Université d'Aix-Marseille - Licence informatique 3ème année

Logique - TP 2 de Prolog

1 Résolution d'une formule CNF

On considère les formules propositionnelles sous forme normale conjonctive, par exemple : $(\overline{a} \vee b \vee c) \wedge (a \vee \overline{b}) \wedge (a \vee \overline{c})$). En Prolog, on représentera le littéral positif d'une variable `X` par le terme `p(X)` et son littéral négatif par le terme `n(X)`. On représentera une clause par une liste de littéraux : `[p(A), n(B)]` pour $a \vee \overline{b}$). On représentera une formule CNF par une liste de clauses : `[[n(A), p(B), p(C)], [p(A), n(B)], [p(A), n(C)]]` pour $(\overline{a} \vee b \vee c) \wedge (a \vee \overline{b}) \wedge (a \vee \overline{c})$). On représentera les valeurs de vérités par `v` et `f`.

a. Écrire le prédicat `clause_sat(C)` qui est vrai si au moins un des littéraux de la clause `C` est vrai. Par exemple, la requête `?- clause_sat([n(A), p(B), p(C)]).` retourne successivement `A=f`, `B=v` et `C=v`.

b. Écrire le prédicat `sat(F)` qui est vrai si la formule `F` représentée comme indiqué ci-dessus est satisfiable. Par exemple, la requête `sat([[n(A),p(B),p(C)], [p(A), n(B)], [p(A), n(C)]]).` retourne successivement `A=f, B=f, C=f`, puis `A=v, B=v`, puis `A=v, B=f, C=v`.

2 Exercices sur des entiers

À partir de cet exercice, vous pouvez utiliser les entiers de Prolog et le prédicat `is` pour faire les calculs arithmétiques. Pour ceci, référez-vous à l'introduction à Prolog qui vous a été fournie la semaine dernière.

a. *Ecrire le prédicat* `puissance(X, Y, Z)` *qui est vrai si,* $Z = X^Y$. (X et Y doivent être instanciés)

b. *Ecrire le prédicat* `entier(X)` *qui est vrai si X est un nombre entier (décimal).* La requête `?- entier(X).` doit donner `X=0` comme solution, puis le successeur de l'entier précédent tant qu'on demande une autre solution.

c. *Ecrire le prédicat* `entier(X, Min, Max)` *qui est vrai si* `X` *est un entier compris entre* `Min` *et* `Max`. La requête `?- entier(X, 2, 6).` doit donner successivement les solutions `X=2`, `X=3`, ..., `X=6` (tant qu'on demande une nouvelle solution).

d. *Ecrire le prédicat* `premier(X)` *qui est vrai si X est un nombre premier.* La requête `?- premier(X).` doit donner `X=2` comme solution, puis les nombres premiers suivants (3,5,7,11,...) tant qu'on demande une autre solution.

3 Un jeu de solitaire

Nous allons programmer un solveur de jeu de solitaire, dont trois variantes d'aire de jeu avec leur position de départ se trouvent en figure 1.

Le but du jeu est de faire disparaître tous les pions sauf un. Pour qu'un pion disparaisse, il faut qu'un autre pion qui le côtoie (en haut, en bas, à droite ou à gauche) saute au dessus de lui et aterrisse sur la case de l'autre côté, qui doit donc être vide au départ. Par exemple, à partir de la position initiale de la variante représentée au milieu de la figure, le pion en case (4, 1) peut sauter le pion en case (3, 1) et se retrouver sur la case (2, 1). Après ce coup, la case (2, 1) contient un pion mais les cases (3, 1) et (4, 1) sont vides. L'autre coup possible est le pion sautant de la case (2, 3) à la case (2, 1).

