Examen du lundi, 5 janvier 2009

Durée: 2 heures. Documents autorisés : Notes de cours et notes personnelles, copies des transparents du cours, énoncés et solutions des TP. Les ordinateurs et des livres ne sont pas autorisés. Le barème est indicatif.

Exercice 1 (8 points)
Cinq personnes de nationalité différente habitent 5 maisons différentes (qui sont sur le même côté de la rue) de couleur différente. Chaque personne à une profession différente, un animal préféré différent, une boisson préférée différente.

- L'anglais habite la maison rouge.
- L'espagnol a un chien.
- Le japonais est peintre.
- L'italien boit du thé.
- Le norvégien habite la première maison.
- L'habitant de la maison verte boit du café.
- La maison verte vient après la maison blanche.
- Le sculpteur élève des escargots.
- Le diplomate habite la version jaune.
- Dans la troisième maison on boit du lait.
- La maison du norvégien est à côté de la maison bleue.
- Le violoniste boit du jus d'orange.
- Il y un renard dans la maison à côté de celle du docteur.
- Il y a un cheval dans la maison à côté de celle du diplomate.
- Il y a un zèbre dans la maison blanche.
- Il y a une personne qui boit de l'eau.

Écrire un prédicat `zebra(X)` qui résout ce puzzle, et qui unifie X avec le numéro de la maison dans laquelle est le zèbre. On attend de vous d'écrire un programme en GNU-Prolog, on n'attend pas que vous trouvez le numéro de la maison du zèbre vous-même !

Indication : Les maisons ont les numéros 1 à 5. Vous utiliserez plusieurs variables à domaine fini; chacune des variables dénote un numéro de maison.

Correction :

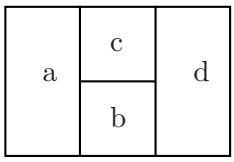
```
next(X,Y) :- X #= Y-1.
next(X,Y) :- X #= Y+1.

zebra(Zebra) :-

        Nations = [English, Spanish, Japanese, Italian, Norwegian],
        Colors = [Green, Red, Yellow, Blue, White],
        Professions = [Painter, Diplomat, Violinist, Doctor, Sculptor],
        Animals = [Dog, Zebra, Fox, Snails, Horse],
        Drinks = [Juice, Water, Tea, Coffee, Milk],

        append(Nations,Colors,X1),
        append(X1,Professions,X2),
        append(X2,Animals,X3),
        append(X3,Drinks,All),

        fd_domain(All,1,5),

        fd_all_different(Nations),
        fd_all_different(Colors),
        fd_all_different(Professions),
        fd_all_different(Animals),
        fd_all_different(Drinks),

        English #= Red,
        Spanish #= Dog,
        Japanese #= Painter,
        Italian #= Tea,
        Norwegian = 1,
        Green = Coffee,
        Green #> White,
        Sculptor #= Snails,
        Diplomat #= Yellow,
        Milk = 3,
        next(Norwegian,Blue),
        Violinist #= Juice,
        next(Fox,Doctor),
        next(Horse,Diplomat),
        Zebra #= White,

        fd_labeling(All).
```

Exercice 2 (9 point)

Écrire un programme pour colorer une carte. La carte est donnée comme une liste qui associe à chaque pays la liste des ses voisins. Par exemple,

```
[(a,[b,c]),(b,[a,c,d]),(c,[a,b,d]),(d,[b,c])]
```

décrit une carte avec les quatre pays a,b,c,d comme suit :



1. Écrire un prédicat en GNU-Prolog, et en utilisant les contraintes sur domaine fini, `coloring(L,N,C)` qui, quand L est une carte et N le nombre de couleurs disponibles, unifie C avec une affectation de couleurs aux pays telle que deux pays adjacents n'ont jamais la même couleur. Sur l'exemple de la carte au dessus une solution possible pour $N = 3$ est

   `[(a,1),(b,2),(c,3),(d,1)]`

2. Est-ce qu'il y a une symétrie dans le problème, et comment faire pour l'exploiter ? Expliquer en une phrase ou deux, on ne vous demande pas de modifier votre programme.

Correction :

```
coloring(Map,NumberColors,Coloring) :-
        makevariables(Map,NumberColors,Coloring),
        putconstraints(Map,Coloring),
        extractvars(Coloring,Vars),
        fd_labeling(Vars).

makevariables([],_,[]).
makevariables([(Country,_)|RestCountries],NumberColors,
        [(Country,Color)|RestColoring]) :-
        fd_domain(Color,1,NumberColors),
        makevariables(RestCountries,NumberColors,RestColoring).

putconstraints([],_).
putconstraints([(Country,Neighbours)|RestMap],Coloring) :-
        assoc(Country,Coloring,ThisColor),
        differentcolor(Neighbours,ThisColor,Coloring),
        putconstraints(RestMap,Coloring).

differentcolor([],_,_).
differentcolor([Country|RestCountries],OldColor,Coloring) :-
        assoc(Country,Coloring,ThisColor),
        ThisColor #\= OldColor,
        differentcolor(RestCountries,OldColor,Coloring).

assoc(Key,[(Key,Value)|Rest],Value).
assoc(Key,[_|Rest],Value) :- assoc(Key,Rest,Value).

extractvars([],[]).
extractvars([(_,Var)|Rest],[Var|RestVars]) :- extractvars(Rest,RestVars).
```

La symétrie consiste en le fait qu'on peut obtenir de nouvelles solutions en permutant les couleurs d'un solution. Pour éviter cette symétrie il suffit de fixer arbitrairement la couleur d'un pays choisi arbitrairement.

Exercice 3 (3 points)
Rendre la contrainte suivante borne-consistante, en suivant l'algorithme vu en cours :

$$X \neq 2 \wedge X = Y + Y \wedge Z = 3 * X$$
$$D(X) = [0 \dots 3], D(Y) = [0 \dots 2], D(Z) = [0 \dots 3]$$

Détailler pour chaque étape

- laquelle des trois contraintes simples est traitée;
- les nouveaux domaines des variables.

Correction :

1. *(2),* $D(X) = [0 \dots 2], D(Y) = [0 \dots 1], D(Z) = [0 \dots 3]$
2. *(1),* $D(X) = [0 \dots 1], D(Y) = [0 \dots 1], D(Z) = [0 \dots 3]$
3. *(2),* $D(X) = [0 \dots 0], D(Y) = [0 \dots 0], D(Z) = [0 \dots 3]$
4. *(3),* $D(X) = [0 \dots 0], D(Y) = [0 \dots 0], D(Z) = [0 \dots 0]$