Partiel 2016, exercice 1

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

...

Dans les questions suivantes, on suppose que le fichier `ex.pl` ci-dessous ait été compilé:

```
/********* ex.pl *********/
a(0,1).
b(X,1):-X=0.
c(X,1):-X=:=0.
d(X,1):-X==0.
/************************/
```

7. `a(1-1,Y).`
8. `b(0,Y).`
9. `c(1-1,Y).`
10. `d(1-1,Y).`

Partiel 2014, exercice 1

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

...

Dans les questions suivantes, on suppose que le fichier `ex.pl` ci-dessous ait été compilé:

```
/* fichier fact.pl */
fact1(0,1).
fact1(N,R) :- N>0, P is N-1, fact1(P,V), R is N*V.
fact2(X,1) :- X=:=0.
fact2(N,R) :- N>0, P is N-1, fact2(P,V), R is N*V.
/* fin de fact.pl */
```

7. `fact1(3-3,R).`
8. `fact2(3-3,R).`
9. `fact1(7-3,R).`
10. `fact2(7-3,R).`

Partiel 2008, Exercice 2

On considère les trois prédicats suivants :

```
if_1(T,P,Q):-T,!,P.
if_1(T,P,Q):-Q.

if_2(T,P,Q):-T,P,!.
if_2(T,P,Q):-Q.

if_3(T,P,Q):-!,T,P.
if_3(T,P,Q):-Q.
```

1. Pour chaque paire `j,k`, ($1 \leq$ `j` $<$ `k` ≤ 3) trouver des arguments `T_jk,P_jk,Q_jk` tels que les buts `if_j(T_jk,P_jk,Q_jk)` et `if_k(T_jk,P_jk,Q_jk)` donnent des résultats différents[1].
2. Dessiner les arbres de dérivation de deux de ces buts, pour une paire `j,k` au choix.
3. Lequel de ces prédicats implémente correctement le `if-then-else`? (motiver brièvement).

[1] Il s'agit d'exhiber trois triplets: `T_12,P_12,Q_12`, puis `T_13,P_13,Q_13` et finalement `T_23,P_23,Q_23`.

Partiel 2014, Exercice 3

Soit `p/1` le prédicat défini par:

```
p(1).
p(2).
p(3).
p(4).
```

1. Dessiner l'arbre de dérivation de la requête `p(X).`
2. Dessiner l'arbre de dérivation de la requête `p(X),!.`

Partiel 2014, Exercice 3, suite et fin

On se propose de définir un prédicat `p2/1` tel que la requête `p2(X)` renvoie les deux premiers résultats renvoyés par `p(X)`, et termine. Voici deux définitions de `p2`, dont une seule est correcte.

```
p_aux(X,Y) :- p(X), p(Y), X\==Y, !.
p2(X) :- p_aux(X,_).
p2(X) :- p_aux(_,X).
/*********/
p_aux(X) :- p(X) , !.
p2(X):- p_aux(X) ; p_aux(X).
```

3 Quel prédicat satisfait la spécification donnée, celui défini en (a) ou en (b) ci-dessus? Que fait l'autre prédicat? Justifiez vos réponses.

Partiel 2015, Exercice 4

On appellera *formules* les termes de Prolog définis inductivement ci-dessous:

- Les atomes (c.à.d. les constantes) sont des formules.
- Si `p` est une formule, alors `neg(p)` est une formule.
- Si `p` et `q` sont deux formules, alors `et(p,q)` et `ou(p,q)` sont des formules.
- Aucun autre terme n'est une formule.

Chaque élémént de l'ensemble de termes ainsi défini représente une formule du calcul propositionnel. Par exemple, le terme `ou(neg(et(z,t)),ou(w,v))` représente la formule du calcul propositionnel qu'on écrit généralement $\neg(z \wedge t) \vee (w \vee v)$. D'autres exemples de formules:

$$\texttt{x} \qquad \texttt{et}(\texttt{x}, \texttt{neg}(\texttt{y})) \qquad \texttt{et}(\texttt{et}(\texttt{z}, \texttt{toto}), \texttt{ou}(\texttt{w1}, \texttt{v2}))$$

. Exemples de termes qui ne sont pas de formules:

$$\texttt{et}(\texttt{x}, \texttt{f}(4)) \qquad \texttt{neg}(2+1)$$

Partiel 2015, Exercice 4, suite

1. Écrire un prédicat `est_formule(P)` qui réussit si le terme `P` est une formule, et échoue sinon (rappel: le prédicat `atomic(T)` réussit si `T` est un atome, et échoue sinon.).
2. Écrire un prédicat `symboles(+P,-At,-Neg,-Et,-Ou)` qui compte le nombre d'atomes (y compris les doublons), de négations, de conjonctions et de disjonctions d'une formule. Exemple:

```
[eclipse 5]: symboles(et(x,ou(x,y)),A,N,E,O).
A = 3
N = 0
E = 1
O = 1
Yes (0.00s cpu)
```

Partiel 2015, Exercice 4, suite

3 Écrire un prédicat `variables(+P,-Liste_Var)` qui renvoie la liste (sans doublons) des atomes de la formule `P`. Vous pouvez utiliser le prédicat `union(+L1,+L2,-L3)` qui renvoie l'union sans doublons des listes `L1` et `L2`. Exemple:

```
[eclipse 6]: variables(et(x,ou(x,y)),L).
L = [x, y]
Yes (0.00s cpu)
```