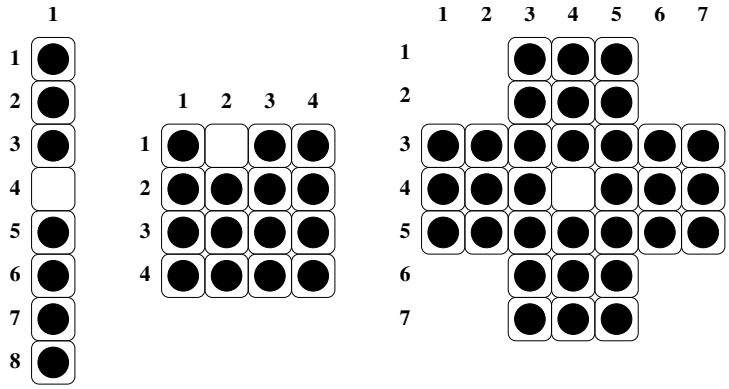

Figure 1: Dans les 3 positions de départ de chacune des variantes, toutes les cases sont occupées par un pion sauf une case qui est vide.

Avant d'écrire un prédicat de résolution du problème, nous allons écrire un certain nombre d'autres prédicats qui lui seront utiles.

a. Ecrire le prédicat `n_entiers(Min, Max, Liste)`, qui permet d'obtenir la liste des entiers compris entre `Min` et `Max` inclus. Ex : la requête `?- n_entiers(3, 6, L).` permet d'obtenir la réponse `L = [3, 4, 5, 6]`.

b. On va écrire le prédicat `produit_cartesien(L1, L2, P)`, qui permet d'obtenir le produit cartésien `P` de deux ensembles `L1` et `L2` (L1, L2 et P sont représentés sous forme de liste).
Ex : la requête `?- produit_cartesien([a, b], [1, 2, 3], P).` permet d'obtenir la réponse (unique) `P = [[a, 1], [a, 2], [a, 3], [b, 1], [b, 2], [c, 3]]`.

Mais, auparavant, il faudra écrire le prédicat `place_devant(X, L, L2)` qui permet d'obtenir la liste de couple `L2` dont chacun des premiers éléments des couples et `X` dont les deuxièmes éléments des couples sont les éléments de `L`. Ex : la requête `?- place_devant(a, [1, 2, 3], L).` permet d'obtenir la réponse (unique) `L = [[a, 1], [a, 2], [a, 3]]`.

c. Ecrire le prédicat `init1(P)` qui permet d'obtenir la liste des cases occupées (par un pion) de la position initiale de la première variante du jeu. Ecrire aussi les prédicats `init2(P)` et `init3(P)` pour les deux autres variantes. (On pourra faire des produits cartésiens et ensuite retirer une case vide.

d. Ecrire le prédicat `alignes(Case1, Case2, Case3)` qui permet d'obtenir les coordonnées `Case2` et `Case3` à partir de `Case1`. Ces coordonnées sont celles de 3 cases qui se côtoient soit horizontalement, soit verticalement. Ex : la requête `?- alignes([3,5], C2, C3).` permet d'obtenir 4 réponses :

```
C2 = [3, 6], C3 = [3, 7];
C2 = [3, 4], C3 = [3, 3];
C2 = [4, 5], C3 = [5, 5];
C2 = [2, 5], C3 = [1, 5]
```

e. Ecrire le prédicat `jouable(Occupe, Libre, Coup, Occupe2, Libre2)` qui, à partir des ensembles `Occupe` (les cases occupées) et `Libre` (les cases libres) permet d'obtenir `Coup` (un des coups jouables) et les ensembles `Occupe2` et `Libre2` qui résultent de ce coup. (Ce prédicat doit permettre d'obtenir tous les coups jouables à partir d'une position donnée, définie par la liste des cases occupées et la liste des cases libres). `Coup` est un terme de la forme `c(C1, C2)` où `C1` et `C2` sont des coordonnées de cases.

f. Ecrire le prédicat `solution(Occupe, Libre, Coups)` qui permet d'obtenir la liste `Coups` des coups à jouer à partir de la position caractérisée par `Occupe` et `Libre` pour obtenir une position ne contenant plus qu'une case occupée (par le dernier pion qui reste).

