CS182 Solutions to Midterm
Fall Term 1994

Question 1

General Comments

First, you must remember to negate your goal before proving by refutation. Also, if you have all nested existentials (but no universals) the existentials skolemize into *constants*. There are no dependencies between them. In our particular question, that there is some fudge in the world does not depend on the fact that there is also ice-cream in the world. In contrast, suppose we had the sentence “Everybody is eating some ice cream.” Here, we have a direct mapping between a person and the person’s ice cream:

$(\forall x (\exists y (\mathtt{person}(x) \Rightarrow (\mathtt{eats}(x,y) \land \mathtt{IceCream}(y)))))$ becomes
$\neg \mathtt{person}(x) \lor (\mathtt{eats}(x,f(x)) \land \mathtt{IceCream}(f(x)))$ becomes
$\neg \mathtt{person}(x) \lor \mathtt{eats}(x,f(x))$ and
$\neg \mathtt{person}(x) \lor \mathtt{IceCream}(f(x))$

Solution

1 $\neg \mathtt{IceCream}(x_1) \lor \mathtt{Food}(x_1)$
2 $\neg \mathtt{Fudge}(x_2) \lor \mathtt{Food}(x_2)$
3 $\neg \mathtt{Food}(x_3) \lor \neg \mathtt{Food}(x_4) \lor \neg \mathtt{Cold}(x_3) \lor \neg \mathtt{Merge}(x_3,x_4) \lor \mathtt{Cold}(x_4)$
4 $\mathtt{IceCream}(\$c_1)$
5 $\mathtt{Fudge}(\$c_2)$
6 $\mathtt{Cold}(\$c_1)$
7 $\mathtt{Merge}(\$c_1,\$c_2)$
$(\exists z (\mathtt{Fudge}(z) \land \mathtt{Cold}(z)))$.

Negate:
8 $\neg \mathtt{Fudge}(z) \lor \neg \mathtt{Cold}(z)$

Proof:
9 $\langle 8 \text{ and } 5\text{: } z/\$c_2 \rangle$ $\neg \mathtt{Cold}(\$c_2)$
10 $\langle 9 \text{ and } 3\text{: } x_4/\$c_2 \rangle$ $\neg \mathtt{Food}(x_3) \lor \neg \mathtt{Food}(\$c_2) \lor \neg \mathtt{Cold}(x_3) \lor \neg \mathtt{Merge}(x_3,\$c_2)$
11 $\langle 10 \text{ and } 6\text{: } x_3/\$c_1 \rangle$ $\neg \mathtt{Food}(\$c_1) \lor \neg \mathtt{Food}(\$c_2) \lor \neg \mathtt{Merge}(\$c_1,\$c_2)$
12 $\langle 5 \text{ and } 2\text{: } x_2/\$c_1 \rangle$ $\mathtt{Food}(\$c_1)$
13 $\langle 4 \text{ and } 1\text{: } x_1/\$c_2 \rangle$ $\mathtt{Food}(\$c_2)$
14 $\langle 11 \text{ and } 12 \rangle$ $\neg \mathtt{Food}(\$c_2) \lor \neg \mathtt{Merge}(\$c_1,\$c_2)$
15 $\langle 14 \text{ and } 13 \rangle$ $\neg \mathtt{Merge}(\$c_1,\$c_2)$
16 $\langle 7 \text{ and } 15 \rangle$ $\$F$

Question 2

General Comments

Determinism refers to whether or not the next state of the environment is completely determined by the current state and the actions selected by the agent. Can an agent assume its actions will have their desired/intended effect, or could something happen that would cause the action to fail?

The dynamic vs. static dimension refers to whether the environment can change while the agent is thinking.

Episodic environments are ones in which each action of the agent is self-contained; you can think of this as the agent acts, is done, and then we all applaud (or don't). At a large enough granularity, anything could be made to look episodic (even life), but then the distinction is not useful for agent design. So, we take it to mean small units.

Solution

Part (a)

Give a full answer for Category A, the classifications for Category B, and explain some points:

- **X-ray Agent.** Accessible, deterministic, static (or semidynamic), episodic
- **Pulmonary Specialist.** Not accessible, not deterministic, dynamic, not episodic
- **Accessibility.** All the information the X-ray agent needs is in the image; patient records, tests, etc. are partial.
- **Determinism.** The X-ray agent's classification is the only act. Tests may be affected by a range of things outside the PS's control.
- **Static.** The X-ray doesn't move while the agent thinks; the patient's condition can worsen while the doctor thinks.
- **Episodic.** One X-ray at a time; PS diagnosis requires complex sequence of actions.

