| | | |
|---|---|---|
| CS 188 | Introduction to AI | |
| Fall 1991 | 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.

Panic not.

1. **(15 pts.) Definitions**
   Provide brief, *precise* definitions of the following:
   (a) Soundness (of an inference procedure)
   (b) Term (in predicate calculus)
   (c) Inheritance
   (d) Frame axiom
   (e) Admissible (algorithm)

2. **(13 pts.) Search**
   (a) (3) What elements are necessary to formally define a specific search problem?
   (b) (5) Define algebraic equation-solving (e.g., "Solve $x^2y^3 = 3 - \sin xy$" for x) as a search problem. (You need to provide clearly-specified examples of each of the elements necessary for the definition, but need not provide an exhaustive list).
   (c) (3) Give a reasonable heuristic function for equation-solving, and calculate its value for the state given in b).
   (d) (2) Would hill-climbing be an appropriate method for equation-solving using your heuristic?

3. **(13 pts.) Logic and semantic nets**
   (a) (2) Inheritance information in semantic nets states that all members of a given class P have a particular value Y for a given slot Q. Write this as a rule in predicate calculus.
   (b) (3) Now consider the information content in Kelly's blue book: that, for example, 1973 Dodge Vans are worth \$575. Suppose all this information (for 11,000 models) is encoded as logical rules, as in part a). Write down three such rules, including that for 1973 Dodge Vans. How would you use the rules to find the value of a *particular* car (eg JB, which is a 73DodgeVan) given a backward-chaining theorem prover (such as `prologx`)?
   (c) (3) Compare the time efficiency of the backward-chaining method for solving this problem with the inheritance method used in semantic nets.
   (d) (3) Explain how forward chaining allows a logic-based system to solve the same problem efficiently, assuming that the KB contains only the 11,000 rules about price.
   (e) (2) Describe a situation in which neither forward nor backward chaining on the rules will allow the price query for an individual car to be handled efficiently.
   (f) (0) *No credit*: Can you suggest a solution enabling this type of query to be solved efficiently in all cases in logic systems? (As a hint, you might want to consider the fact that many semantic net systems inherit information from a *prototype member* of the class, e.g., Typical73DodgeVan.)

4. **(20 pts.) Situation-calculus planning**
   In this question we will consider the problem of planning a route from one city to another. The basic action taken by the robot is `(go x y)` which takes it from city x to city y provided there is a direct route. `(DirectRoute x y)` is true iff there is a direct route from x to y; you can assume that all such facts are already in the KB. The robot begins in Savannah and must reach Gary.

(a) (1) Write a suitable logical description of the initial *situation* of the robot.

(b) (1) Write a suitable logical query whose solutions will provide possible paths to the goal.

(c) (3) Write a rule describing the `go` action.

(d) (2) Is a frame axiom needed? Why (not)?

(e) (8) Now suppose that following the direct route between two cities consumes an amount of *fuel* equal to the *distance* between the cities. The robot starts with fuel at full capacity.

i. Augment your representation to include these considerations;

ii. write a new rule or rules describing the `go` action; and

iii. describe the initial situation.

Your action description should be such that the query you specified above will still result in feasible plans.

(f) (5) Now suppose some of the vertices are also *gas stations*, at which the robot can fill its tank using the `(fillup)` action. Extend your representation to include gas stations and write *all* the rules needed to *completely* describe the `(fillup)` action. (You can assume the robot doesn't have to worry about money.)

**5. (22 pts.) Lisp, Game-playing**

The following code implements minimax search in a two-player game:

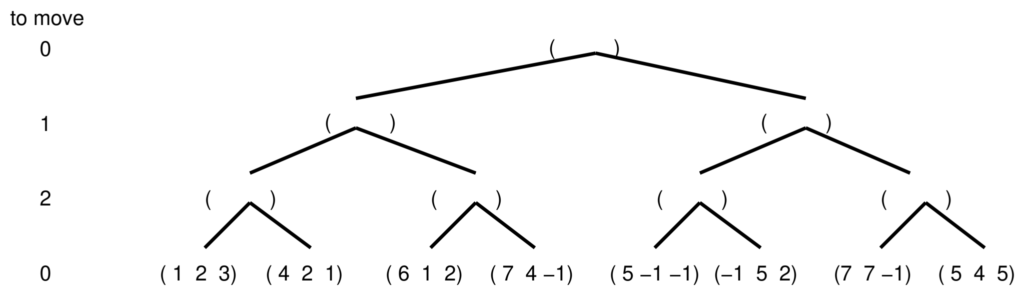
```
(defun choose (side state limit)
  (the-biggest #'(lambda (s) (backed-up-value (opponent side) state 1 limit))
               (successors state)))

(defun backed-up-value (side state depth limit)
  (if (= depth limit)
      (evaluate side state)
      (apply (if (oddp depth) #'min #'max)
             (mapcar #'(lambda (s) (backed-up-value (opponent side) s (1+ depth) limit))
                     (successors state)))))
```

(a) (3) `(the-biggest f l)` returns that item x in list l that has the highest value of (f x). Write `the-biggest`.

(b) (3) In this and subsequent parts, we will consider the problem of search in a *three-player* game (you can assume no alliances are allowed for now). We will call the players 0, 1 and 2 for convenience. The first change is that `evaluate` will return a list of three values, indicating (say) the likelihood of winning for players 0, 1 and 2 respectively.

Copy and complete the following game tree by filling in the backed-up value triples for all remaining nodes:




(c) (11) Rewrite `choose` and `backed-up-value` so that they work correctly for the three-player game. `choose` should be rewritten in such a way that it generates an *iterative deepening* search (never mind why!). (You may define additional functions if needed. You may find `the-biggest` useful also.)

(d) (5) (open-ended) Discuss the problems that might arise if players could form and terminate alliances as well as make moves "on the board". Indicate briefly how these problems might be addressed.