THE UNIVERSITY OF AUCKLAND

FIRST SEMESTER, 2004
Campus: City

COMPUTER SCIENCE

Foundations of Artificial Intelligence

(Time allowed:)

NOTE: Attempt all questions.
Put the answers in the boxes below the questions.

MARKS	
SECTION A: PREDICATE CALCULUS	(out of 35)
SECTION B: PROLOG	(out of 15)
SECTION C: SEARCH	(out of 20)
SECTION D: PLANNING	(out of 30)
TOTAL:	(out of 100)

SURNAME:	
FORENAME(S):	
STUDENT ID:	

CONTINUED

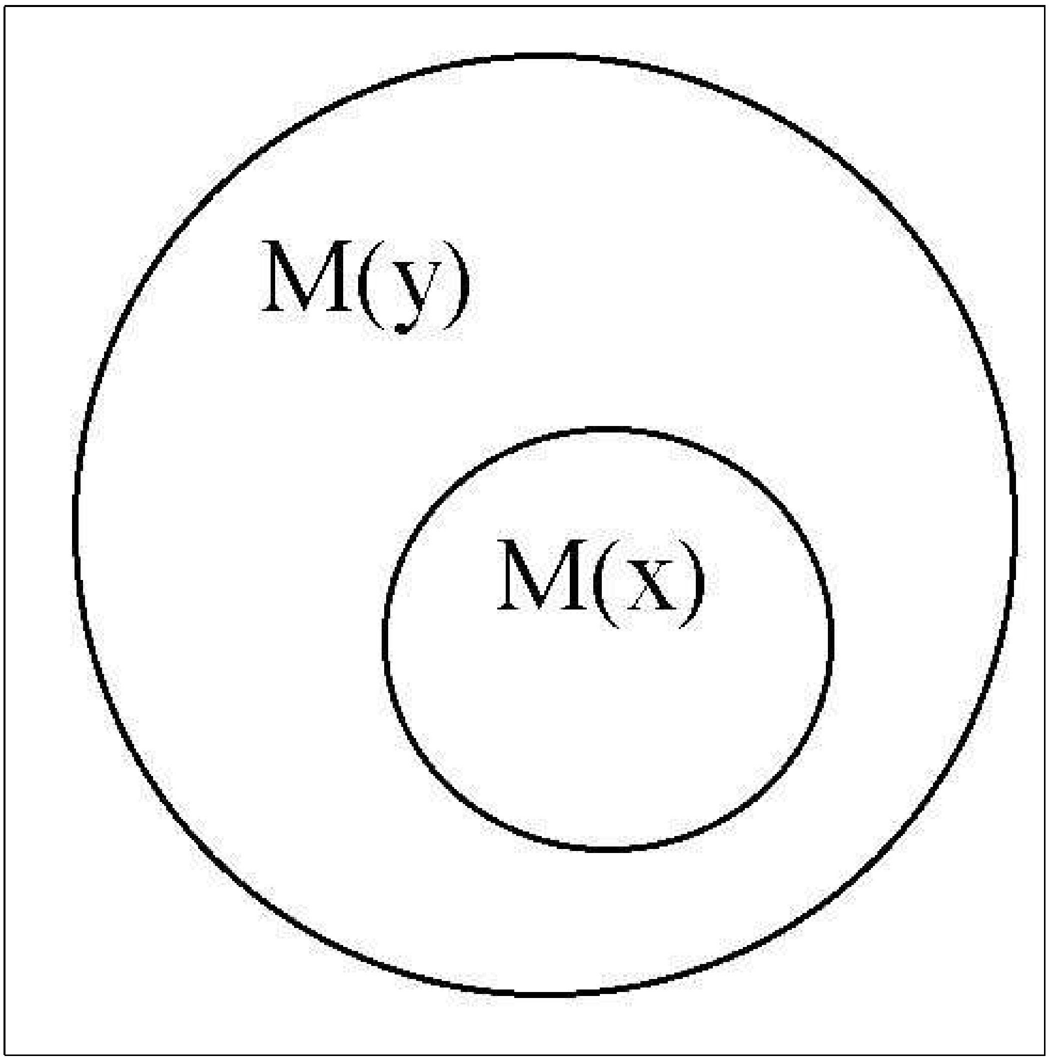
Surname: ______________________________

Forename(s): ______________________________

Section A: Predicate Calculus

1. Draw the venn diagram of M(x) and M(y) if $x \vdash y$.

[2 marks]



CONTINUED

Surname: ______________________________

Forename(s): ______________________________

2. Are the following sentences satisfiable, unsatisfiable or valid?

   a. $A \rightarrow \neg(A \rightarrow B)$

   b. $A \vee (A \rightarrow B)$
   c. $A \vee \neg A$
   d. $A \wedge \neg A$
   e. $A \wedge \neg B$

