Master MLDM/DSC/CPS2 - 2019/2020 - First year
Introduction to Artificial Intelligence - Exam on Prolog

Maximum time allocated: 3h00 - 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 (≃ 4 points)

Consider the Prolog program below:

```
p1(A) :- p2(A,X), t(X,A).

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

s(b).     s(a).     s(c).    s(d).
q(42).    q(21).
r(f).     r(c).     r(g).
t(42,b).  t(42,f).  t(21,c). t(21,d).
```

1. Draw the proof tree of the resolution of the goal: `?- p1(V).` and give all the solutions for this goal.
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, which branches are pruned during the resolution of the goal: `?- p1(V).` and give again all the solutions for this goal.

2 Lists (≃ 4 points)

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

1. `is_in/2` where `is_in(X,L)` is true if `X` is an element of the list `L`.
2. `mylast/2` where `mylast(X,L)` is true if `X` is the last element of the list `L`. To write this predicate you are **not** allowed to use the built-in predicate `last`.
3. `element_k/2` where `element_k(X,L,K)` is true if the value `X` is in position `K` in the list `L`.
4. `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.
5. `compress/2` where `compress(L1,L2)` is true if `L2` is equal to `L1` without any consecutive duplicated values.

3 Lazy evaluation (≃ 2 points)

Let us recall that the Ackermann function is defined by:

$$ack(m,n) = \begin{cases} n+1 & if\ m = 0 \\ ack(m-1,1) & if\ m > 0\ and\ n = 0 \\ ack(m-1, ack(m,n-1)) & if\ m > 0\ and\ n > 0 \end{cases}$$

1. Write the basic Prolog program that defines a predicate `ack/3` where `ack(M,N,Res)` is true if `Res` is the result of the Ackermann function of `M` and `N`.
2. Write the Prolog program that defines the predicate `lazyAck/3` where `lazyAck(M,N,Res)` is true if `Res` is the result of the Ackermann function of `M` and `N` but the Ackermann function is calculated in a lazy way, i.e. once we have calculated a value for `ack(M,N,Res)` we store this information so that it can be reused in the calculation of another value for `ack(M',N',Res')`.

4 Knowledge base modeling and querying (≃ 4 points)

Convert the following information into a Prolog program (to choose the appropriate Prolog representation, take time to read the text entirely and also look at the goals we then want to prove):

1. If a student S has some knowledge about a problem P and this student has self-confidence, then this problem is solvable.
2. If a problem P is a known problem of a scientific domain D and a student S owns a magic wand, then the problem P is solvable.
3. If a student S works on a course C and a problem P is studied in this course C, then that person P has some knowledge about this problem P.

4. If a teacher T does a course C and a student S takes notes on this course and appreciates the teacher and learns the course, then we can say this student works on this course.
5. If a teacher T does a course C and a student S owns a smartphone and the course is publicly available and the student watches it, then we can say this student works on this course.
6. If a student S captures a course C with his camera and converts it to a mp4 file F and put it on youtube, then this course is publicly available.
7. If a student owns a paper and owns a pen, and attends a course and publishes it on facebook, then this course is publicly available.
8. If a teacher T does a course C and this teacher is in a room R and a student is also in this room, then the student attends this course.
9. Smith does a course on AI. Johnson does a course on ML.
10. Paul owns a paper, Paul owns a pen, Mary owns a smartphone, Harry owns a magic wand.
11. Overfitting is a known problem of ML. Bias is a known problem of ML. Ethics is a known problem of AI.
12. John has self-confidence.
13. Mary captures the course ML with her camera.
14. Mary converts the course ML to a mp4 file f1.
15. Mary publishes the file f1 on youtube, Paul publishes the course AI on facebook.
16. Mary watches the course ML.
17. John takes notes on the course AI. John appreciates Smith. John learns the course AI.
18. The problem of overfitting is studied in the course ML. The problem of ethics is studied in the course AI.
19. Smith is in room B08. Paul is in room B08.

Considering the Prolog program you just wrote, convert the following english queries into Prolog goals:

1. What does John own?
2. Who works on which course?
3. Which course is publicly available?
4. Which problem is solvable?

