Université d'Ottawa
Faculté de génie

School of Electrical Engineering and Computer Science

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

École de science informatique et de génie électrique

CSI 2520 Paradigmes de programmation

EXAMEN de MI-SESSION

Durée de l'examen : 75 minutes **11 février 2016, 14:30**

Professeur : Jean-Lou De Carufel

Nom de famille : _____

Prénom : _____

Numéro d'étudiant : _____

Signature : _____

Vous avez le droit d'utiliser une feuille de notes 8½ x 11, recto seulement, écrite à la main (ruban de Möbius interdit).

Question	Résultat	sur
1		6
2		4
3		3
4		5
5		2
6		6
Total		26

Question 1 [6 pts]

Considérer le programme Prolog suivant.

```
% nom, carburant, puissance (en chevaux vapeur)
locomotive(gp38-2,diesel,2000).
locomotive(ac4400cw,diesel,4400).
locomotive(2816,steam,3500).

% nom, poids brut (en livres)
wagon(boxcar,286000).
wagon(hopper,286000).
wagon(gondola,217500).
wagon(tanker,263000).

% puissance nécessaire (HP) pour tirer un poids brut en livres (GW)
pullFlat(GW,HP) :- number(GW), HP is GW//1000, !.
pullFlat(GW,HP) :- number(HP), GW is HP*1000, !.

% crée une liste de wagons de poids brut total ne dépassant pas une
% certaine limite (GW)
cars(_,[]).
cars(GW,[wagon(N,W)|T]) :- wagon(N,W),
      NGW is GW-W,
      NGW >= 0,
      cars(NGW,T).
```

Écrire une requête qui génère toutes les combinaisons valides possibles de locomotive avec sa liste de wagons. Les trois premières réponses trouvées par Prolog doivent être :

```
L = gp38-2,
LC = [] ;
L = gp38-2,
LC = [wagon(boxcar, 286000)] ;
L = gp38-2,
LC = [wagon(boxcar, 286000), wagon(boxcar, 286000)]
```

```
?-
```

_____

_____

_____

Écrire une requête qui trouve une locomotive appropriée pour tirer 3 wagons de type *tanker*. Vous **devez** utiliser le prédicat `cars` défini à la page précédente.

Première réponse possible (il y en a d'autres) :

```
L = gp38-2,
HP = 2000,
GW = 2000000

?-
```

La requête suivante ne donne pas les résultats voulus. Elle retourne ***false***.

```
?- wagon(tanker,W), locomotive(N,_,HP), GW is 3*W, pullFlat(GW,HP).
false.
```

Toutefois, la réponse attendue commence par :

```
L = gp38-2,
HP = 2000,
GW = 789000
```

Modifier la règle `pullFlat` définie à la page précédente pour que la requête ci-haut retourne les résultats voulus. Veuillez écrire la description complète du nouveau prédicat `pullFlat`, incluant les règles de la page précédente que vous désirez conserver.

Question 2 [4 pts]

La fonction de McCarthy est définie pour tous les ***entiers positifs*** de la façon suivante :

$$M(n)=\begin{cases} M(M(n+11)) & si\ n \leq 100 \\ n-10 & si\ n > 100 \end{cases}$$

Écrire un prédicat `mccarthy` en Prolog qui calcule la fonction de McCarthy et qui répond ***false*** si N est plus petit ou égal à 0.

```
% N valide ?
mccarthy(N,R) :-  N =< 0, !, fail.

% Cas de base
mccarthy(N,R) :- _____________________________________________________

                 _____________________________________________________

                 _____________________________________________________



% Cas récursif
mccarthy(N,R) :- _____________________________________________________

                 _____________________________________________________

                 _____________________________________________________

                 _____________________________________________________

                 _____________________________________________________

                 _____________________________________________________
```

Question 3 [3 pts]

Écrire un prédicat en Prolog qui teste si tous les éléments d'une liste sont différents. Vous ne pouvez pas utiliser les prédicats `all_different/1` et `all_distinct/1` déjà définis en Prolog.

