Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

Devoir 2
CSI2520 Paradigme de Programmation
Hiver 2019
6 points

Les questions 1 et 2 se réfèrent à la base de données présentée à la dernière page.

Question 1. **[2.5 points]**

Trouver la requête permettant d'obtenir l'information suivante:

a) La liste de tous les joueurs de basketball:
   `L = [jane, joe, robert].`
b) La liste des équipes de l'Ontario:
   `L = [ggs, gryphons, queens, ravens].`
c) La liste des joueurs pratiquant plus d'un sport:
   `L = [annie, suzy].`
d) En utilisant le **`findall`** obtenir la liste des joueurs d'Ottawa ainsi que leur sport:
   `L = [(paul, crosscountry), (joe, basketball), (marie, crosscountry), (marie, crosscountry), (simon, lacrosse)].`
e) La liste précédente contient le nom de `marie` à deux reprises? Utiliser le `setof` afin d'éliminer cette répétition.
   `L = [(joe, basketball), (marie, crosscountry), (paul, crosscountry), (simon, lacrosse)].`

Afin d'aider la correction, créer un prédicat `myAnswer` contenant les cinq requêtes précédentes a)-e) de facon à afficher les réponses comme suit:

```
?- myAnswer.
[jane,joe,robert]
[ggs,gryphons,queens,ravens]
[annie,suzy]
[(paul,crosscountry),(joe,basketball),(marie,crosscountry),(marie,cro
sscountry),(simon,lacrosse)]
[(joe,basketball),(marie,crosscountry),(paul,crosscountry),(simon,lac
rosse)]
true.
```

Question 2. **[2 points]**

Le prédicat ci-dessous pourrait se lire ainsi:
"*Je m'intéresse à un joueur dont le sport est le* `ski` *ou un joueur appartenant à une équipe du* `quebec`*".*

```
interest(X) :-
     sport(X, ski).

interest(X) :-
     sport(X,S),
     S \= ski,
     player(X, T),
     team(T, C),
     city(C, quebec).
```

Dessiner l'arbre de recherche complet pour le requête suivante:

```
?- interest(X).
```

Question 3. **[1.5 points]**

Soit le prédicat `area/2` permettant de calculer l'aire d'un triangle en utilisant la formule du déterminant. Par exemple:

```
?- area( [[4,0],[7,2],[2,3]], A).
A = 6.5.
```

L'aire se calcule ainsi: $A_T = \frac{1}{2}\begin{vmatrix} b_x - a_x & c_x - a_x \\ b_y - a_y & c_y - a_y \end{vmatrix}$

Notez que ce prédicat utilise deux paramètres: les coordonnées du triangle (représentées à l'aide d'une liste de listes) et l'aire à calculer.

Question 4. **[2 points]**

a) Write a predicate `skip` that returns a list which skips elements as toggled by elements in a second list. In particular, the elements of the first list are skipped until an element is encountered that is in the second list. All elements after this element are included in the result list. When another element of the first list is encountered that is also in the second list, then the elements of the first list are skipped again. In other words, the predicate toggles back-and-forth from skipping to insertion into the output list.

```
?- skip([1,2,3,4,5],[3],L).
L=[4, 5].

?- skip([b,3,5,hello,5,7,z],[5,b,a],L).
L=[3, 5, 7, z].
```

b) Write a predicate `turn` that returns a list where the order of elements is reversed as toggled by elements in a second list. In particular, the elements of the first list are reversed until an element is encountered that is in the second list. All elements after this element are included in the result list in unchanged order. When another element of the first list is encountered that is also in the second list, then the elements of the first list are reversed again. In other words, the predicate toggles back-and-forth from reverse to forward insertion into the output list.

