CS 188 Intro to AI

Fall 1992 Stuart Russel Midterm 2

You have 1 hr 20 min. The exam is epen-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):
   an inference procedure is complete iff all logical entailments of a given theory can be proved by the procedure.

   (b) Validity (of a sentence):
   a sentence is valid iff it is true in all possible worlds under all interpretations.

   (c) Agent:
   a physical object that can be analysed as having perceptions and producing actions.

   (d) Procedural attachment
   a method for connecting a non-logical solution mechanism to specific types of goals and assertions in order to speed up theorem-proving or generate side-effects.

   (e) Heuristic search
   any search mechanism employing domain-specific information to guide the search process (usually, information about the estimated quality of partial solutions).

2. (22+3 pts.) Search

   (a) (3) Termination condition: when the two open lists have a non-empty intersection of states - remember that the open lists usually contain nodes, which may differ on aspects othere than the state; if the lists just contain states, then there is no way to get two different paths from the common state! Once there is some intersection, it only disappears again if the stae in common is expanded by one of the two searches. Hence we can get away with checking if the node being expanded is on the othere list.

   (b) (2) Solution Extraction: applkying get-path to each of the two nodes sharing state, we obtain two halves of the solution. We need to reverse the path from the goal node and append it to the end of the path from the start node. We also need to make sure that the common node does not appear twice!

   (c) (2) If we use successors to generate nodes from the goal state, we must be sure that the steps are reversible: i.e., if A is a successor of B then B must be a successor of A.

   (d) (9) The key issues are:
      i. Failure occurs if either of the lists become empty.
      ii. For intersection must check states, not nodes.
      iii. We have to check intersection after each expansion, not after one expansion of both lists (otherwise the two searhces might cross over. Since the expansions are going to alternate, it's easiest to

switch the arguments each time:

```
(defun bds (open1 open2 &optional (inorder? t))
   (cond ((or (null open1) (null open2)) 'fail)
           ((check-for-solution open1 open2 inorder?))
           (t (bds open2 (append (cdr open1) (successors (car open1))) (not inorder?)))))

(defun check-for-solution (open1 open2 inorder?))
   (let ((join2 (car (member (car open1) open2 :key #'node-state :test#'equal))))
      (when join2
         (let ((join1 (car open1)))
            (if inorder?
               (append (get-path join1) (revese (cdr (get-path join2))))
               (append (get-path join2) (revese (cdr (get-path join1)))))))))
```

We seren't too worried about the nittyh-gritty details of the solution extraction, which can be rather annoying, as long as the basic aspects mentioned above appear somewhere.

Owns(x, y, s) means x owns y in situation s
Funds(x, m, s) means x funds m in situation s.
(for all)xyzpmn Owns(z, y, s) (or) Price(z, y, p) (or) Funds(x, m, s) (or) >= (m, p) (or) Funds(z,n,s) (implies)
   Owns(x,y result(buy(x,y,z), s)) (or)
   Funds(x, -(m,p), result(buy(x,y,z),s)) (or)
   Funds(z, +(n,p), result(buy(x,y,z), s)) (or)
   (not)Owns(z, yy, result(buy(x,y,z), s))
We need frame axioms to make sure that other people's funds, and ownership of all other objects, are unchanged, along with any other situation-dependent predicates there might be.

(b) (4) Generally speaking, not much to choose between them since 1) blocks world actions are reversible 2) the branching factor is therefore about the same in both directions. This might not be the case for an incompletely specified goal, though.

6. (12 pts.) Games against nature
   This question was generally answered pretty well. Thee answers to part d) were especially good, considering that this is a current research issue. Most people in the class seem to have more sense than most AI researches.

(a) (3) Solution plan: in the original solution, each step was a single action. Since the only form of fiailure has no side effects, we can keep trying again and will eventually succeed. So each step becomes a "loop until success" fo the same action. (On average, it will take 1/(1-p) tries to make a move.)

(b) (2) If we just treat each operator as a "loop until success", then because the average total solution cost is the sum of the average step costs, the same algorithm will apply proveded we just multiply the g and h costs by the appropriate amount. In fact, even if we left them untouched the solution returned would be the same.

(c) (3) If each action fails with probability jp (jp < 1), where j is the nujmber on the tile being moved, then we need to recognize that some moves are more expensive. Each step cost going into the g cost is multipled by some factor (in fact, 1/(1-jp)) and the h estimates for each tile are multiplied by the same factor (since the total expected cost to move 3 squarees, say, is 3 times the total expected cost to move one square).

(d) (4) If actions somtimes "mess up" by moving some other adjacent tile into the empty square, then we are in a real bind. Now we can't just keep trying until success, because we might mess up the pattern totally.

Since we can observe action outcomes, we can tell when an undesired eevent has ocurred, and try to undo it. This means out plan has to have a repair action inserted for every step, conditonal on the various possible failures. Unfortunately, the repairs can fail too, leading us even further away from the "mailine" solution path, so we need repairs for repairs ad infinitum. If failures are sufficiently unlikely, we could terminate this regress at some point, leaving us with a conditional solution plan that succeeds with high probability.

This process is pretty ugly, on the whole, because we have to anticipate so many events that may never actually occur and each has to be dealt with in its own way. The clean way to think about it is the following. Since, in principle, a sequence of unexpected outcomes can lead us to any state in the state space, we could just calculate, for each state, what is the right thing to try (the most likely to get us to a solution quickly) and store it. Dynamic programming techniques are designed for this kind of problem; for example, we can build up such a table by starting from the goal state and working back.
A more sphisticated approach is to put planning steps into the plan itself. Thus, we could have a main line plan just as in part a), and conditional branches so that if unexpected results occur, the search mechanism is reinvoked to solve the the new problem that arises. Obviously, we could combine this mechanism with the ideas in the previous paragraph in various degrees, depending on the failure probabilities.