Master MLDM/DSC/CPS2 - 2018/2019 - First year
Introduction to Artificial Intelligence - Exam on Prolog

Maximum time allocated: 2h00 - No documents allowed. Scoring will depend on the cleanliness of your examination paper and the clarity of the explanations. TAKE CARE: any cheating will be severely punished and will lead to a formal complaint to the disciplinary council of the university.

1 Proof tree (5 points)

Consider the Prolog program below:

```
p3(X,Y,Z) :- p2(X,Z), t(Y).

p2(X,Y) :- s(X), q(Y).
p2(X,Y) :- q(Z), r(X,Z,Y).

s(a).  s(b). s(c).
q(a).  q(e).
t(42). t(21).

r(f,a,g).
r(g,e,d).
```

1. Draw the proof tree of the resolution of the goal: `?- p3(A,B,C).`
2. Suppose we put a cut between `s(X)` and `q(Y)` in the second clause of the program. Show, on the tree you built at the previous question, what branches are pruned during the resolution of the goal: `?- p3(A,B,C).`

2 Lists (3 points)

Define the following Prolog predicates that specify some relationships between lists.

1. `duplicate/3` where `duplicate(L1,N,L2)` is true if the elements of L1 are duplicated N times in the list L2.
2. `myreverse/2` where `myreverse(L1,L2)` is true if L2 is the list L1 reversed. To write this predicate you are **not** allowed to use the built-in predicate `append` or any equivalent predicate.
3. `compress/2` where `compress(L1,L2)` is true if L2 is equal to L1 without any consecutive duplicated values.

3 assert/retract and metapredicates (6 points)

The built-in predicates `asserta/1` or `assertz/1` can add a clause at the beginning or at the end of a certain set of clauses of the Prolog workspace. Design a predicate `addFact/2` such as `addFact(F,N)` can add the particular fact `F` at position `N` in the same set of facts of the Prolog workspace.

To understand how `addFact/2` works, suppose we load the following facts to the Prolog workspace:

```
:- dynamic p/1.
p(1). p(2). p(1). p(3).
```

Then, here are some examples of goals using `addFact/2`:

```
?- addFact(p(a),2).
true.

?- listing(p).
:- dynamic p/1.
p(1). p(2). p(a). p(1). p(3).

?- addFact(p(aaa),99).
Take care, you cannot insert this fact at position 99 as there are only 5 facts!
false.
```

4 DCG (6 points)

Consider the formal grammar of regular expressions over the alphabet 'a', 'b' and 'c':

regexp → regexp,['|'],regexp1.
regexp → regexp1.
regexp1 → regexp1,['.'],regexp2.
regexp1 → regexp2.
regexp2 → regexp3,['*'].
regexp2 → regexp3.
regexp3 →['('],regexp,[')'].
regexp3 →['a'].
regexp3 →['b'].
regexp3 →['c'].

1. Write a DCG, based on this grammar, that can be used to prove whether a regular expression is syntactically correct or not.

2. Write the Prolog goal you have to run to prove that the regular expression `(a|b)*.c*|a*` is syntactically correct.

3. Modify your DCG to build a tree representation of any regular expression. You will use those definitions:

   - The tree representation of an expression `E1|E2` is the compound term `or(E1,E2)`
   - The tree representation of an expression `E1.E2` is the compound term `and(E1,E2)`
   - The tree representation of an expression `E*` is the compound term `star(E)`
   - The tree representation of an expression `(E)` is the same as the tree representation of `E`
   - The tree representation of `a` is the compound term `letter(a)`
   - The tree representation of `b` is the compound term `letter(b)`
   - The tree representation of `c` is the compound term `letter(c)`

   For example, the tree representation of `(a|b)*.c*|a*` is the compound term:
   `or(and(star(or(a,b)),star(c)),star(a))`

4. More generally, give the Prolog clause generated from the following DCG rule after loading it into the Prolog workspace: `p(X) --> s(X,Y),[a],t(Y),[b].`

```
① duplicate3(L1, N, L2) :- duplicate4(L1, N, L2, N).

  duplicate4([ ], _, [ ], _).

  duplicate4( [_|T], 0, T2, N) :- duplicate4( T, N, T2, N).

 duplicate4 ( [H|L1], P, [H|L2], N) .
                    P>0,
                  N1 in P-1,
                 duplicate4 ( [H|L1], N1, L2, N)
```