Midterm solutions

1. (15 pts.) Definitions

(a) *Soundness*: an inference procedure is sound if its conclusions are logically entailed by its premises. (Truth-preservation of course follows from this).

(b) *Term*: a syntactic construct that denotes an object; can be a constant symbol, a variable, or a function symbol applied to terms.

(c) *Inheritance*: the process whereby properties of individual objects are inferred from their membership in a class.

(d) *Frame axiom*: an axiom stating that certain properties of the situation remain unchanged as a result of an action.

(e) *Admissible algorithm*: guaranteed to return the shortest path to a goal if one exists. Admissible heuristic is really a derived term, since if $h \leq h^*$ then A^* will be admissible.

2. (13 pts.) Search

(a) (3) Initial state, goal test and operator descriptions.

(b) (5) Initial state is the equation to be solved.
Goal test is that one side of the equation contains just the target variable and the other side contains no occurrences thereof.
Operators: multiply both sides by an expression; square both sides; simplify an expression; etc.

(c) (3) Sum of the depths of nesting of all occurrences of the target variable, minus 1; depth of nesting means number of enclosing operators to reach the outermost expression. On the LHS the occurrence is depth 2 (squaring and multiplication); on the RHS 3 (multiplication, sin, and minus); so $h = 5$.

(d) (2) Hill-climbing would be inappropriate *for this heuristic* since equation-solving often involves steps that appear to make the equation more complex (such as multiplying out brackets).For a perfect heuristic hill-climbing is ideal.

3. (13 pts.) Logic and semantic nets

(a) (2) `(All x (IF (P x) (Q x Y)))` or possibly
`(ALL x (IF (IsA x P) (Q x Y)))`.

(b) (3)
```
(All x (IF (73DodgeVan x) (Price x 575)))
(All x (IF (87RollsCorniche x) (Price x 77900)))
(All x (IF (91Yugo x) (Price x 43c)))
```
The query is obviously `(Price JB x)`; however we first need to store the fact that JB is a 1973 Dodge Van: `(73DodgeVan JB)`.

(c) (3) The query unifies with the conclusions of every single rule in the KB, so the premises of each rule must be tested. Hence the complexity is O(n) where n is the number of rules. Inheritance simply follows the IsA pointer from JB and gets the value from the class 73DodgeVan, i.e., O(1).

(d) (3) If we use the rules in the forward direction, then when we store `(73DodgeVan JB)` the fact `(Price JB 575)` will be asserted; then we can use `knownx` to get the answer. Since only this particular rule premise unifies with the initial fact about JB, any efficient indexing system can ensure O(1) triggering. (Indexing does not help with backward chaining because the rules *do* match the query!)

(e) (2) If the KB contains a lot of rules for each class of cars (eg all the spare parts information) then asserting `(73DodgeVan JB)` will cause many hundreds of facts to be asserted if the rules are used in the forward direction.

(f) (0) The inheritance basically asks the following series of questions: What is the class JB belongs to? What is the prototype of that class? What is the price of the prototype? With a *single* rule:

```
(All w x y z (IF (AND (IsA x y)
                      (Prototype w y)
                      (Price w z))
                 (Price x z)))
```

plus a prototype for each model, we can answer all price queries in O(1) time by backward chaining.

4. (20 pts.) Situation-calculus planning

This question was misunderstood by many students, who took a wumpus-style approach (despite the question heading and frequent use of the word "situation". The wumpus style is OK (although many wumpusers tried to describe when `(go x y)` should be done, rather than what it does), except when you want to get a plan in b): the wumpus-style axioms will only tell you what time you will arrive!

(a) (1) `(At Savannah S0)` where S0 is the initial situation.

(b) (1) `(At Gary s)`

(c) (3)

```
(All x y s (IF (AND (At x s)
                    (DirectRoute x y))
               (At y (Result (go x y) s))))
```

(d) (2) No frame axiom is needed since At is the only property of situations and it is always changed.

