M1 Informatique – Paris Diderot - Paris 7
Programmation Logique et par Contraintes

Examen partiel du 23 octobre 2018 - Durée: 1h50

Documents autorisés; le barème est donné à titre indicatif.

Exercice 1 (**5 points**) Pour chacune des requêtes suivantes, donner le résultat renvoyé par l'interpréteur Prolog (sans justifier) :

1. `(X=1;X=2),(Y=3;Y=4),!.`
2. `(X=1;X=2),!,(Y=3;Y=4).`
3. `!,(X=1;X=2),(Y=3;Y=4).`
4. `X=1+1,X=2.`
5. `X=1+1,X=:=2.`
6. `X is 1+1,X=2.`
7. `X=1+1,X\==2.`
8. `[X|[Y|[Z,Y]]]=[a,[b,c],[d,e],W].`
9. `bagof(X,(member(X,[1,2]),!),L).`
10. `not(not(X=1)).`

Exercice 2 (**4 points**)

1. Écrire un prédicat `partition_positifs(+l,-r,-s)` qui prend une liste d'entiers et renvoie les listes de ses éléments strictement positifs et de ses éléments négatifs ou nuls, réspectivement. Par exemple:

   ```
   [eclipse 1]: partition_positifs([3,-5,0,1,2,-6],R,S).
   R = [3, 1, 2]
   S = [-5, 0, -6]
   Yes (0.00s cpu)
   ```

2. Écrire un prédicat `partition(+l, +p, -r, -s)` qui prend une liste et le nom d'un prédicat unaire s'appliquant aux éléments de la liste, et renvoie la liste des éléments qui vérifient `p`, en `r`, et celle des éléments qui ne vérifient pas `p`, en `s`. Par exemple:

   ```
   [eclipse 2]: partition([3,-5,0,1,2,-6],pos,R,S).
   R = [3, 1, 2]
   S = [-5, 0, -6]
   Yes
   ```

   si `pos` est défini par `pos(X) :- X>0`, et

   ```
   [eclipse 3]: partition([1,a,5,b,f(42),-2],integer,R,S).
   R = [1, 5, -2]
   S = [a, b, f(42)]
   Yes
   ```

   Rappel: `Q=..[P,X]` unifie `Q` et `P(X)`.

Exercice 3 (**8 points**) En logique propositionnelle, un *litteral* est soit une variable propositionnelle soit la négation d'une variable propositionnelle, une *clause* est la disjonction d'un nombre quelconque de litteraux et une *forme normale conjonctive* (CNF) est la conjonction d'un nombre quelconque de clauses. Par exemple, la formule suivante est une CNF: $(x \vee \neg y) \wedge (\neg x \vee z) \wedge (\neg w)$. Remarque: la CNF qui est conjonction de 0 clauses est la constante 1, et la clause qui est disjonction de 0 litteraux est la constante 0.

En Prolog, nous représentons les variables propositionnelles par des atomes (un atome étant un terme `t` tel que le but `atomic(t)` réussit), la négation d'une variable `x` par le terme `neg(x)`, une clause par la liste des litteraux qui la composent et finalement une CNF par la liste des clauses qui la composent. Donc par exemple la CNF ci-dessus est representés par le terme `[[x,neg(y)],[neg(x),z],[neg(w)]]`.

1. Écrire un prédicat `est_cnf(+l)` qui réussit si l'argument représente une CNF, comme expliqué ci-dessus. Par exemple:

```
[eclipse 4]: est_cnf( [[x,neg(y)],[],[neg(w)]]).
Yes
[eclipse 5]: est_cnf([]).
Yes
[eclipse 6]: est_cnf([x,[neg(x),y]]).
No
```

2. Une clause est *rédondante* si elle contient à la fois `x` et `neg(x)`, pour une variable `x`. Écrire un predicat `normalise(+cnf, -res)` qui prend une liste qui représente une CNF et élimine ses clauses rédondantes.

   Par exemple

```
eclipse 7]: normalise([[x,neg(x)],[y,neg(w)],[y,z,neg(y)]],R).
R = [[y, neg(w)]]
Yes
```

3. Si c est une clause, x une variables propositionnelle et b une valeur de vérité, l'*évaluation de c par rapport à $x = b$* est défini comme suit:

$$eval(c, x, b) = \begin{cases} c & \text{si ni } x \text{ ni } \neg x \text{ n'apparaissent dans } c \\ 1 & \text{si } x \text{ est dans } c \text{ et } b = 1 \text{ ou bien } \neg x \text{ est dans } c \text{ et } b = 0 \\ c' & \text{si } c = c' \vee x \text{ et } b = 0 \text{ ou bien } c = c' \vee \neg x \text{ et } b = 1 \end{cases}$$

   ainsi par exemple $eval(x \vee \neg y, x, 0) = \neg y$, $eval(x \vee \neg y, y, 0) = 1$ et $eval(x \vee \neg y, w, 0) = x \vee \neg y$.

   L'évaluation d'une CNF f par rapport à $x = b$ est la CNF obtenue en mappant la fonction $eval(c, x, b)$ sur f, et en retirant au passage les clauses 1. Ainsi par exemple l'évaluation de $(x \vee \neg y) \wedge (\neg x \vee z) \wedge (w \vee t)$ par rapport à $x = 1$ donne la CNF $(z) \wedge (w \vee t)$.

   Ecrire un prédicat `evalue(+f,+x,+b,-r)` qui prend une CNF `f`, un atome `x` , une valeur de vérité `b` et renvoie la CNF obtenue par évaluation de `f` par rapport à `x=b`. Par exemple

```
eclipse 8]: evalue([[neg(x),y],[x,z],[t,neg(w)]],x,1,R).
R = [[y],[t,neg(w)]]
yes
```

4. Ecrire un prédicat `choosevar(+f,-x)` qui prend une CNF et renvoie une à une ses variables. Par exemple

```
eclipse 9]: choosevar([[neg(x),y],[t,z]],X).
X = x ? ;
X = y ? ;
X = t ? ;
X = z ? ;
yes
```

5. Ecrire un prédicat `solution(+f, -a)` qui prend une CNF et renvoie une affectation qui la satisfait, en utilisant `choosevar` et `evalue`. Par exemple:

```
eclipse 10]: solution([[x,neg(y)],[neg(x),neg(t)]],A).
A = [(t,0),(x,1)]
Yes
```

Exercice 4 (**3 points**) On considère le jeu soustractif $\{2, 3\}$: une position est un nombre naturel, et les coups jouables à partir de n sont $n - 2$, si $n > 1$, et $n - 3$, si $n > 2$; le joueur qui ne peut plus jouer perd.

1. Définir le prédicat `move/2` pour ce jeu.
2. Définir le prédicat `gagne/1` utilisant la méthode "force brute", et dessiner l'arbre de dérivation de `gagne(2)`.
3. Définir un prédicat `gagne1/1` "intelligent" (= non récursif) pour ce jeu.