CSE560 Midterm Exam

Name: ____________________

Page 1	/8
Page 2	/4
Page 3	/7
Page 4	/7
Page 5	/5
Page 6	/5
Page 7	/5
Page 8	/4
Page 9	/3

Syntax of Datalog

Question 1 (2 marks)

Consider the constants *tom* and *sally*, and the 0-ary predicate $raining$ and the 2-ary predicate $mother(\_,\_)$. What are all of the legal **atomic** clauses (also called facts) that can be written. Do not use variables.

Answer:
```
raining
mother(tom,sally)
mother(sally,tom)
mother(tom,tom)
mother(sally,sally)
```

Question 2 (2 marks)

Give two non-atomic clauses (also called rules) that are legal. Do not use variables.

Answer:
```
Many possible answers.  Here are two:
raining <- mother(tom,sally)
mother(tom,sally) <- mother(sally,tom)
```

Question 3 (2 marks)

If we add the function $motherof(\_)$, give an *atomic* clause that is legal and that uses this function. Do not use variables.

Answer:
```
Many possible answers.  Here is one:
mother(motherof(tom),tom).
```

Question 4 (2 marks)

With the addition of the function $motherof(\_)$, how many different atomic clauses can we now write? Do not use variables.

Answer:
```
infinite number, as we can have arbitrary embedding of motherof.
```

Intended Interpretation

Question 5 (2 marks)

Annie is Beth's mother, and Beth is Corine's mother. Imagine that you are a knowledge engineer and have the intended interpretation where *a* maps to Annie, *b* maps to Beth, and *c* maps to Corine, and *mother(X,Y)* is true if X is Y's mother.

The following two clauses are clearly true of the intended interpretation.

mother(a,b)
mother(b,c)

Mark each of the following clauses as to whether it is true or false in the intended interpretation.

mother(c,a)

Answer:
```
false
```

mother(b,a)

Answer:
```
false
```

mother(a,b) $\leftarrow$ *mother(b,c)*

Answer:
```
true, since both parts are true.
Note that ‘<-’ does not capture any sense of causality.
```

mother(a,b) $\leftarrow$ *mother(c,a)*

Answer:
```
true, as true <- false is always true.
```

mother(c,a) $\leftarrow$ *mother(a,b)*

Answer:
```
false, as false <- true is always false.
```

Question 6 (2 marks)

Now say we add the function *motherof(X)* that will indicate who the mother of X is.

Write down one fact that the knowledge engineer should add to the knowledge base to capture the full meaning of *motherof*.

Answer:
```
mother(motherof(X),X)
```

Interpretations

Question 7 (2 marks)

Consider the language with constants *a* and *b*, and the 0-ary predicate $raining$ and the 1-ary predicate $girl(\_)$. For the domain $D = \{x, y, z\}$, how many different ways can an interpretation map the constants (how many different ϕ's are there)? Explain your answer.

Answer:
```
a -> x,y or z
b -> x,y or z
So, 3*3
```

Question 8 (1 marks)

How many ways can an interpretation map the predicate $raining$ (how many different ways can π map $raining$)?

Answer:
```
raining -> T/F   2 ways
```

Question 9 (2 marks)

How many ways can an interpretation map the predicate $girl(\_)$. Give one of the mappings in full.

Answer:
```
girl(x) -> T/F
girl(y) -> T/F
girl(z) -> T/F
so 2*2*2
girl(x) -> T, girl(y) -> T, girl(z) -> T
```

Question 10 (1 marks)

How many interpretations are there altogether?

Answer:
```
3*3 * 2 * 2*2*2
```

Question 11 (1 marks)

If we add the variable X to our syntax, does this change the number of interpretations? Why or why not?

Answer:
```
No.  Interpretations do not specify the mapping of variables, that is
what rho does.
```

Question 12 (1 marks)

How many of the interpretations from Question 10 are models of $KB = \{\}$.

Answer:
```
All of them
```

Question 13 (3 marks)

How many of the interpretations are models of $KB = \{girl(a), girl(b), raining\}$

Answer:
```
Case 1: phi(a) = phi(b)
   3 different phi's
   Without loss of generality, say phi(a) = x
     then girl(x) must be true,
     and does not matter whether girl(y) and girl(z) is true or false
   raining must map to true
   So, 2*2 pi's
Case 2: phi(a) != phi(b)
   6 different phi's (3 ways to map a and 2 ways to map b)
   Without loss of generality, say phi(a) = x and phi(b) = y
      then girl(x) and girl(y) must be true.
      and does not matter whether girl(z) is true or false
   raining must map to true
   So 2 pi's
Altogether 3*2*2 + 6*2 models
```

Semantic Proofs

Question 14 (2 marks)

What does $KB \models \alpha$ mean? Phrase your answer in terms of interpretations (not models).

Answer:

```
For any interpretation I that makes KB true, I also makes alpha true.
```

Question 15 (1 marks)

What does it mean if α does not logically follow from KB? Does this mean that α is false in our intended interpretation? Explain your answer.

Answer:

```
It means there is at least one model of KB that makes alpha false.
But, there might be models of KB that make alpha true.   So, the intended
interpretation might make alpha true or false.
```

Question 16 (3 marks)

Consider the following KB

$p.$

$q \leftarrow p.$

Use a truth table to prove that q logically follows from KB ($KB \models q$). Make sure you explain why your truth table shows $KB \models q$.

