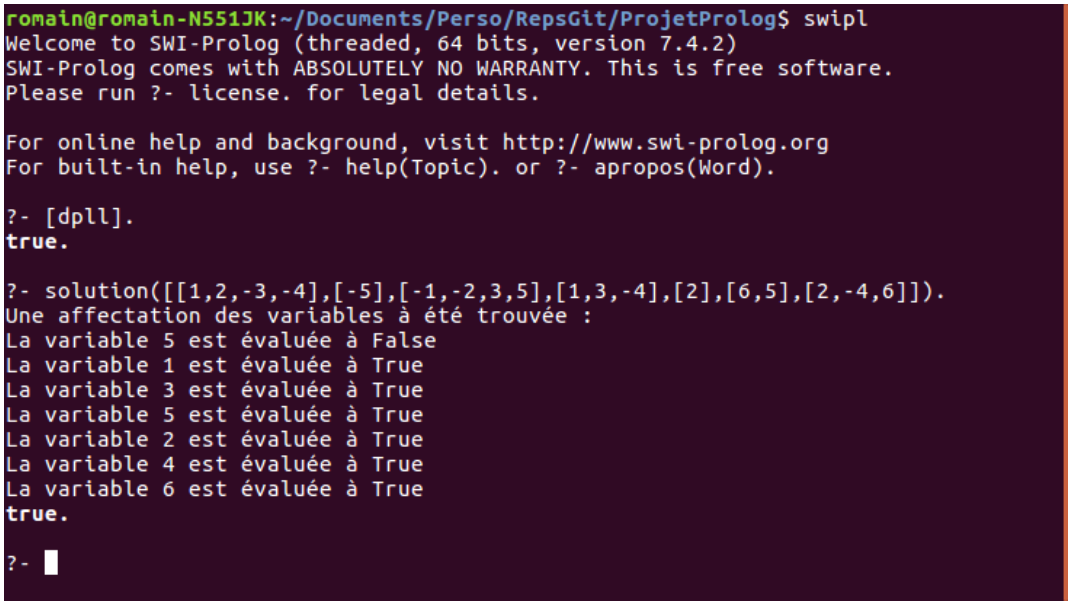
- le prédicat `get_unitaire` permet de trouver dans une formule sous forme normale conjonctive la liste des littéraux présents dans des clauses unitaires.
- le prédicat `get_polarisation_positive` permet de trouver dans une formule sous forme normale conjonctive la liste des littéraux possédant uniquement une polarisation positive.
- le prédicat `get_polarisation_negative` permet de trouver dans une formule sous forme normale conjonctive la liste des littéraux possédant uniquement une polarisation négative.

Algorithme L'algorithme est implémenté de la manière suivante :

- on cherche les littéraux dans des clauses unitaires, on les modifie les différentes clauses de la formule en fonction de si la valuation des littéraux mis en évidence les valide ou non, puis on résout par DPLL la nouvelle formule ;
- on cherche les littéraux de valuation uniquement positive, on les modifie les différentes clauses de la formule en fonction de si la valuation des littéraux mis en évidence les valide ou non, puis on résout par DPLL la nouvelle formule ;
- on cherche les littéraux de valuation uniquement négative, on les modifie les différentes clauses de la formule en fonction de si la valuation des littéraux mis en évidence les valide ou non, puis on résout par DPLL la nouvelle formule ;
- sinon on prend la variable en tête de liste et on essaie de lui affecter la valeur $true$, on les modifie les différentes clauses de la formule en fonction de si cette valuation les valide ou non, puis on résout par DPLL la nouvelle formule ;
- lors du backtracking, on prend la variable en tête de liste et on essaie de lui affecter la valeur $false$, on les modifie les différentes clauses de la formule en fonction de si cette valuation les valide ou non, puis on résout par DPLL la nouvelle formule ;

Dans un soucis de simplification, on évalue toutes les variables non significatives (c'est-à-dire les variables auxquelles on peut associer $true$ ou $false$ dans modifier la satisfiabilité de la formule) à $true$.

Le code se trouve dans le fichier "dpll.pl", la commande `solution(+Fnc)` permet d'obtenir une valuation de la formule Fnc décrite comme une liste de liste d'entiers relatifs, comme le montre la capture d'écran ci-dessous :