[5 marks]

a. satisfiable
b. valid
c. valid
d. unsatisfiable
e. satisfiable

CONTINUED

Surname: ______________________________

Forename(s): ______________________________

3. What are the English sentences these formulas represent?

   a. $\forall x\ likes(X, icecream) \rightarrow likes(X, brocolli)$
   b. $\forall x\ likes(X, icecream) \wedge likes(X, brocolli)$
   c. $\exists x\ likes(X, icecream) \rightarrow likes(X, brocolli)$
   d. $\exists x\ likes(X, icecream) \wedge likes(X, brocolli)$
   e. $\forall x \exists\ y\ likes(x, y)$
   f. $\exists x \forall y\ likes(x, y)$

[12 marks]

a. Anyone who likes ice cream also likes brocolli.
b. Everyone likes ice cream and brocolli.
c. Someone doesn't like ice cream or does like brocolli.
d. Someone likes ice cream and brocolli.
e. Everyone likes someone.
f. Someone likes everyone.

CONTINUED

Surname: ______________________________

Forename(s): ______________________________

4. Give the predicate calculus for these sentences using the predicates: cat(x), furry(x), animal(x), dog(x), frog(x), and slimy(x).

```
All cats are furry.
Some frogs are slimy.
If an animal is furry then it is a cat or a dog.
```

[6 marks]

$\forall x\ cat(x) \rightarrow furry(x)$
$\exists x\ frog(x) \wedge furry(x)$
$\forall x\ animal(x) \wedge furry(x) \rightarrow cat(x) \vee dog(x)$

CONTINUED

Surname: ______________________________

Forename(s): ______________________________

5. Write a definition for sister-in-law using only married(x,y), brother-of(x,y), and sister-of(x,y). [4 marks]

$\forall x, y \exists z \; sister - in - law(x, y) \;\leftrightarrow\; (brother - of(x, z) \wedge married(z, y)) \vee (married(x, z) \wedge sister - of(z, y)$

6. Unify the following formulas

```
a. p(x)             p(y)
b. p(x)             p(f(y))
c. p(x)             d(y)
d. p(x)             p(a)
e. p(x)             p(f(a))
f. p(a)             p(f(x))
```

[6 marks]

a. x/y
b. x/f(y)
c. failed
d. x/a
e. x/f(a)
f. failed

CONTINUED

Surname: ______________________________

Forename(s): ______________________________

Section B: Prolog

7. Translate the logic expression into the appropriate Prolog code. Use the same predicates for the Prolog code as appears in the logic expression.

   $human(Socrates) \wedge human(Plato) \wedge \forall x human(x) \rightarrow mortal(x)$

   [4 marks]

   human(socrates).
   human(plato).
   mortal(X) :- human(X).

8. Translate the Prolog code into the appropriate predicate calculus expression. Use the same predicates for the logic expression as appears in the Prolog code.

```
wealthy(X) :- inherited(X, money).
wealthy(X) :- won(X, Y), typeOf(Y, lottery).
```

   [2 marks]

   $\forall x \exists y \ (inherited(x, Money) \vee won(x, y) \wedge typeOf(y, Lottery)) \rightarrow weathy(x)$

CONTINUED

Surname: ____________________

Forename(s): ____________________

9. Translate the Prolog code into the appropriate predicate calculus expression. Use the same predicates for the logic expression as appears in the Prolog code.

```
popular(X) :- won(X, lotto).
idle(X) :- won(X, lotto).
```

[2 marks]

$\forall x\ won(x, Lotto) \rightarrow popular(x) \wedge idle(x)$

Surname: ____________________

Forename(s): ____________________

10. Given the following Prolog code:

```
a(X) :- p(X).
a(X) :- q(X).
p(X) :- r(X), s(X).
q(X) :- t(X), u(X).
r(1).  r(2). s(2). t(2). t(3). t(4). u(2). u(4).
```

What answer/s will you get if you give Prolog the query "a(A)" and repeatedly ask for more answers? Put down all the Prolog responses (e.g., bindings, etc.) in the space below (if responses are repeated then state how often they are repeated). [3 marks]

A=2;A=2;A=4

11. Given the following Prolog code:

```
a(X) :- p(X).
a(X) :- q(X).
p(X) :- not(r(X)), s(X).
q(X) :- t(X), not(u(X)).
r(1). s(2). t(2). t(3). t(4). u(2). u(4).
```

What answer/s will you get if you give Prolog the query "a(A)" and repeatedly ask for more answers? Put down all the Prolog responses (e.g., bindings, etc.) in the space below (if responses are repeated then state how often they are repeated). [3 marks]