5 DCG (≃ 6 points)

Consider the formal grammar of a subset of the english language:
sentence → nounphrase verbalphrase
nounphrase → determiner adjectives noun
adjectives → ∅
adjectives → adjective
adjectives → adjective adjective
verbalphrase → adverb verb nounphrase
verb → 'eats' | 'sees'
noun → 'mouse' | 'cat'
adjective → 'big' | 'small'
adverb → 'slowly'
determiner → 'the' | 'a'

1. Write a DCG, based on this grammar, that can be used to prove whether an English sentence is syntactically correct or not with respect to this grammar.
2. Write the Prolog goal you have to run to prove that the sentence `''The big cat slowly eats the small mouse''` is syntactically correct.
3. Modify your DCG to build a tree representation of any syntactically correct sentence. For example, the tree representation of `''The big cat slowly eats the small mouse''` should be the compound term:
   `s(np(d(the),adj(big),n(cat)),vp(v(eat),adv(slow),np(d(the),adj(small),noun(mouse))))`
4. More generally, give the Prolog clause generated from the following DCG rule after loading it into the Prolog workspace:

```
p(A,B) --> r(A), [a], s(B), {q42(A,B)}.
r(A)   --> p, [b], p.
p      --> [a].
s(0)   --> [].
s(N)   --> [e], s(P), {N is P+1}.
q42(99,88).    q42(77,66).
```

① P1(V)

V = -1
X = -2

P2(-1,-2), t(-2,-1)

S(-1), q(-2)

-1=a -1=b -1=

S(a) S(b) S(c)

-2=21 -2=42 4

q(21) q(42) q(42) q(21) q(42) q(21)

t(21,a) t(42,a) t(42,b) t(21,b) t(42,c)

fake fake true false fals

② in-in(X, L)

H|T

H=5

in-in(H, [H|T]).

in-in(X, [H|T]) :- in-in(X, T).

[H|T] = [1,2,9,5,6]

H=1

T = [2,4,5,6]

in-in(5, [2,5,6])

②

my-last(X, [X]).

my-last(X, [H|T]) :- my-last(X, T)

③ elem(5, [2,4,5,6])

element-K(X, [H|T], 1) :- H=X.

element-K(X, [H|T], K) :-

element-K(X, T, K1),

K1 is K-1

my-list([1,2],[2,1]) :-

my-list-2([1,2],[2,1],K)

④ my-reverse(L1, L2) :- my-reverse(L1, L2, [])

my-r3([], L2, L2).

my-r3([X|L1], L2, L) :- my-r3(L1, L2, [X|L])

[a, a, b, b]

[a, b]

Compress([X|L1], [X|L2]) :- compress(L1, [X|L2])

Compress([X|L1], [X|L2]) :- Compress([X|L1], L2).

③

① ack(0, N, N+1).

② ack(M, N, R1) :- M>0, N==0, M1 is M-1, ack(M1, 1, R1).

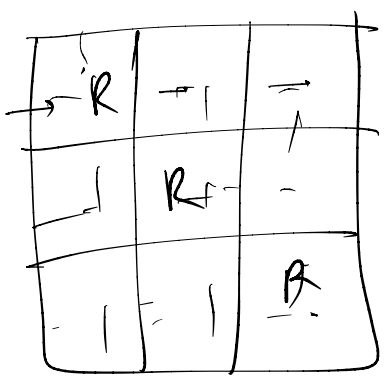
③ ack(M, N, Res) :-

M1 is M-1,

N1 is N-1,

ack(M, N1, Y),

ack(M1, Y, Res).



O(n

① <V, Ac>

① For each column, check if you same color.

② If yes, continue, else reject.

② For each row of Ac, check if there is at least one black stone.