La requête `?- init2(Occupe), solution(Occupe, [[1,2]], Coups).` permet d'obtenir (par exemple) le résultat :

`Occupe = [[1,1],[1,3],[1,4],[2,1],[2,2],[2,3],[2,4],[3,1],[3,2],[3,3],[3,4],[4,1],[4,2],[4,3],[4,4]]`

`Coups = [c([1,4],[1,2]), c([1,1],[1,3]), c([3,1],[1,1]), c([2,3],[2,1]), c([1,1],[3,1]), c([3,4],[1,4]), c([1,4],[1,2]), c([4,1],[2,1]), c([4,2],[2,2]), c([1,2],[3,2]), c([3,3],[3,1]), c([2,1],[4,1]), c([4,4],[4,2]), c([4,1],[4,3])]`

N.B : Vous devriez obtenir presque instantanément une solution (parmi les nombreuses autres) pour les deux premières variantes du jeu. Pour la troisième variante, le temps de réponse risque d'être beaucoup trop élevé, à moins que vous n'ayez trouvé un moyen intelligent que vos prédicats soient très efficaces, ce qui n'est pas demandé. N'insistez donc pas trop sur la troisième variante.

4 Exercices combinatoires

a. *Ecrire le prédicat* `hanoi(N, T1, T2, T3, L)` *qui est vrai si* `L` *est la liste de coups qui permet de déplacer* `N` *disques de la tour* `T1` *vers la tour* `T3` *(via la tour* `T2`*).*

Exemple d'utilisation du prédicat : la requête `?- hanoi(3, a, b, c, L).`
donne la solution `L = [d(a,c),d(a,b),d(c,b),d(a,c),d(b,a),d(b,c),d(a,c)]`.

On rappelle que, quels que soient N, X, Y et Z, on déplace une tour de N disques de la tour X vers la tour Y en déplaçant d'abord N - 1 disques de la tour X vers la tour intermédiaire Z, puis le disque restant de la tour X vers la tour Y, puis les N - 1 disques de la tour Z vers la tour Y.

b. *Ecrivez le prédicat* `domino(L, S)` *qui est vrai si S constitue un arrangement des dominos de L pour former une suite valide de dominos.* On représente par `d(X,Y)` le domino qui contient X points d'un côté et Y points de l'autre. Une suite de dominos est valide si chaque côté de domino touchant un autre domino (le suivant ou le précédent) contient autant de points que le côté qu'il touche. Ex : la requête `dominos([d(1,2), d(1,3), d(1,4), d(2,3), d(2,4)], L).` donne 12 solutions, parmi laquelle `L = [d(1,2),d(2,3),d(3,1),d(1,4),d(4,2)]`.

c. On appelle *serpent* de longueur `N` et de hauteur `H`, une liste de `N` entiers compris entre 1 et `H` tels que la différence entre deux entiers consécutifs est 1. Ex : [2,3,4,3,4,5,4,3,2,1,2] est un serpent de longueur 11 et de hauteur 5.
Ecrivez le prédicat `serpent(N, H, V, S)` *qui est vrai si* `S` *est un serpent de longueur* `N`*, de hauteur* `H` *et qui commence par l'entier* `V`. Ex : la requête `?- serpent(4, 3, 2, S).` retourne les solutions `S = [2, 1, 2, 1]`, `S = [2, 1, 2, 3]`, `S = [2, 3, 2, 1]` et `S = [2, 3, 2, 3]`.

d. "Le compte est bon" est un jeu qui consiste à obtenir un entier donné à partir d'un ensemble de 6 entiers en utilisant les 4 opérations arithmétiques de base (addition, soustraction, multiplication et division). L'entier à obtenir est compris entre 100 et 999 et les entiers de l'ensemble de départ sont à prendre parmi 1, 2, ..., 9,10, 25, 50, 75 et 100. On ne peut utiliser chaque entier qu'une seule fois.

Ecrire le prédicat `comptestbon(L, N, Lo)` *qui détermine la liste* `Lo` *des opérations à effectuer pour obtenir* `N` *à partir de la liste* `L` *d'entiers.* Chaque opération de la liste `Lo` sera de la forme `op(X Op Y, Res)` où `Res` est le résultat de l'opération `Op` sur les entiers `X` et `Y`. `Op` peut être l'opérateur `+`, `-`, `*` ou `/`.
Ex : la requête `?- comptestbon([1, 25, 4, 3, 7, 2], 777, Lo).` permet d'obtenir la solution suivante (entre autres) : `L = [op(25 + 1, 26), op(7 * 4, 28), op(28 + 2, 30), op(30 * 26, 780), op(780 - 3, 777)]`.