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

Examen partiel du 29 octobre 2015 - Durée: 2h00

*Documents autorisés; le barème est donné à titre indicatif.*

**Les points 4, 5 et 6 de l'exercice 4 sont hors barème**

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

1. `X=3+4, X==7.`
2. `X is 3+4, X==7.`
3. `[A|[B,C|[6]]]=[3,4,5,D].`
4. `A=1,B=A,C=A+B,ground(C).`
5. `setof(X,(X=1;X=2;X=3),L).`
6. `setof(X,(X>0,X<4),L).`

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

```
/********* ex.pl *********/
a_1(X,Y):-!,b(X),c(Y).
a_2(X,Y):- b(X),!,c(Y).
a_3(X,Y):-b(X),c(Y),!.
b(0).
b(1).
c(0).
c(1).
/***********************/
```

7. `a_1(X,Y).`
8. `a_2(X,Y).`
9. `a_3(X,Y).`

**Exercice 2** (**5 points**) On dira qu'une liste de chiffres binaires est *dense* si tout préfixe de la liste contient au moins autant de `1` que de `0`. Par exemple, la liste `[1,0,1,1,0,0]` est dense, et la liste `[1,0,0,1,1]` ne l'est pas car son préfixe `[1,0,0]` contient plus de `0` que de `1`. Dans les points 1, 2 et 3 de cet exercice, vous mettrez en œuvre un algorithme de type "générer et tester" pour engendrer toutes le listes denses de taille donnée.

1. Écrire un prédicat `engendre(+N,-L)` qui prend un entier `N` et construit par des retours en arrière successifs toutes les listes binaires de taille `N`. Par exemple:

   ```
   [eclipse 1]: engendre(2,L).
   L = [1, 1]
   Yes (0.00s cpu, solution 1, maybe more) ? ;
   L = [1, 0]
   Yes (0.00s cpu, solution 2, maybe more) ? ;
   L = [0, 1]
   Yes (0.01s cpu, solution 3, maybe more) ? ;
   L = [0, 0]
   Yes (0.01s cpu, solution 4, maybe more) ? ;
   No (0.01s cpu)
   ```

2. Écrire un prédicat `est_dense(+L)` qui prend une liste binaire `L` et reussit si `L` est dense, échoue sinon. On pourra utilement définir un prédicat auxiliaire prenant comme arguments une liste et un compteur.

3. Écrire un prédicat `denses(+N,-L)` qui prend un entier `N` et renvoie toutes les liste binaires denses de taille `N`, en utilisant `engendre` et `est_dense`. Exemple:

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

4. Écrire un prédicat `denses_bis(+N,-L)` qui prend un entier `N` et renvoie toutes les liste binaires denses de taille `N` sans générer toutes les listes binaires (on pourra utilement définir un prédicat auxiliaire prenant deux compteurs et renvoyant une liste).

5. Écrire un prédicat `card_denses(+N,-R)` qui calcule le nombre de liste binaires denses de taille `N`, en utilisant l'un ou l'autre des prédicats `denses`, `denses_bis`.

**Exercice 3** (**5 points**) 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.).

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.

**Exercice 4** (**5 points**) 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:

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

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 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)
```

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)
```