A=3

CONTINUED

Surname: ____________________

Forename(s): ____________________

12. Given the following Prolog code:

```
a(X) :- p(X).
a(X) :- q(X).
p(X) :- r(X), !, s(X).
q(X) :- t(X), u(X).
r(1). r(5). s(5). t(2). t(3). t(4). u(2). u(4).
```

What answer/s will you get if you give Prolog the query ”a(A)” and repeatedly ask for more answers? Put down all the Prolog responses (e.g., bindings, etc.) in the space below (if responses are repeated then state how often they are repeated). [1 mark]

A=2;A=4

CONTINUED

Surname: ______________________________

Forename(s): ______________________________

Section C: Search

13. In the search tree below, the **G** node is a goal node, the rest of the nodes are not goal nodes.
(1) How many nodes would be created by the breadth-first search algorithm?
(2) How many nodes would be created by the iterative-deepening search algorithm?
(3) List the nodes created by the breadth-first algorithm in their order of creation.
(4) List the nodes created by the iterative-deepening algorithm in their order of creation.

```
             ....a....
            /         \
         b             c
        / \           / \
      d     e      f     G
     / \   / \    / \
    h   i j   k  l   m
```

Figure 1: **Search Tree**

[8 marks]

(1) 13 nodes. (2) 11 nodes. (3) a b c d e f G h i j k l m. (4) a a b c a b d e c f G.

Surname: ____________________

Forename(s): ____________________

14. The **A*** search algorithm uses **f**-values to order its traversal of the search tree.
(1) Intuitively, what does a node's **f**-value represent?
(2) How does it compute this value?
(3) What must be true of the components of this computation to guarantee that the algorithm finds the optimal path to a goal node? [12 marks]

(1) It represents an estimate of the cost of the shortest path from the initial state to a goal node going through this node.
(2) It is the sum of the length of shortest path from the initial state to this node plus an estimate of the shortest path from this node to a goal node.
(3) The cost of the path from the initial node to this node must be the minimal cost path from the initial state to that node and the estimate must not be greater than the true cost of the minimum-cost path from that node to any goal node.

CONTINUED

Surname: ______________________

Forename(s): ______________________

Section D: Planning

15. States represent what the agent/planner knows about the world. There are two main approaches to representing what we know about the state of the world. One is called the *Closed World Assumption (CWA)*, the other is called the *Open World Assumption (OWA)*.
(1) Why do we make these types of assumptions?
(2) What is the main difference between these assumptions?
(3) For each of the two assumptions, what is implicitly represented about the state?
(4) For each of the two assumptions, what is explicitly represented about the state?

[4 marks]

(1) They cut down on the number of literals we need to explicitly store in order to specify a state.
(2) The difference between these assumptions is whether or not we assume the planner knows the truth value of every literal, i.e., whether the planner is omniscient. CWA assumes the planner does know the truth value of every literal (both positive and negative). OWA does not assume the planner knows the truth value of every literal.
(3) CWA - false conditions are implicitly represented, OWA - unknown conditions are implicitly represented.
(4) CWA - true conditions about the world are explicitly represented, OWA - known conditions about the world are explicitly represented.

CONTINUED

Surname: ______________________________

Forename(s): ______________________________

16. Goals (and operator preconditions) represent what states the agent/planner desires to bring about in the world. There are two main types of conditions that can occur in a goal expression. One type is object-level conditions and the other type is meta-level conditions.
(1) What is the difference between these?
(2) Why do we sometimes need to have meta-level preconditions?
(3) Give an example of a meta-level condition we might find in the precondition of a blocksWorld's operator.

[4 marks]

(1) Object-level conditions describe something that's testable in a state, and meta-level conds describe something that is not testable about states.
(2) To ensure that the operators perform ”legal” actions in the world (e.g., cannot put a block on top of itself).
(3) neq(Block, Destination).

CONTINUED

Surname: ____________________

Forename(s): ____________________

17. The effects of an operator represent how the application of an operator changes the world. Effects can contain both positive and negative object-level literals.
(1) What do positive object-level effect literals say about how the operator affects the world?
(2) What do negative object-level effect literals say about how the operator affects the world?
(3) Give an example of a negative object-level condition we might find in the effects of a blocksWorld's operator. [4 marks]

(1) They are added to the world description.
(2) The are deleted from the world description.
(3) not(clear(B))

CONTINUED

Surname: ____________________

Forename(s): ____________________

18. Planners need to generate successor plans from a given plan.
    (1) Describe how the successors are generated in a progressive planner.
    (2) Describe how the successors are generated in a regressive planner. [4 marks]

(1) Given a plan, there is an associated progressed state and given that state the planner calculates all *applicable* operators, i.e., all operator instantiations where all preconditions are satisfied by the progressed state. The successor plans are all those plans which have the given plan plus one of the applicable operators (placed just before the finish pseudo-operator step).
(2) Given a plan, there is an associated set of regressed goals and given that set the planner calculates all *relevant* operators, i.e., all operator instantiations where at least one of their effects satisfy a regressed goal. The successor plans are all those plans which have the given plan plus one of the relevant operators (placed just after the start pseudo-operator step).

