Midterm Test — October 24, 2018
Solutions and Grading Guide

Duration: 80 minutes
No aids allowed.
Total marks: 75.

Name:

Student Number:

1)	/10
2)	/10
3)	/5
4)	/10
5)	/20
6)	/10
7)	/10
Total	/75

1. [10 points 2 points each] For each of the following statements indicate whether it is *true* or *false*:

   (a) An agent is an entity that perceives its environment and acts upon it. **True**

   (b) In the Turing Test, a judge must determine whether he/she is chatting with a computer or a human. **True**

   (c) A chess playing agent is operating in a *fully observable* environment. **True**

   (d) A self-driving vehicle is operating in a *discrete* environment. **False**

   (e) Resolution is sound, i.e., if a clause c can be derived by resolution from a set of clauses S, then S entails c. **True**

   (f) Resolution is complete, i.e., if a clause c is entailed by a set of clauses S, then c can be derived from S in a forward chaining resolution proof. **False**

   (g) $(p, q, \neg r)$ is a Horn clause. **False**

   (h) In SLD-resolution, only negative clauses are derived. **True**

   (i) An advantage of the forward chaining procedure for Horn clauses is that it is goal directed. **False**

   (j) In general, determining whether a sentence is entaiiled by a set of sentences in first-order logic is undecidable. **True**

2. [10 points, 2 points each] Translate the following English sentences into first-order logic:

   Give:
   2 points for completely correct
   1.5 points if minor problems, e.g. dropping outside $\forall$
   1 point for partially correct
   0.5 points if a few elements correct
   0 points if completely wrong

   (a) Every employee has a supervisor.

   $$\forall E.employee(E) \supset \exists S.supervises(S, E)$$

   (b) No student likes a test that is hard.

   $$\neg\exists S.\exists T.student(S) \wedge test(T) \wedge hard(T) \wedge likes(S, T)$$
   or $\forall S.\forall T.student(S) \wedge test(T) \wedge hard(T) \rightarrow \neg likes(S, T)$

   (c) There is no question such that no student knows the answer.

   $$\neg\exists Q.question(Q) \wedge \neg\exists S.student(S) \wedge knowAnswer(S, Q)$$
   or $\forall Q.question(Q) \rightarrow \exists S.student(S) \wedge knowAnswer(S, Q)$

   (d) All but one of the students are happy.

   $$\exists S.student(S) \wedge \forall T.student(T) \wedge T \neq S \rightarrow happy(T)$$

   (e) A professor is happy if he/she belongs to no committees.

   $$\forall P.professor(P) \wedge \neg\exists C.committee(C) \wedge belongs(P, C) \rightarrow happy(P)$$

3. [5 points] Say whether or not the following pairs of expressions are unifiable, and show the most general unifier for each unifiable pair:

   Give:
   For a) and c):
   1 points for saying whether or not it is unifiable, *plus*
   1 point if they have correct MGU
   0.5 points if they are missing a part of the MGU or if substitutions need to be applied in sequence
   0 points if completely wrong

   For b): 1 point for saying it is not unifiable

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

   yes, $\{X = a, Y = b, Z = b\}$

   (b) $p(f(X, X), a)$ and $p(f(Y, f(Y, a)), a)$

   no

   (c) $p(g(f(V)), g(U))$ and $p(X, X)$

   yes, $\{X = g(f(V)), U = f(V)\}$

4. [10 points] Implement the following description of a *grandfather* relation using one or more Prolog rules:

   > X is the father of Y and Y is the father of Z or X is the father of Y and Y is the mother of Z implies X is the grandfather of Z .

```
grandfather(X, Z) :-  father(X, Y), father(Y, Z).
grandfather(X, Z) :-  father(X, Y), mother(Y, Z).
```

   or

```
grandfather(X, Z) :-  father(X, Y), father(Y, Z)
                      ;
                      father(X, Y), mother(Y, Z).
```

   10 marks minus number of individual errors (missing terms, wrong variable order, wrong variable, etc.)

5. [20 points] Suppose that we have the following knowledge base (KB) represented as a set of clauses about three elephants, Sam, Clyde, and Oscar:

   (1) $(pink(sam))$ Sam is pink.

   (2) $(gray(clyde))$ Clyde is gray.

   (3) $(likes(clyde, oscar))$ Clyde likes Oscar.

   (4) $(pink(oscar), gray(oscar))$ Oscar is either pink or gray.

   (5) $(\neg pink(oscar), \neg gray(oscar))$ Oscar is not both pink and gray.

   (6) $likes(oscar, sam)$ Oscar likes Sam.

   (a) Prove that the KB does *not* entail that *Oscar is pink*. That is, show that there is an interpretation that satisfies the KB but does not satisfy this conclusion.

   Let $\mathcal{I} = \langle D, \phi, \psi, v \rangle$ where
   $D = \{sam, clyde, oscar\}$
   $\phi(x) = x$ for all $x \in D$
   $\psi(pink) = \{sam\}$
   $\psi(gray) = \{clyde, oscar\}$
   $\psi(likes) = \{\langle clyde, oscar \rangle, \langle oscar, sam \rangle\}$
   v can be any variable assignment.