(e) (8)

i. `(At x f s)`: robot is at x with fuel level f in s
`Capacity`: constant, the maximum amount of fuel
`(distance x y)`: the distance between x and y
`(- a b)`: the numerical difference between a and b
`(< a b)`: a is numerically less than b.

ii.
```
(All x y f s (IF (AND (At x f s)
                      (DirectRoute x y)
                      (< (distance x y) f))
                 (At y (- f (distance x y)) (Result (go x y) s))))
```

iii. `(At Savannah Capacity S0)`

Other representations are possible, e.g., using a fuel predicate or a fuel function; these representations require two rules, one for concluding location and one for fuel; on the other hand, they are more realistic.

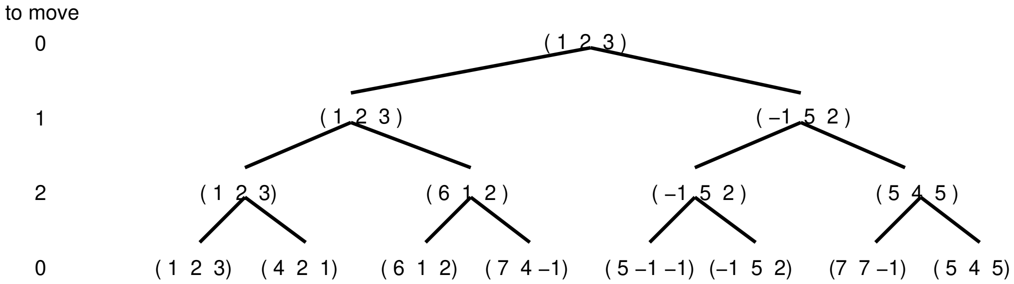
(f) (5) `(GasStation x)`: x is a gas station
`(All x f s (IF (AND (At x f s) (GasStation x)) (At x Capacity (Result (fillup) s))))`

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

(a) (3) The key things here are to remember to return x and not (f x); and to realize that in Lisp applying a function variable requires using `apply` or `funcall`.

```
(defun the-biggest (f l &aux biggest (value most-negative-double-float))
  (dolist (x l biggest)
    (let ((v (funcall f x)))
      (when (> v value)
        (setq biggest x)
        (setq value v)))))
```

(b) (3) The point here is that each player selects the move that maximizes his/her own benefit, and the player above in the tree has to select among these options.



(c) (11) By simply following the process applied to the tree in the previous part, it becomes straightforward to figure out the necessary changes: backed-up-value returns three values; the side to move *picks* the triple that is best for him, and passes up the whole triple. Also `opponent` is replaced by `next`. In some ways the new code is clearer than the original.

```
(defun choose (side state limit)
  (dotimes (d limit)
    (the-biggest #'(lambda (s) (nth side (backed-up-value (next side) s 1 (1+ d))))
                 (successors state))))
(defun backed-up-value (side state depth limit)
  (if (= depth limit)
      (evaluate side state)
      (the-biggest #'(lambda (vvv) (nth side vvv))
                   (mapcar #'(lambda (s) (backed-up-value (next side) s (1+ depth) limit))
                           (successors state)))))
(defun next (side) (mod (1+ side) 3))
```

(d) (5) Most people did pretty well on this. The things to mention include

- Once in an alliance, a player will not maximize his own score necessarily, but will also try to avoid hurting his ally; this is rational because cooperation allows the allies to defeat an otherwise invincible third player, for example; but
- defection is often advantageous once an ally has helped you, so players must be cautious not to lay themselves open to attack by their allies; also covert defection is a valuable strategy.
- Deciding when to make and break alliances, or defect, should be included in the set of possible moves; in deciding to break an alliance or defect one should consider also the costs of losing trustworthiness and goodwill; similarly these factors should be considered in choosing allies, as should the likely outcome of a battle between allies after elimination of a third player.
- all the above considerations need to be part of the player's model of his opponents' and allies' action choices in the search tree, as well as part of his/her own deliberations.
- All this requires very sophisticated reasoning, but one Diplomacy program exists that has proven to be a master of chicanery.