Master d'ingénierie Informatique de Paris Diderot - Paris 7
M1: Prolog et programmation par contraintes

Examen partiel du 2 novembre 2012- Durée: 1h45
Documents autorisés; le barème est donné à titre indicatif.

Exercice 1 (**3 points**) Pour chacune des requêtes suivantes, donner le résultat renvoyé par l'interpréteur Prolog (sans justifier) :

1. `X is 12, X==12`
2. `!, (X is 1; X is 2).`
3. `(X is 1; X is 2), !.`
4. `f(X,h(Z,Y))=f(g(a,b),h(X,i(X))).`
5. `[1,2|[3|X]]=[1|[2,3,4]].`
6. `\+(\+ !).`

Exercice 2 (**7 points**)

1. - Écrire un prédicat `puissance_2(+N)` tel que `puissance_2(N)` réussit si et seulement si `N` est une puissance de 2.
   - Écrire un prédicat `existe_puissance_2(+L)` tel que `existe_puissance_2(L)` réussit si et seulement si la liste `L` contient une puissance de 2.
   - Écrire un prédicat `pourtout_puissance_2(+L)` tel que `pourtout_puissance_2(L)` réussit si et seulement si tous les éléments de la liste `L` sont des puissances de 2.

2. Rappel sur le prédicat `=..`

```
 ?- X=..[f,a,b].
X = f(a,b)
yes
```

   A l'aide du prédicat `=..`, écrire les prédicats binaires `exists` et `forall` tels que:

   - `exists(+Predicat,+Liste)` réussit si et seulement si au moins un élément de `Liste` satisfait le prédicat unaire `Predicat` .
   - `forall(+Predicat,+Liste)` réussit si et seulement si tous les éléments de `Liste` satisfont le prédicat unaire `Predicat` .

   Par exemple, si le prédicat unaire `positif` est défini par la règle: `positif(X):-X>0.` on doit avoir:

```
?- exists(positif,[-1,2]).
yes
?- forall(positif,[-1,2]).
no
```

3. Soit `L` une liste de listes d'entiers. Utiliser les prédicats définis ci-dessus pour écrire les prédicats unaires:

   - `exists_forall_positif(+L)` qui réussit si et seulement si `L` contient au moins une liste dont tous les éléments sont positifs, et

- `forall_exists_positif(+L)` qui réussit si et seulement si tous les éléments de `L` contiennent au moins un entier positif.

Exercice 3 (**7 points**) La conjecture de Goldbach est une assertion mathématique non démontrée qui s'énonce comme suit:

Tout nombre entier pair strictement supérieur à 3 peut s'écrire comme la somme de deux nombres premiers.

On ne connaît pas de contre-exemples à cet énoncé.

Écrire un prédicat sans arguments `goldbach/0` dont le comportement est illustré par les exemples d'utilisation au top level ci-dessous:

```
| ?- goldbach.
Donner un nombre pair plus grand que 3:
36.
36 est la somme des nombres premiers 31 et 5.
yes

| ?- goldbach.
Donner un nombre pair plus grand que 3:
71.
71 n'est pas un nombre pair plus grand que 3.
yes

| ?- goldbach.
Donner un nombre pair plus grand que 3:
96.
96 est la somme des nombres premiers 89 et 7.
yes

| ?- goldbach.
Donner un nombre pair plus grand que 3:
trente-quatre.
trente-quatre n'est pas un nombre pair plus grand que 3.
yes
```

Les prédicats auxiliaires utilisés dans la définition de `goldbach/0` doivent être définis.

L'ordre dans lequel les nombres premiers sont engendrés et testés n'a pas d'importance: les réponses `5 et 31`, ou même `13 et 23` par exemple, sont tout aussi acceptables dans le premier exemple d'utilisation.

Exercice 4 (**3 points**) On considère la variante suivante du jeu de Marienbad: une position du jeu est un nombre entier positif, et les coups disponibles pour les joueurs sont

- Soustraire 1.
- Diviser par 2 (il s'agit de la division entière, notée `//` en Prolog. Par exemple, si le joueur qui a la main lorsque la position du jeu est 5 décide de diviser par 2, la nouvelle position sera 2.).

Le joueur qui a la main quand la position du jeu est l'entier 1 perd.

1. Définir un prédicat `move(+N,-M)` qui implémente la règle du jeu.
2. Définir un prédicat `gagne(+N)` qui réussit si la position `N` est gagnante.
3. Dessiner l'arbre de jeu dont la racine est la position 5, et MAX joue. Évaluer cet arbre en utilisant l'algorithme minimax à arbre de jeu (on dira que la valeur d'une feuille est 1 si MAX gagne, 0 si MIN gagne).