③ If yes accept, else reject

```
lazyAck(0,N,Res) :- Res is N+1,
                    anerta(lazyAck(0,N,Res)).

lazyAck(M,0,Res) :-
        M1 is M-1,
        lazyack(M1,1,Res),
        anerta(lazyACK(M,0,Res)).

lazyAck(M,N,Res) :-
        M1 is M-1,
        N1 is N-1,
        LazyAck(M,N1,Y),
        LazyAck(M1,Y,Res),
        anerta(lazyAck(M,N,Res)).
```

```
sentence(s(np,vp)) :- nounphrase(np),
                      verb-phrase(vp)
```

① $\neg((\neg q \vee (\neg(p \vee \neg p) \wedge r) \Rightarrow s) \Rightarrow (s \vee q)$

$\Rightarrow \neg((\neg q \vee ((p \vee \neg p) \wedge r) \vee s) \Rightarrow (s \vee q))$

$\Rightarrow \neg((q \wedge (\neg p \wedge p)) \wedge r) \wedge \neg s) \vee (s \vee q))$

$\Rightarrow (\neg q \vee (p \vee \neg p) \vee \neg r \vee s) \wedge (\neg s \wedge \neg q)$

$\{\neg q, p, \neg p, \neg r, s\}$

$\{\neg s\}, \{\neg q\}$

② $\{\neg q,$ $\{\neg q, s, q\}, \{\neg p, p,$

$\forall x(\neg p(x) \vee \exists y\, q(y))$

$\forall x(\neg p(x) \vee q(f(x)))$

$\rightarrow \Rightarrow \{\neg p(x), q(f(x))\}$

$\forall x(\neg q(x) \vee \forall y\, w(y))$

$\rightarrow \Rightarrow \{\neg q(x), w(y)\}$

$\rightarrow \{\neg p(x), \neg w(x), s(x)\}$

$\rightarrow p(mary)$

$\rightarrow \neg s(mary)$

$\Rightarrow \{\neg w(mary), s(mary)\}$

$\Rightarrow \{\neg p(x), w(x)\}$

$\Rightarrow \{w(mary)\}$

$\Rightarrow \{s(mary)\}$

$\Rightarrow$ False.

$$\neg\left[\left(\neg q \vee \left(\neg(p \vee \neg p) \wedge r\right) \Rightarrow s\right) \Rightarrow (s \vee q)\right]$$

$$\Rightarrow \neg\left[\neg\left(\neg q \vee \left(\neg(p \vee \neg p) \wedge r\right) \Rightarrow s\right) \vee (s \vee q)\right]$$

$$\Rightarrow \neg\left[\neg\left(\neg q \vee \left((\neg p \wedge p) \vee \neg r\right) \vee s\right) \vee (s \vee q)\right]$$

$$\Rightarrow \neg\left[\left(q \wedge (\neg p \vee p) \wedge r\right) \wedge \neg s) \vee (s \vee q)\right]$$

$$\Rightarrow \left(\neg q \vee \left(p \wedge \neg p \vee \neg r\right) \vee \neg s\right) \wedge (\neg s \wedge \neg q$$

$\therefore (a \Rightarrow b) \wedge (\neg a \Rightarrow (b \vee c)) \wedge (\neg c \Rightarrow \neg b) \wedge ((b \wedge c) \Rightarrow \neg a) \wedge (c \Rightarrow a)$

$\Rightarrow (\neg a \vee b) \wedge (a \vee b \vee c) \wedge (c \vee \neg b) \wedge (\neg b \vee \neg c \vee \neg a) \wedge (\neg c \vee a)$

$\{\neg a, b\}$

$\{a, b, c\}$

$\{b, c\}$

$\{c\}$

$\{c, \neg b\}$

$\{\neg b, \neg c, \neg a\}$

$\{\neg c, a\}$

$\{a\}$

$\{\neg b, \neg c\}$

$\{\neg b\}$

$\{\neg a\}$ False

$$\neg\left[\left(\neg q \vee \left(\neg(p \vee \neg p) \wedge r\right) \Rightarrow s\right) \Rightarrow (s \vee q)\right]$$

$$\Rightarrow \neg\left[\neg\left(\neg q \vee \left(\neg(p \vee \neg p) \wedge r\right) \Rightarrow s\right) \vee (s \vee q)\right]$$

$$\Rightarrow \neg\left[\neg\left(\neg(\neg q \vee\right.\right.$$