Par ailleurs, un prédicat `temp(+Fnc)` permet d'évaluer le temps d'exécution de la résolution. La figure ci-dessous montre la résolution d'une formule sous forme normale conjonctive issue d'un picross.

```
?- temp([[27,28,29,30,31],[32,33,34,35,36],[37,38,39,40,41],[42],[43,44,45],[46,
47,48,49,50],[51,52,53,54,55],[56,57,58],[59,60,61, 62],[63,64],[65,66,67],[68,6
9,70,71,72],[73,74],[75,76,77,78],[-27,-31],[-29,-27],[-28,-27],[-30,-27],[-29,-
31],[-28,-31],[-30,-31],[-29,-28],[-30,-29],[-30,-28],[-35,-33],[-35,-36],[-35,-
34],[-35,-32],[-33,-36],[-34,-33],[-33,-32],[-34,-36],[-32,-36],[-34,-32],[-37,-
40],[-39,-40],[-41,-40],[-38,-40],[-39,-37],[-37,-41],[-38,-37],[-39,-41],[-39,-
38],[-38,-41],[-43,-45],[-44,-45],[-44,-43],[-47,-50],[-48,-47],[-47,-46],[-49,-
47],[-48,-50],[-46,-50],[-49,-50],[-48,-46],[-49,-48],[-49,-46],[-53,-52],[-53,-
55],[-54,-53],[-53,-51],[-52,-55],[-54,-52],[-52,-51],[-54,-55],[-51,-55],[-54,-
51],[-57,-56],[-57,-58],[-56,-58],[-59,-62],[-60,-62],[-62,-61],[-60,-59],[-59,-
61],[-60,-61],[-64,-63],[-66,-65],[-66,-67],[-65,-67],[-68,-72],[-70,-68],[-69,-
68],[-71,-68],[-70,-72],[-69,-72],[-71,-72],[-70,-69],[-71,-70],[-71,-69],[-73,-
74],[-78,-77],[-77,-75],[-77,-76],[-78,-75],[-78,-76],[-76,-75],[-35,-27],[-33,-
27],[-35,-31],[-33,-31],[-36,-31],[-35,-29],[-29,-33],[-29,-36],[-34,-29],[-35,-
28],[-33,-28],[-28,-36],[-34,-28],[-32,-28],[-35,-30],[-30,-33],[-30,-36],[-30,-
34],[-30,-32],[-35,-40],[-35,-37],[-33,-40],[-33,-37],[-39,-33],[-36,-40],[-37,-
36],[-39,-36],[-41,-36],[-34,-40],[-34,-37],[-34,-39],[-34,-41],[-34,-38],[-32,-
40],[-32,-37],[-39,-32],[-32,-41],[-38,-32],[-53,-47],[-52,-47],[-53,-50],[-52,-
50],[-55,-50],[-53,-48],[-48,-52],[-48,-55],[-54,-48],[-53,-46],[-52,-46],[-46,-
55],[-54,-46],[-51,-46],[-49,-53],[-49,-52],[-49,-55],[-49,-54],[-49,-51],[-66,-
68],[-66,-72],[-66,-70],[-66,-69],[-65,-68],[-65,-72],[70,-65],[-65,-69],[-71,-6
5],[-68,-67],[-72,-67],[-70,-67],[-69,-67],[-71,-67],[1,-27],[2,-31],[3,-29],[4,
-28],[-30,5],[-35,1],[2,-33],[3,-36],[-34,4],[-32,5],[1,-40],[2,-37],[-39,3],[4,
-41],[-38,5],[-42,6],[-42,7],[-42,8],[9,-42],[10,-42],[11,-45],[12,-45],[13,-45]
,[12,-43],[13,-43],[-43,14],[-44,13],[-44,14],[-44,15],[-47,16],[17,-50],[-48,18
],[19,-46],[-49,20],[-53,16],[-52,17],[18,-55],[-54,19],[20,-51],[-57,21],[-57,2
2],[-57,23],[-56,22],[-56,23],[24,-56],[23,-58],[24,-58],[25,-58],[1,-62],[6,-62
],[6,-59],[11,-59],[-60,11],[-60,16],[16,-61],[21,-61],[2,-63],[-63,7],[12,-63],
[-63,17],[-64,7],[-64,12],[-64,17],[-64,22],[-66,3],[-66,8],[-66,13],[-65,8],[13
,-65],[-65,18],[13,-67],[18,-67],[-67,23],[3,-68],[8,-72],[13,-70],[-69,18],[-71
,23],[4,-73],[9,-73],[-73,14],[19,-73],[9,-74],[14,-74],[19,-74],[24,-74],[-77,5
],[10,-77],[10,-78],[-78,15],[-75,15],[20,-75],[20,-76],[25,-76],[27,-1,35,40],[
37,33,-2,31],[-3,29,39,36],[-4,28,34,41],[-5,32,38,30],[-6,42],[42,-7],[42,-8],[
42,-9],[42,-10],[-11,45],[43,-12,45],[43,-13,45,44],[43,44,-14],[44,-15],[47,53,
-16],[-17,52,50],[-18,48,55],[46,-19,54],[51,-20,49],[-21,57],[56,57,-22],[56,57
,-23,58],[56,-24,58],[-25,58],[-1,62],[63,-2],[68,-3,66],[73,-4],[-5,77],[59,-6,
62],[63,64,-7],[65,72,66,-8],[73,-9,74],[77,78,-10],[59,-11,60],[64,63,-12],[65,
-13,66,70,67],[73,74,-14],[75,78,-15],[60,61,-16],[-17,63,64],[-18,65,69,67],[73
,-19,74],[75,76,-20],[-21,61],[64,-22],[71,-23,67],[-24,74],[76,-25]]).
Une affectation des variables à été trouvée en 0.05550241470336914 secondes
true.

?-
```