Nom _____

Prénom _____

Code permanent _____

Solution de l'examen final

Date : 28 avril 2018

Titre du cours : Paradigmes de programmation

Sigle et groupe : INF2160

Enseignants : Bruno Malenfant (groupe 10) et Alexandre Blondin Massé (groupe 30)

Instructions

1) Vous avez trois heures pour répondre à l'examen ;
2) Vous avez droit à toute votre documentation papier ;
3) Vous ne devez pas dégrafer le questionnaire ;
4) Il est interdit d'utiliser un ordinateur, peu importe sa taille et sa forme (téléphone portable, agenda électronique, etc.) ;
5) Il est interdit de parler et de prêter de la documentation à un autre étudiant ;
6) Indiquez clairement vos réponses finales ;

Question	1	2	3	4	5	Total
Sur	20	20	10	20	30	100
Note						

Question 1. . . . . (20 points)

(a) (8 points) Considérez le prédicat `lis` défini par

```
lis([]) :- !.
lis([_]) :- !.
lis([X,Y|XS]) :- X @< Y, !, lis([Y|XS]).
```

Quel est le résultat de l'appel suivant ?

```
lis([a, b, c, b, a]).
```

Solution: `false`.

(b) (2 points) Proposez un nom plus significatif que `lis` qui décrit ce que fait le prédicat.

Solution: Comme le prédicat est satisfait lorsque la liste est triée (ou croissante, ou ordonnée), on peut simplement l'appeler `liste_est_triee`.

(c) (8 points) Considérez le prédicat `rcd` défini par

```
rcd([], []).
rcd([X], [X]).
rcd([X,X|XS], ZS) :- rcd([X|XS], ZS).
rcd([X,Y|YS], [X|ZS]) :- X \= Y, rcd([Y|YS], ZS).
```

Quel est le résultat de l'appel suivant ?

```
rcd([b, a, c, a, a, e, d, b, b, d], L).
```

Solution:

```
L = [b, a, c, a, e, d, b, d].
```

(d) (2 points) Proposez un nom plus significatif que `rcd` qui décrit ce que fait le prédicat.

Solution: Le prédicat est vérifié si la deuxième liste est obtenue de la première en supprimant les doublons *consécutifs*, on peut l'appeler `uniq` (en référence au programme Unix) ou `supprimer_doublons_consecutifs`.

Question 2. ..... (20 points)
Considérez le programme Prolog suivant :

```
en(0, []) :- !.
en(X, [s|N]) :- Y is X - 1, en(Y, N).

no([], 0) :- !.
no([s|N], Y) :- no(N, Z), Y is Z + 1.

d(A, [], A) :- !.
d(A, [s|B], [s|X]) :- d(A, B, X).

f(X, Y, Z) :- en(X, X1),
              en(Y, Y1),
              d(X1, Y1, Z1),
              no(Z1, Z).

g(X, Y, Z) :- en(X, X1),
              en(Y, Y1),
              d(Z1, Y1, X1),
              no(Z1, Z).
```

Pour chacun des appels ci-bas, indiquez le résultat affiché.

(a) (4 points) `en(4, A).`

Solution: `A = [s, s, s, s]`

(b) (4 points) `no([s, s], B).`

Solution: `B = 4`

(c) (4 points) `d([s, s, s], [s, s], C).`

Solution: `C = [s, s, s, s, s].`

(d) (4 points) `f(4, 3, D).`

Solution: `D = 7.`

(e) (4 points) `g(4, 3, E).`

Solution: `E = 1.`

Question 3. . . . . (10 points)
Dans Prolog, il est possible d'utiliser l'opérateur `=..` pour convertir des structures en des listes (et inversement). Voici son comportement :

```
?- f(x,y) =.. L.
L = [f, x, y].

?- f(x,y,z) =.. [f|XS].
XS = [x, y, z].

?- Term =.. [g,x,y].
Term = g(x, y).
```

Considérez maintenant le programme suivant, qui utilise cet opérateur :

```
w(a).
w(b).
w(d).

x(Z, [Z]).

y([], _, []) :- !.
y([X|XS], F, [Y|YS]) :- A =.. [F|[X,Y]], A, !, y(XS, F, YS).

z([], _, []) :- !.
z([X|XS], P, YS) :- A =.. [P|[X]], \+ A, !, z(XS, P, YS).
z([X|XS], P, [X|YS]) :- A =.. [P|[X]], A, !, z(XS, P, YS).
```

Quel sera le résultat de la requête suivante ?

```
z([a, b, c, d, e], w, R), y(R, x, S).
```

Note : Vous aurez une partie des points si vous identifiez correctement le résultat de la première clause de la requête.