   $\mathcal{I}$ trivially satisfies (1), (2), (3), and (6).
   $\mathcal{I} \models$ (4), since $\mathcal{I} \models gray(oscar)$.
   $\mathcal{I} \models$ (5), since $\mathcal{I} \models \neg pink(oscar)$.
   However, $\mathcal{I} \not\models pink(oscar)$, since $\psi(pink)(\phi(oscar)) = false$.
   QED

   Mark out of 8. Give
   8 if all correct
   7 correct, but interpretation not given in full
   6 correct, but interpretation not given in full, and no argument for satisfaction of (4) and (5)
   5 some incomplete argument based on interpretation
   4 if they argue that resolution can't produce the empty clause
   3 resolution refutation but incomplete argument
   2 buggy resolution or informal argument
   0 if completely wrong.

(b) Use *resolution refutation* to prove that *some grey elephant likes some pink elephant*. Give the representation of the query in first-order logic, and its translation in clausal form. You can display the proof either in tree form or as a sequence of statements. Clearly indicate the parents of each resolvent.

Query: $\exists X.\exists Y.grey(X) \wedge pink(Y) \wedge likes(X, Y)$.
Negated query in clausal form: $(\neg grey(X), \neg pink(Y), \neg likes(X, Y))$ (7).

Proof:
(8) R[2, 7a]{X = clyde } $(\neg pink(Y), \neg likes(clyde, Y))$
(9) R[3, 8b]{Y = oscar} $\neg pink(oscar)$
(10) R[4a, 9] $grey(oscar)$
(11) R[1, 7b]{Y = sam } $(\neg grey(X), \neg likes(X, sam))$
(12) R[6, 11b]{X = oscar} $\neg grey(oscar)$
(13) R[10, 12] []

Mark out of 12. Give
12 if all correct
10 if minor errors in prrof or query
8 if correct query and clausal form, but incomplete or buggy proof
6 if buggy query and correct clausal form and some proof steps, or correct query and clausal form but no proof
4 if buggy query and correct clausal form for it
2 if buggy query and incorrect clausal form for it
0 if completely wrong.

6. [10 points] Consider the following Prolog program:

```
p(a). p(b).
q(a). q(c).

r1(X):- p(X), q(X).

r2(X):- p(X).
r2(X):- q(X).

r3(X):- p(X), \+ q(X).

r4(X):- \+ q(X), p(X).

r5(X):- p(X), !, q(X).
r5(X):- X = b.
```

For each of the following queries, say whether it succeeds or fails; also if it succeeds and contains the variable `X`, give *all* the values of `X` for which the query succeeds:

a) `?- r1(X).`

```
X = a ;
false.
```

b) `?- r2(X).`

```
X = a ;
X = b ;
X = a ;
X = c.
```

c) `?- r3(X).`

```
X = b.
```

d) `?- r4(X).`

```
false.
```

e) `?- r5(b).`

```
false.
```

Mark each subquestion out of 2.
For a) , b), c), and d), subtract 1 for each extra or missing value.

7. [10 points]
   (a) Give a definition of a Prolog predicate `permutation(L1, L2)` that holds if and only if the list `L2` is a permutation of the list `L1`. `L2` is a permutation of `L1` if the two lists contain exactly the same components, but possibly in a different order. Include definitions of all non-built-in predicates that you use.

```
permutation([],[]).
permutation(L, [H|T]):- append(PL,[H|SL],L), append(PL,SL,NL),
                        permutation(NL,T).
```

Another version uses an auxiliary predicate `remove(L,X,R)` which behaves like `append(PL,[X|SL],L), append(PL,SL,NL)`.

The definition should work when there are duplicates.

5 marks for a correct solution
4 marks for mostly correct solution, with minor errors
3 marks for partially correct solution with base case
2 marks for partially correct solution without base case
1 marks for having some minor elements of solution or only base case
0 otherwise

(b) Suppose that there are 5 people (John, Wally, Mary, Helen, and Ming) who work in an office with 5 cubicles in a single row. We want to assign the workers to cubicles in a way that satisfies their constraints. Let's represent a cubicle assignment as a list of 5 workers where the first component is the name of the worker who will get the first cubicle, the second component is the name of the worker who will get the second cubicle, and so on. The constraints are that Ming does not want to be next to Wally and that John does not want to have the first cubicle. Give a definition of a Prolog predicate `cubicleAssignment(L)` that holds if and only if the list `L` is a cubicle assignment that satisfies all the constraints. Include definitions of all non-built-in predicates that you use. In your answer, you may use the `permutation` predicate specified in (a).

```
cubiculeAssignment(L) :- permutation([john, wally, mary, helen, ming],L),
                         \+ nextTo(ming,wally,L),
                         \+ L = [john|_].

nextTo(A,B,L):- append(_,[A,B|_],L); append(_,[B,A|_],L).
```

5 marks for a correct solution
4 marks for minor error
3 marks for buggy, but make a permutation/list of names with member plus some other elements
2 marks for multiple serious bugs, but have some constraints
1 marks for having some minor elements of solution, e.g. using permutation
0 otherwise