Accumulateurs, listes de différence, DCG, révision

Exercice 1
Écrire un prédicat `renverse(+L,-LR)` qui renverse la liste `L` en utilisant le prédicat `append/3`. `renverse([a,b,c],R)` devrait donner `R = [c,b,a]`. Ensuite, écrire un prédicat `renverse2(+L,-LR)` qui renverse la liste `L` en utilisant un accumulateur.

Exercice 2
Écrire un prédicat `minetmax(+L,-Min,-Max)` qui calcule le minimum et le maximum d'une liste d'entiers non vide avec un seul parcours de liste. Utiliser deux accumulateurs. Essayer de faire le moins possible de comparaisons.

Exercice 3
Définir un prédicat `ajout_fin(+DL,+EL,-RDL)` qui ajoute un élément `EL` à la liste de différence `DL`. Rappel : une liste de différence est de la forme `[...|X]-X`.
`ajout_fin([a,b,c|X]-X,d,R)` devrait donner `R = [a,b,c,d|_185]-_185` ou similaire.

Exercice 4
- Écrivez une grammaire pour des phrases françaises très simples. Utilisez les symboles suivants :

Symbole	Sens	Exemples
P	phrase	le chat joue avec la balle jaune
GN	groupe nominal	le chat, la balle jaune
GV	groupe verbal	joue avec la balle, chante
GP	groupe prepositionnel	avec le ballon
N	nom	balle, chat
V	verbe	joue, chante, voit
P	preposition	avec, sous
A	adjectif	jaune, rouge, bleu
Art	article	le, la

- Donnez une phrase française correcte qui n'est pas une phrase de votre grammaire.
- Donnez une phrase de la grammaire qui n'est pas une phrase française correcte.
- Écrivez la DCG correspondante à la grammaire et vérifiez en PROLOG que les phrases données par vous précedément (ne) sont (pas) acceptées.
- Définissez un prédicat `phrases/1` qui affiche toutes les phrases correctes à partir d'un symbole.
- Modifiez la DCG pour tenir compte de l'accord entre nom et adjectif.

Exercice 5
On se propose de définir un prédicat `p2/1` tel que la requête `p2(X)` renvoie les deux premiers résultats renvoyés par `p(X)` défini ailleurs, 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_version2(X) :- p(X) , !.
p2_version2(X):- p_aux_version2(X) ; p_aux_version2(X).
```

Quel prédicat satisfait la spécification donnée ? Que fait l'autre prédicat ?

Exercice 6
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 :

`x` `et(x,neg(y))` `et(et(z,toto),ou(w1,v2))`

Exemples de termes qui ne sont pas de formules : `et(x,f(4))` `neg(2+1)`

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)
```

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 affectations 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)
```

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 `satisfait(+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)
```

Exercice 7

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.

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").
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 :

$d \backslash m$	0	1	2	3	4
0	P	G	P	G	G
1	G	G	G	G	G
2	P	G	P	G	P
3	G	G	G	G	G
4	P	G	P	G	P

   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 correct.