UNIVERSITY OF DUBLIN
TRINITY COLLEGE

Faculty of Engineering and Systems Sciences

DEPARTMENT OF COMPUTER SCIENCE

B.A.(Mod.) Computer Science

Junior Sophister Examination **Trinity Term 2001**

3BA2 Artificial Intelligence

Thursday, 24th May Goldsmith Hall 9.30 - 12.30

Tim Fernando and Mike Brady

Answer **five** questions, at least **one** from each section. Each question is worth 20 marks.

Section A

1. (a) What are *Turing machines* and what can they do? Give an example of a problem some Turing machine can solve but which no finite state machine can.

   (b) What is the *Church-Turing thesis* and what would it take to invalidate it?

   (c) What is a *non-deterministic* Turing machine, and what does non-determinism have to do with intelligence (artificial or otherwise)?

   (d) Can we turn every non-deterministic Turing machine into a deterministic Turing machine? (That is, for every non-deterministic Turing machine is there a deterministic Turing machine with the same input/output behavior?) Why or why not?

2. (a) Give a simple example where reasoning with equality can lead to a loop.

   (b) What is the *Unique Names Assumption* (UNA) and to what extent does it hold in Prolog? What do *unification* and the *occurs check* have to do with it?

   (c) How can we introduce equalities such as `ulysses=odysseus` without violating UNA?

3. Recall that a graph with arcs given by a predicate `arc` can be searched for a node `Node` satisfying `isGoal(Node)` via the following generic search procedure `search(Frontier)` (in Prolog):

```
search([Node|FrontierRest]) :-  isGoal(Node).
search([Node|FrontierRest]) :-
        findall(X,arc(Node,X),Children),
        addToFrontier(Children,FrontierRest,FrontierNew),
        search(FrontierNew).
```

   (a) Recall that a search typically starts with the frontier initialized to a list consisting of a single node — say, `start`

   ```
   | ?- search([start]).
   ```

   So why then does the `search` procedure take a whole frontier (i.e. a list of nodes) as an argument, and not just a single node?

   (b) What extensions/modifications need to be made above to get A*? (Three or four informative lines in English should suffice.)

   (c) Simplify the code above to turn it into depth-first search. Can we get by, by keeping track of a single node rather than a whole frontier?

   (d) Is depth-first search *admissible* (when conceived as an instance of A*)? Why or why not?

4. Consider the knowledge base

```
a :- b,d.
a :- c.
b :- c.
b :- d.
c :- b.
d.
```

   (a) Evaluate the query `?-a` top-down (as in Prolog) and bottom-up (as described in our text and in lecture) against this knowledge base.

   (b) Building on the code defining `search` in question 3 above, show how a successful answer to the query `?-a` can be computed through `search([[a]])`, by taking a node to be a list of goals (from `a,b,c,d`) that have to be proved. That is, define the predicates `arc`, `isGoal` and `addToFrontier` so that the query `?-search([[a]])` yields a successful derivation of `a`.

5. (a) What is *STRIPS* and what is the *STRIPS assumption*?

   (b) What is a *situation* and what is the *situation calculus*?

(c) How are STRIPS and the situation calculus similar? How do STRIPS and the situation calculus differ?

6. (a) What is an *integrity constraint* and what is it good for?

(b) What is *abduction*? How does it relate to *deduction* and *explanation*?

(c) What role do integrity constraints play in abduction?

SECTION B

7. Write a Prolog program to count the numbers of distinct terms in an unsorted list. For example, for the list `[a,1,4,3,as(f),4,1,3]`, your program should return: `[1-2,3-2,4-2,a-1,as(f)-1]` (that is, 'there are two occurrences of 1, two occurrences of 3, two occurrences of 4, one occurrence of 'a' and one occurrence of the term 'as(f)').

Comment on the order of magnitude efficiency of your program (i.e. is its run time proportional to n, n log n, n^2, etc.), and briefly discuss Kowalski's well-known aphorism "Program = Logic + Control".

8. Give an account of the similarities and differences that exist between Prolog when treated as a logic formalism and Prolog as a programming system.

Describe, with examples, the problems that can arise and explain what features of the language and its implementation account for these similarities and differences.