Solution:

```
?- z([a, b, c, d, e], w, R), y(R, x, S).
R = [a, b, d],
S = [[a], [b], [d]].
```

Question 4. ..... (20 points)
Considérez les structures suivantes :

```
% Un arbre binaire représentant une expression mathématique
exemple1(E) :- E = produit(somme(4, 5), 8).

% Une arborescence de fichiers
exemple2(E) :- E = repertoire(
    'tp2',
    repertoire('src',
        fichier('tp2.pl'),
        fichier('test1.pl'),
        fichier('test2.pl')
    ),
    repertoire('doc',
        fichier('config.txt'),
        fichier('index.html')
    ),
    fichier('Makefile'),
    fichier('README.md')
).

% Une scène pour représenter un "monde"
exemple3(E) :- E = scene(
    lumieres(soleil, lampadaire),
    lieux(maison, hopital, gare, parc),
    routes(
        route(maison, hopital),
        route(maison, parc),
        route(parc, hopital),
        route(gare, hopital)
    )
).
```

Rappelons également quelques prédicats disponibles dans une implémentation Prolog standard :

- **functor**(*S*, *F*, *N*) qui est vérifié lorsque *S* est une structure dont le foncteur est *F* et le nombre d'arguments est *N* ;
- **arg**(*I*, *S*, *A*) qui est vérifié lorsque *S* est une structure dont le *I*-ème argument (*I* >= 1) est *A*.
- **findall**(*I*, *B*, *L*) : retourne toutes les instances possibles de *I* qui vérifient *B* et place le résultat dans la liste *L* ;
- *sum_list*(*L*, *S*) qui est vérifié si *S* est la somme des nombres apparaissant dans la liste *L*.

Proposez une implémentation des prédicats suivants :

(a) (10 points) *nombre_noeuds*(*T*, *N*) qui est satisfait si *N* est le nombre de noeuds dans la structure *T*.

(b) (10 points) *nombre_feuilles*(*T*, *N*) qui est satisfait si *N* est le nombre de feuilles dans la structure *T*.

On s'attend au comportement suivant :

```
?- exemple1(E), nombre_noeuds(E, N), nombre_feuilles(E, F).
E = produit(somme(4, 5), 8),
N = 5,
F = 3.

?- exemple2(E), nombre_noeuds(E, N), nombre_feuilles(E, F).
E = repertoire(tp2, repertoire(src, fichier('tp2.pl'), fichier('test1.pl'),
    fichier('test2.pl')), repertoire(doc, fichier('config.txt'), fichier('index
    .html')), fichier('Makefile'), fichier('README.md')),
N = 20,
F = 10.

?- exemple3(E), nombre_noeuds(E, N), nombre_feuilles(E, F).
E = scene(lumieres(soleil, lampadaire), lieux(maison, hopital, gare, parc),
    routes(route(maison, hopital), route(maison, parc), route(parc, hopital),
    route(gare, hopital))),
N = 22,
F = 14.
```

Solution:

```
nombre_noeuds(S, 1) :- functor(S, _, 0), !.
nombre_noeuds(S, N) :-
    findall(N, (arg(_, S, A), nombre_noeuds(A, N)), NS),
    sum_list(NS, NC),
    N is NC + 1.

nombre_feuilles(S, 1) :- functor(S, _, 0), !.
nombre_feuilles(S, N) :-
    findall(N, (arg(_, S, A), nombre_feuilles(A, N)), NS),
    sum_list(NS, N).
```

Question 5. . . . . (30 points)
Considérez la base de connaissances décrite à la page 10. Vous devez implémenter un certain nombre de prédicats permettant d'interroger cette base de connaissances. Il est permis en tout temps de faire appel à un prédicat d'une sous-question précédente, même si vous n'avez pas réussi à l'implémenter correctement.

(a) (5 points) Proposez un prédicat `gagnant(NomDeFamille, IDTournoi)` qui est satisfait si le joueur dont le nom de famille est `NomDeFamille` a remporté la dernière édition du tournoi `IDTournoi`.

Solution:

```
gagnant(NomDeFamille, IDTournoi) :-
    match(IDTournoi, finale, IDJoueur, _),
    joueur(IDJoueur, _, NomDeFamille).
```

(b) (5 points) Proposez un prédicat `enregistrer_match(IDTournoi, TypeMatch, Gagnant, Perdant)` qui remplace dans la base de connaissance le fait de la forme `match(IDTournoi, TypeMatch, Gagnant, Perdant)` avec de nouvelles valeurs. Vous devez vous assurer que l'information enregistrée est bien valide, c'est-à-dire que le tournoi et le joueur sont déclarés dans la base de connaissance, qu'un joueur n'a pas joué contre lui-même et que le type de match est `finale` ou `demi_finale`.

