CS 188 Fall 1992 — Introduction to AI — Stuart Russell — Midterm examination

You have 1 hr 20 min. The exam is open-book, open-notes.
You will not necessarily finish all questions, so do your best ones first.
Write your answers in blue books. Hand them all in.

80 points total (1 point per minute). Panic not.

1. **(15 pts.) Definitions**
   Provide brief, *precise* definitions of the following:
   (a) Completeness (of an inference procedure)
   (b) Validity (of a sentence)
   (c) Agent
   (d) Procedural attachment
   (e) Heuristic search

2. **(22+3 pts.) Search**
   The search algorithms we have studied all begin at the initial state and work forward until a goal is found. In this question we will look at a different approach, called *bidirectional search* (BDS). The idea is to search simultaneously from the start and from the goal until the two searches join up. We (well, to be honest, you) will begin by implementing BDS as two breadth-first searches. The following is the code for breadth-first search (not especially optimized!):

```
(defun bfs (open)
  (cond ((null open) 'fail)
        ((goalp (car open)) (get-path (car open)))
       (t (bfs (append (cdr open) (successors (car open)))))))
```

   (Here `get-path` returns the path from the start to the goal if called with the goal node.)

   (a) (3) BDS will be called with two open lists (`open1` and `open2`) containing the start and goal nodes. Describe (in English) what the termination condition should be.
   (b) (2) How will a solution be extracted and returned once the search terminates successfully?
   (c) (2) What condition must be satisfied by the `successors` function in order for the solution to be executable from the initial state?
   (d) (9) Write BDS in Common Lisp (you may use appropriate auxiliary functions without writing code for them, as long as you say what they do.)
   (e) (3) Suppose the search space has branching factor b and the shortest path from start to goal has length d. Approximately how many nodes does BDS examine?
   (f) (3) Indicate briefly (in English) how you would modify BDS to use a heuristic function.
   (g) (3, extra credit) Can you do this so that BDS is still admissible?

3. (10 pts.) Logic

Represent the following sentences in predicate calculus:

(a) (2) Calculators contain at least one battery.

(b) (3) All calculators have a 4 button below the 7 button.

(c) (3) HP calculators are cheaper than Sharp calculators.

(d) (2) Only nerds have calculators.

4. (11 pts.) Inference

(a) (2) Give the Modus Ponens inference rule for propositional calculus.

(b) (6) Use resolution to prove that it is sound.

(c) (3) Is resolution complete for propositional calculus? Why (not)?

5. (10 pts.) Situation calculus, knowledge representation

(a) (6) Recall our discussion of shopping earlier in the course. One of the actions involved was *buying*. Let $buy(x, y, z)$ denote the action of x buying object y from z. Write situation calculus axioms to describe this action's effects (including payment!). You will need to use additional predicates, which you should define (in English). What frame axioms would you need (no need to write them out)?

(b) (4) Suppose you are given a blocks-world problem with completely specified start and goal states. Would it be better to do situation calculus planning by forward chaining or backward chaining? Why?

6. (12 pts.) Games against nature

Consider a traditional search problem such as the 8-puzzle. Actions (left, right, up, down) are usually *deterministic*, i.e., they always work. Hence we can build a normal search tree and a solution is just a sequence of moves. In this question, we will consider what happens when actions sometimes have unpredictable effects, but we still need to produce solutions that will get us to the goal. (You can assume thatthe outcome of an action is observable.)

(a) (3) Suppose that each action *fails* with some fixed probability $p < 1$, i.e., sometimes nothing happens when we try a move. What does a solution plan look like now?

(b) (2) Suppose we had been using A* with Manhattan distance to find optimal solutions, i.e., shortest paths. Now, instead of "shortest", optimal means "using the smallest number of attempted moves on average." Would we need to change the algorithm at all?

(c) (3) Suppose that each action fails with probability jp ($jp < 1$), where j is the number on the tile being moved (maybe some tiles are heavier, for example). What change would you need to make to get optimal solutions?

(d) (4) Suppose now that rather than just failing, actions sometimes "mess up" by moving some other adjacent tile into the empty square. Briefly indicate what problems this would cause (particularly regarding what constitutes a "solution"), and how you might deal with them. [This question has a lot of ramifications. Don't spend too much time trying to deal with all of them: just propose something reasonable even if not perfect.]