CONTINUED

Surname: ______________________________

Forename(s): ______________________________

19. Planners need to determine when a plan solves a given problem. Assuming that our plans have *start* and *finish* psuedo-steps.
(1) Describe how, in a progressive planner, the plans are tested for being a solution to the given problem.
(2) Describe how, in a regressive planner, the plans are tested for being a solution to the given problem.

[4 marks]

(1) In a progressive planner, a plan is tested for being a solution by checking whether the plan's associated progressed state satisfies all of the preconditions of the *finish* pseudo-step.
(2) In a regressive planner, a plan is tested for being a solution by checking whether the plan's associated set of regressed goals is satisfied by the effects of the *start* pseudo-step.

CONTINUED

Surname: ____________________

Forename(s): ____________________

20. Often, domain actions need to have meta-level preconditions. In our discussions of planners, there has only been one type of meta-level precondition mentioned.
(1) How would progressive planners handle the testing of that type of meta-level precondition? Can it always be tested? Why or why not?
(2) How would regressive planners handle the testing of that type of meta-level precondition? Can it always be tested? Why or why not?

[4 marks]

(1) Progressive planners test *neq*/2 meta-level preconditions by checking whether its two arguments have the same value. The *neq*/2 meta-level preconditions can always be tested when checking to see if the operator is applicable. This is because all of the variables that appear in the *neq*/2 meta-level preconditions also appear in that operator's object-level preconditions and for an operator to be applicable, all of the object-level preconditions must unify with the fully instantiated progressed state causing all of those variable to become fully instantiated.
(2) Regressive planners must check whether both variables have been instantiated or have been unified with each other (i.e., they **must** instantiate to the same value). If the former then the values can be checked. If the latter then regardless of their eventual value this condition must fail. Otherwise, the *neq*/2 meta-level precondition **cannot be** tested at that time to see whether that relevant operator needs to be ruled out. Unlike with the progression planner, not all of the variables need to be bound when checking whether an operator is relevant. This is because not all of the operator's effects need to unify with the set of regressed goals for the operator to be considered relevant. This means that even if all of the regressed goals are fully instantiated, it is still possible for some of the variables appearing in the effects to remain uninstantiated. If the variables are uninstantiated then it is not always possible to test for inequality (i.e., *neq*/2) of the variables. For example, given *neq(X,Y)*, where *X* and *Y* are uninstantiated, how can the planner know whether *X* and *Y* will eventually be instantiated to the same value or not? It can't!

CONTINUED

Surname: ____________________________________

Forename(s): ____________________________________

21. The **start** and **finish** pseudo steps were introduced in partially-ordered plans so that the problem's initial situation and top-level goals could be handled the same way as were operator preconditions and effects. For the most part this works, however, there is one important time when a pseudo-step cannot be handled just like an ordinary operator-step.
(1) For which pseudo-step is this true?
(2) When is it true?
(3) Why is it true?

[6 marks]

There turned out to be two answers for this question. I had only thought of the first, but both are worth full marks (provided you answer them fully and correctly).

<u>**First answer**</u>

(1) The start pseudo-step.
(2) When it will produce a negative condition to be used by a later step via a causal link.
(3) This is true because unlike real steps, negative conditions are not represented in the effects of the start pseudo-step. The effects of the start pseudo-step represent a state description (using the Closed World Assumption) and consequently only positive conditions are explicitly represented while the effects of real steps represent a state update description and consequently the negative conditions represent positive conditions to be removed from the current state description.

<u>**Second answer**</u>

(1) Both pseudo-steps.
(2) When one of its causal links is potentially clobbered by another step.
(3) This is true because potential clobberers of causal links that only involve real steps can be ordered before the causal link's producer or after the causal link's consumer. However, when a causal link involves a pseudo-step at least one of these alternatives is eliminated.

CONTINUED

Surname: ______________________________

Forename(s): ______________________________

Scratch sheet

CONTINUED

Surname: ____________________

Forename(s): ____________________

Scratch sheet

CONTINUED

Surname: ____________________

Forename(s): ____________________

Scratch sheet