Solution:

```
enregistrer_match(IDTournoi, TypeMatch, Gagnant, Perdant) :-
    tournoi(IDTournoi, _, _),
    type_match(TypeMatch),
    Gagnant \= Perdant,
    joueur(Gagnant, _, _),
    joueur(Perdant, _, _),
    retract(match(IDTournoi, TypeMatch, _, _)),
    assert(match(IDTournoi, TypeMatch, Gagnant, Perdant)).
```

(c) (10 points) Donnez l'implémentation d'un prédicat `resultat(IDJoueur, IDTournoi, Resultat)` qui est satisfait si le joueur `IDJoueur` a obtenu le résultat `Resultat` lors de la dernière édition du tournoi `IDTournoi`. Les résultats possibles sont `gagnant`, `finaliste`, `demi_finaliste` et `autre`. Attention ! On s'intéresse au *meilleur* résultat du joueur. Par exemple, si un joueur a gagné un tournoi, on ne considère pas qu'il est finaliste ou demi-finaliste.

Solution:

```
resultat(IDJoueur, IDTournoi, gagnant) :-
    match(IDTournoi, finale, IDJoueur, _).
resultat(IDJoueur, IDTournoi, finaliste) :-
    match(IDTournoi, finale, _, IDJoueur).
resultat(IDJoueur, IDTournoi, demi_finaliste) :-
    (match(IDTournoi, demi_finale, IDJoueur, _);
    match(IDTournoi, demi_finale, _, IDJoueur)),
    (\+ resultat(IDJoueur, IDTournoi, gagnant)),
    (\+ resultat(IDJoueur, IDTournoi, finaliste)).
resultat(IDJoueur, IDTournoi, autre) :-
    joueur(IDJoueur, _, _),
    tournoi(IDTournoi, _, _),
    (\+ resultat(IDJoueur, IDTournoi, gagnant)),
    (\+ resultat(IDJoueur, IDTournoi, finaliste)),
    (\+ resultat(IDJoueur, IDTournoi, demi_finaliste)).
```

(d) (10 points) Donnez l'implémentation d'un prédicat `points(IDJoueur, NombrePoints)` qui est satisfait si le joueur `IDJoueur` a obtenu `NombrePoints` pour sa participation aux 4 tournois majeurs. Le nombre de points est calculé de la façon suivante :

- Aucun point si le joueur ne s'est pas rendu à la demi-finale ;
- 1 point si le joueur a participé à la demi-finale ;
- 2 points si le joueur a participé à la finale ;
- 4 points si le joueur a remporté le tournoi.

Solution:

```
points(IDJoueur, NombrePoints) :-
    points(IDJoueur, [aus, wim, rg, us], NombrePoints).

points(IDJoueur, IDTournoi, 4) :-
    resultat(IDJoueur, IDTournoi, gagnant).
points(IDJoueur, IDTournoi, 2) :-
    resultat(IDJoueur, IDTournoi, finaliste).
points(IDJoueur, IDTournoi, 1) :-
    resultat(IDJoueur, IDTournoi, demi_finaliste).
points(IDJoueur, IDTournoi, 0) :-
    resultat(IDJoueur, IDTournoi, autre).

points(_, [], 0).
points(IDJoueur, [IDTournoi|Tournois], NombrePoints) :-
    points(IDJoueur, IDTournoi, N1),
    points(IDJoueur, Tournois, N2),
    NombrePoints is N1 + N2.
```

Voici la base de connaissance de la question 5.
Si vous le souhaitez, vous pouvez dégrafer cette feuille.

```
% joueur(IDJoueur, Prenom, Nom, Classement)
%
% Décrit un joueur de tennis
joueur(nd, novak, djokovic).
joueur(rf, roger, federer).
joueur(rn, rafal, nadal).
joueur(am, andy, murray).
joueur(sw, stan, wawrinka).

% tournoi(IDTournoi, Nom, Terrain)
%
% Décrit un tournoi de tennis
tournoi(aus, "International d'Australie", dur).
tournoi(wim, "Wimbledon", gazon).
tournoi(rg, "Roland-Garros", terre).
tournoi(us, "International des États-Unis", dur).

% type_match(TypeMatch)
%
% On s'intéresse seulement aux finales et aux demi-finales
type_match(finale).
type_match(demi_finale).

% match(IDTournoi, TypeMatch, Gagnant, Perdant)
%
% Décrit un match de tennis qui s'est tenu lors de la dernière édition d'un
% tournoi (il s'agit de tournois annuels).  Le prédicat est dynamique, car il
% est possible d'écraser les anciens résultats d'un match avec ceux de la
% nouvelle édition.
:- dynamic(match/4).
match(aus, finale, nd, rf).
match(aus, demi_finale, rf, sw).
match(aus, demi_finale, nd, rn).
match(wim, finale, rf, am).
match(wim, demi_finale, rf, rn).
match(wim, demi_finale, am, sw).
match(rg, finale, rn, sw).
match(rg, demi_finale, rn, rf).
match(rg, demi_finale, sw, nd).
match(us, finale, nd, am).
match(us, demi_finale, nd, rn).
match(us, demi_finale, am, rf).
```