Exemple:

```
?- unique([a,b,12,d]).
true.

?- unique([a,b,a]).
false.
```

```
% Cas de base (liste vide)

unique(__________________) :-     ____________________________________________

% Cas de base (un seul élément)

unique(__________________) :-     ____________________________________________

% Cas récursif

unique(____________________) :-
                                  ____________________________________________

                                  ____________________________________________

                                  ____________________________________________

                                  ____________________________________________
```

Question 4 [5 pts]

a) Considérer le programme Prolog suivant.

```
p1(X) :- p2(X,Y) , p3(Y).
p1(X) :- p3(X).
p2(X,Y) :- p4(X) , p4(Y) , X \= Y.
p2(a,a).
p3(b).
p3(c).
p4(b).
p4(a).
```

a) Dessiner l'arbre de recherche complet que Prolog construit pour répondre à la requête suivante (indiquer clairement les réponses retournées par Prolog ainsi que l'**ordre** dans lequel ces réponses sont trouvées).

```
?- p1(X).
```

(répondre à la page suivante s.v.p.)

b) Insérer une seule coupure (!) dans le programme ci-dessus de telle sorte que Prolog ne retourne qu'une seule réponse pour la requête suivante.

```
?- p1(X).
```

(Écrire la réponse pour la question 4a) ici.)

Question 5 [2 pts]

Quelle(s) définition(s) du prédicat `combinR` ci-dessous fonctionne(nt) correctement ? La requête `combinR(N, Liste, L)` doit retourner toutes les combinaisons de longueur N des éléments de la liste Liste, où les répétitions sont permises. Voici un exemple :

```
?- combineR(3,[a,b], L ).
L = [a, a, a] ;
L = [a, a, b] ;
L = [a, b, b] ;
L = [b, b, b] ;
false.
```

a)

```
combineR(0,[],[]) :- !.
combineR(N,_,_) :- N < 0, !, fail.
combineR(N,[X|L],[X|RR]):-
     NN is N-1,
     combineR(NN,[X|L],RR).
combineR(N,[_|L],R):-
     combineR(N,L,R).
```

b)

```
combineR(0,_,[]) :- !.
combineR(N,_,_) :- N < 0, !, fail.
combineR(N,[X|L],[X|RR]):-
     NN is N-1,
     combineR(NN,L,RR).
combineR(N,[_|L],R):-
     combineR(N,L,R).
```

c)

```
combineR(0,[],[]) :- !.
combineR(N,_,_) :- N < 0, !, fail.
combineR(N,[X|L],[X|RR]):-
     NN is N-1,
     combineR(NN,L,RR).
combineR(N,[_|L],R):-
     combineR(N,L,R).
```

d)

```
combineR(0,_,[]) :- !.
combineR(N,_,_) :- N < 0, !, fail.
combineR(N,[X|L],[X|RR]):-
     NN is N-1,
     combineR(NN,[X|L],RR).
combineR(N,[_|L],R):-
     combineR(N,L,R).
```

e)

```
combineR(0,_,[]) :- !.
combineR(N,_,_) :- N < 0, !, fail.
combineR(N,[X|L],[X|RR]):-
     NN is N-1,
     combineR(NN,L,[X|RR]).
combineR(N,L,[_|R]):-
     combineR(N,L,[_|R]).
```

f)

```
combineR(0,[],_) :- !.
combineR(N,_,_) :- N < 0, !, fail.
combineR(N,[X|L],[X|RR]):-
     NN is N-1,
     combineR(NN,L,[X|RR]).
combineR(N,L,[_|R]):-
     combineR(N,L,R).
```

Question 6 [6 pts]

Considérer la base de connaissances suivante.

```
foo(a, b, c).
foo(a, b, d).
foo(b, c, e).
foo(b, c, f).
foo(c, c, g).
foo(a, c, f).
```

Quels sont les résultats retournés par Prolog pour chacune des requêtes suivantes ? Si plus d'un résultat peut être retourné en utilisant un ";" donner tous les résultats. Indiquer le résultat pour toutes les variables libres.

```
?- findall(C,foo(A,B,C),L).
```

```
?- bagof([B,C],A^foo(A,B,C),L).
```

```
? -setof(C,A^B^foo(A,B,C),L).
```