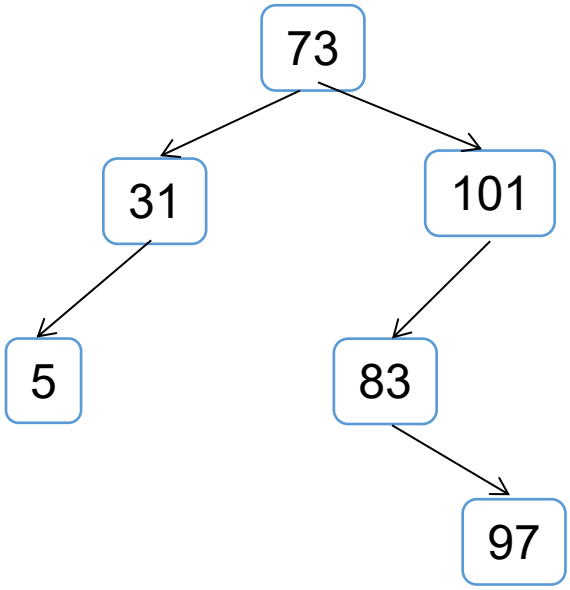
```
?- turn([1,2,3,4,5],[3],L).
L = [3, 2, 1, 4, 5].

?- turn([1,b,3,5,hello,5,7,z],[5,b,a],L).
L = [b, 1, 3, 5, 5, hello, 7, z].
```

Question 5. **[2 points]**

Voici la représentation Prolog d'un arbre binaire:

```
treeEx(X) :-
 X = t(73,t(31,t(5,nil,nil),nil),t(101,t(83,nil,t(97,nil,nil)),nil)).
```

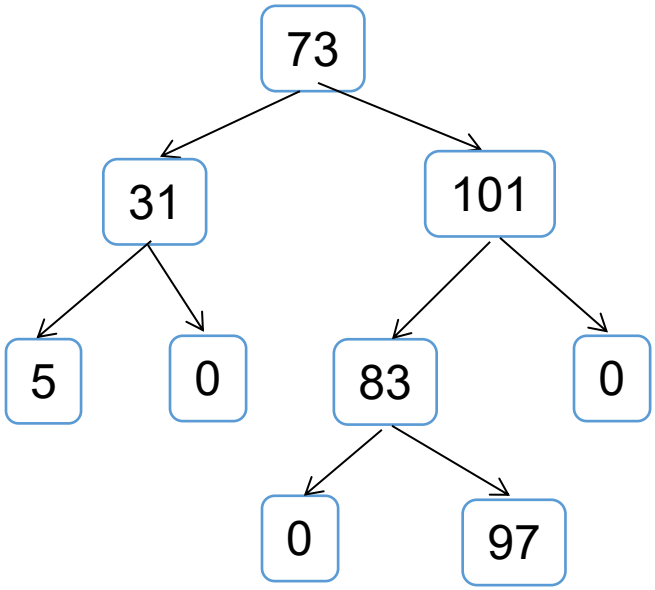


a) Créer le prédicat `single(tree,L)` permettant de trouver tous les nœuds ayant un et un seul enfant.

```
?- treeEx(T),single(T,L).
T = … , % Omitted
L = [31, 101, 83]
```

b) Créer le prédicat `singleFill(tree,V,ntree)` permettant d'ajouter un second nœud enfant à tous les nœuds ayant un seul enfant.

```
?- treeEx(T),single(T,0,L).
T = t(73, t(31, t(5, nil, nil), nil), t(101, t(83, nil, t(97,
nil, nil)), nil)),
L = t(73, t(31, t(5, nil, nil), t(0, nil, nil)), t(101, t(83,
t(0, nil, nil), t(97, nil, nil)), t(0, nil, nil)))
```



Appendice Base de données pour les Questions 1 et 2.

```
city(ottawa,ontario).
city(guelph,ontario).
city(kingston,ontario).
city(gatineau,quebec).
city(montreal,quebec).
team(ravens,ottawa).
team(ggs,ottawa).
team(gryphons,guelph).
team(queens,kingston).
team(torrents,gatineau).
team(stingers,montreal).
sport(annie,lacrosse).
sport(paul,crosscountry).
sport(suzy,ski).
sport(robert,basketball).
sport(tom,lacrosse).
sport(tim,ski).
sport(annie,ski).
sport(joe,basketball).
sport(robert,basketball).
sport(jane,basketball).
sport(marie,crosscountry).
sport(suzy,crosscountry).
sport(jack,ski).
sport(simon,lacrosse).
player(annie,gryphons).
player(tom,torrents).
player(jane,stingers).
player(marie,ggs).
player(joe,ravens).
player(jack,queens).
player(simon,ravens).
player(suzy,torrents).
player(paul,ggs).
player(marie,ggs).
player(simon,gryphons).
```