Answer:

```
p q q<-p Model
T T  T    yes
T F  F    no
F T  T    no
F F  T    no

Only model is p and q are both mapped to T.
So, in all models, q is true.  So, q follows form KB.
```

Derivations

Question 17 (2 marks)

What is the most general unifier of the following.

p(a,b)   p(Y,Z)

Answer:

```
Y/a Z/b
```

---

p(a,b,X)   p(Y,Z,Z)

Answer:

```
Y/a Z/b X/b
```

p(q(X),q(a))   p(Z,Z)

Answer:
```
Z/q(a) X/a
```

Question 18 (3 marks)

Consider the following KB.

1 $parent(X,Y) \leftarrow mother(X,Y)$
2 $parent(X,Y) \leftarrow father(X,Y)$
3 $ancestor(X,Y) \leftarrow parent(X,Y)$
4 $ancestor(X,Z) \leftarrow parent(X,Y) \wedge ancestor(Y,Z)$
5 $mother(amy,bob)$

Give a top-down derivation (from query to KB) of $ancestor(amy,bob)$. Show the details of the derivation (which rule used, and the substition).

Answer:
```
yes <- ancestor(amy,bob)
   use: ancester(X,Y) <- parent(X,Y)
   sub X/amy Y/bob
yes <- parent(amy,bob)
   use: parent(X,Y) <- mother(X,Y)
   sub X/amy Y/bob
yes <- mother(amy,bob)
   use mother(amy,bob)
yes <-
```

Question 19 (2 marks)

Give a bottom-up derivation. Show all steps.

Answer:
```
mother(amy,bob)   KB                                                   1
parent(amy,bob)   1 and parent(X,Y) <- mother(X,Y) with X/amy Y/bob    2
ancester(amy,bob) 2 and ancester(X,Y) <- parent(X,Y) with X/amy Y/bob  3

No other facts can be added.
ancestor(amy,bob) is true since in C.
```

Question 20 (1 marks)

Since the computer only does derivations, why is it necessary to have semantics?

Answer:
```
We need to make sure that the derivations that the computer makes are true
and that it is finding all possible derivations.  We use semantics to check
this.
```

Defining Knowledge

Question 21 (2 marks)

Using definite clauses (rules and facts), define **pulloutone(Item,List,Remainder)**, which is true if **Item** is in **List**, and **Remainder** is the **List** minus **Item**. Use prolog list notation `[H|T]` (with `[]` as the empty list).

Answer:

```
pulloutone(I,[T|Rest],Rest).
pulloutone(I,[T|Rest],New) <-
    putoutone(I,Rest,Remainder)
    New = [T|Remainder]
```

Question 22 (1 marks)

What will the answer be for the following query? (You do not have to give a derivation; just give the answer.)

```
?pulloutone(p(X),[p(a),p(b),r,q(c),Z,p(d)],L).
```

Answer:

```
X=a L=[p(b),r,q(c),Z,p(d)]
```

Question 23 (1 marks)

What are the other answers (if we did an exhaustive search for all of the possible answers)?

```
?pulloutone(p(X),[p(a),p(b),r,q(c),Z,p(d)],L).
```

Answer:

```
X=b     L=[p(a),r,q(c),Z,p(d)]
Z=p(X)  L=[p(a),p(b),r,q(c),p(d)]
X=d     L=[p(a),p(b),r,q(c),Z]
```

Question 24 (3 marks)

Define a predicate `maxInList(V,L)` to determine the maximum value V in list L. The predicate should be designed to work when L is specified (an input varaible).

You can assume that there are no duplicate numbers in the list. Use prolog list notation, and prolog's built in numbers. The only built in predicate you can use is $X > Y$, but it must be placed so that both X and Y are bound when prolog evaluates it. Any other predicate that you use, you need to define.

Base case: Think of whether the base case should be a list of length 0 or a list of length 1.

Answer:

```
maxInList(Y,[Y]).
```

Recursive case: Your solution will be graded three criteria. First, is the first solution it returns correct. Second, does it only return a single solution (does it fail when you try to get a second solution)? Third, how efficient is your code? For a list of length n, is it spending time proportionally to n or to n^2?

Answer:

```
maxInList(Answer,[Top|Rest]) :-
    maxInList(Y,Rest),
    larger(Y,Top,Answer).

larger(A,B,A) :-
    A > B.

larger(A,B,B) :-
    B > A.
```

Search

Question 25 (1 marks)

What is the role of search in converting a top-down proof procedure into a reasoning procedure.

Answer: Search is how the non-determinism of the top-down proof procedure is resolved. It ensures that all choices for what rule are used to resolve with the first atom of the answer clause is eventually searched.

Question 26 (3 marks)

Write a *breadth*-first search algorithm in python, called `BreadthSearch(start,goal)`, where `start` is the start node and `goal` is the goal node. It should take two arguments: a start node and a goal node. It should return true if there is a path from start to goal.

In your code, use the data structure `connected`, which is a list of all connections between nodes. For example, it might look like:
`connected = [['a', 'b'], ['b', 'd'], ['d', 'a'], ['d', 'c']]`
Note that connections are unidirectional, so the pair `['a','b']` means you can go from 'a' to be 'b', but not necessarily from 'b' to 'a'.

Structure your code around the use of the variables `frontier` to keep track of the frontier, and `neighbors` to track all of the neighbors of the current node. You do not need to build in cycle-checking.

If you do not remember the syntact for a particular command, you can mix in pseudo-code. Make sure you indicate which lines are pseudo-code from what is python.

Answer:

```
def BreadthSearch(start,goal):
    frontier = [start]
    while frontier:
       top = frontier[0]
       if top = goal:
           return True
       neighbors = []
       for c in connected:
           if c[0] = top:
               neighbors.append(c[1])
       frontier = frontier[1:]+Neighbors(top)
    return False
```

Explain how to change your code into a depth-first search.

Answer: Change how neighbors are added to the frontier.