4 Écrire un prédicat `combiner(+L,-A)` qui prend une liste d'atomes et renvoie les affectation de valeurs de verité pour ces atomes. Exemple:

```
eclipse 7]: combiner([x,y],A).
A = [(x, 1), (y, 1)]
Yes (0.00s cpu, solution 1, maybe more) ? ;
A = [(x, 0), (y, 1)]
Yes (0.00s cpu, solution 2, maybe more) ? ;
A = [(x, 1), (y, 0)]
Yes (0.00s cpu, solution 3, maybe more) ? ;
A = [(x, 0), (y, 0)]
Yes (0.00s cpu, solution 4)
```

Partiel 2015, Exercice 4, suite et fin

5 Écrire un prédicat `evaluer(+P,+A)` qui prend une formule et une affectation de valeurs de verité pour les variables de la formule, et réussit si la formule est vérifiée par l'affectation donnée. Exemple:

```
[eclipse 8]: evaluer(et(x,y),[(x,1),(y,1)]).
Yes (0.00s cpu)
[eclipse 39]: evaluer(et(x,y),[(x,1),(y,0)]).
No (0.00s cpu)
```

6 Écrire un prédicat `satsfait(+P,-A)` qui prend une formule et renvoie les affectations qui satisfont la formule. Exemple:

```
[eclipse 9]: satisfait(ou(x,y),L).
L = [(x, 1), (y, 1)]
Yes (0.00s cpu, solution 1, maybe more) ? ;
L = [(x, 0), (y, 1)]
Yes (0.00s cpu, solution 2, maybe more) ? ;
L = [(x, 1), (y, 0)]
Yes (0.00s cpu, solution 3, maybe more) ? ;
No (0.00s cpu)
```

Partiel 2015, Exercice 3

Soit `map2_somme(+L1,+L2,-R)` le prédicat qui:

- calcule la somme rang par rang de ses deux premiers arguments si ceux-ci sont deux listes d'entiers de même longueur;
- affiche un message d'erreur et échoue, sinon.

Voici quelques exemples d'utilisation de `map2_somme`:

```
[eclipse 1]: map2_somme([1,2,3],[4,5,6],R).
R = [5,7,9]
Yes (0.00s cpu)
[eclipse 2]: map2_somme([1,2,3],[4,5],R).
Arguments non valides
No (0.00s cpu)
[eclipse 3]: map2_somme([1,a],[2,3],R).
Arguments non valides
No (0.00s cpu)
```

1. Écrire le prédicat `map2_somme` (rappel: le prèdicat `integer(X)` réussit si `X` est un entier, et il échoue sinon.).

Partiel 2015, Exercice 3, suite

2 Écrire un prédicat `map2(+P,+L1,+L2,-R)` tel que `map2(somme,L1,L2,R)` se comporte exactement comme `map2_somme(L1,L2,R)` si `somme` est défini par
`somme(X,Y,Z) :- Z is X+Y.`
et qui plus généralement mappe le prédicat ternaire `P` sur les listes `L1` et `L2`, et renvoie le résultat. Par exemple, si le prédicat `produit` est défini par
`produit(X,Y,Z) :- Z is X*Y.`
on doit avoir:

```
[eclipse 4]: map2(produit,[1,2,3],[4,5,6],R).
R = [4,10,18]
Yes (0.00s cpu)
```

Rappel: le prédicat `Q=..[P,X,Y,Z]` unifie `Q` et `P(X,Y,Z)`.

3 Suivant le même principe qu'au point précédent, écrire un prédicat `map3(+P,+L1,+L2,+L3,-R)` qui mappe un prédicat à quatre arguments sur trois listes d'entiers.

Examen Juin 2015, Exercice 2

Le jeu *nimble* est une variante du jeu de nim. Une position du jeu se compose d'un certain nombre de tas de pièces de monnaie, disposés en une rangée comme dans la figure ci dessous.

A son tour, chaque joueur choisit un des tas, en prélève une pièce et la dépose sur l'un des tas situés à gauche de celui-ci (donc, le tas le plus à gauche ne peut pas être choisi).
Le joueur qui joue le dernier, en deplaçant la toute dernière pièce se trouvant sur un tas différent du tas le plus à gauche vers le tas le plus à gauche, gagne.
En prolog, on représente une position du jeu par une liste d'entiers, dont la longueur est le nombre de tas de la position et dont les éléments correspondent au nombre de pièces se trouvant sur chacun des tas, de gauche à droite.

Examen Juin 2015, Exercice 2, suite

1. Dessiner l'arbre de jeu de racine `[0,1,2]`. Cette position est-elle gagnante?
2. Définir le prédicat `move(+Position1,-Position2)` qui énumère les positions atteignables à partir d'une position donnée. Par exemple, on aura:

```
| ?- move([0,1,2],L).

L = [1,0,2] ? ;

L = [1,1,1] ? ;

L = [0,2,1] ? ;

no
```

   Tous les prédicats auxiliaires utilisés doivent être définis.
3. Définir un prédicat `gagne(+Position)` qui réussi s'il existe une stratégie gagnante à partir de la position donnée (utiliser l'approche "force brute").

Examen Juin 2015, Exercice 2, suite et fin

4 Considérons à présent uniquement les positions comportant trois tas, et pour de telles positions, appellons m le nombre de pièces se trouvant dans le tas du milieu et d le nombre de pièce du tas de droite (le nombre de pièce du tas de gauche est irrelevant). Le tableau ci dessous montre, pour des petites valeurs de m (sur les lignes) et d (sur les colonne), la nature gagnante (G) ou perdante (P) de la position correspondante:

d/m	0	1	2	3
0	P	G	P	G
1	G	G	G	G
2	P	G	P	G
3	G	G	G	G

En déduire le critère qui permet de décider si une position est gagnante, en fonction de m et de d, et définir le prédicat `gagne_bis(+Position)` correspondant à ce critère.

5 Démontrer que le critère trouvé au point précedent est correcte.