Explanations for those categories on which there was some confusion:

- **Deterministic.** The X-ray agent works in a deterministic environment; no other agent is around to affect the effect of its classification actions.
- **Episodic.** The X-ray agent has only one action: after perceiving an X-ray it classifies it [sends off to some bin]; if it gets it right we applaud; if not, we debug. A patient treatment is too large a grain size to consider as a single episode; the PS must engage in a complex sequence of actions to do the diagnosis and we don't evaluate until all of these are done.

- **Swimming pool cleaner.** Accessible, dynamic, not episodic; depending on interpretation of the problem could see as deterministic or not, so both answers were accepted with a reasonable explanation.

- **Shopping agent.** Not accessible, not deterministic, dynamic, not episodic

Explanation of categories for which there was some confusion:

- **Deterministic.** If the explanation made a case for nothing interfering with the effects of an action then we gave credit for a yes answer, even though this is typically not the case for these agents.

- **Episodic.** SPC has to continually act if cleaning pool and so is not episodic. Shopping, like the pulmonary specialist, requires a complex sequence of actions.

Part (b)

Category A. The Pulmonary Specialist is more difficult. Because of the complex and partial information that must be reasoned with, and uncertainty in the environment, more complex representations are needed; goal-based and utility-based reasoning mechanisms are required and the search space is more complex.

Category B. The Shopping Agent is more difficult. The difference in accessibility to the environment and the trade-offs that must be made in selecting products to buy requires more complex representations; goal-based and utility-based reasoning mechanisms are required and the search space is more complex

Part (c)

X-ray and swimming pool agents. Reflex agents: Perceiving environment and matching patterns suffices; it's not only okay, but desired to do the same thing in the same state. So, no internal state or representation needed.

PS and Shopping agents. Utility-based agents: Need to represent complex information about the objects and actions in the world and to reason about trade-offs (utilities).

Question 3

1. $$\begin{aligned}(\forall x,y)\,\mathtt{quatkum}(x) \wedge \mathtt{quatkum}(y) \Rightarrow\ & ((\mathtt{fruit}(x) \wedge \mathtt{fruit}(y)) \vee \\ & \quad (\mathtt{vegetable}(x) \wedge \mathtt{vegetable}(y))) \wedge \\ & \neg(\mathtt{fruit}(x) \wedge \mathtt{vegetable}(x)) \wedge \\ & \neg(\mathtt{fruit}(y) \wedge \mathtt{vegetable}(y))\end{aligned}$$

No one got this completely correct. Most of you tried a similar formula, but with only one quatkum. That is enough to assure that any given quatkum is either a fruit or a vegetable, but not both; however, we emphasized that you needed to make sure *all* quatkums belonged to the *same* group (fruit or vegetable). That is not guaranteed unless we specify that any two arbitrary quatkums are each individually of a single type, and that these two types are in fact the same. Note that if there is only one quatkum on Twin-Mars, we're okay, because then `y = x` and the above WFF still makes the correct claim.

The sentence *cannot* be represented in a semantic network, because such networks do not have the power to represent disjunction.

2. `(∀x) quatkum(x) ⇒ ¬soft-skinned(x)`
(or equivalently) `¬(∃x) quatkum(x) ∧ soft-skinned(x)`

Most of you got this. This sentence also *cannot* be represented in a semantic network because it is a negated claim. Some of you proposed that we might create a quatkum class with hard skin as a property, since that would mean all quatkums have hard skin and therefore not soft skin. However, one could always define a subclass (a more specific kind of quatkum) that overrides the skin-type property and has soft skin.

3. This sentence cannot be represented in standard first-order logic, because there is no quantifier available that corresponds to our understanding of "most." All we can make use of are the ideas "every," "none," and "at least one." It can, however, easily be represented in a semantic network; this is one of the major motivations behind the formalism, in fact. We simply provide the January ripening as a default property which can be overridden by individual instances or subclasses of the quatkum class.

4. `(∃x,y) quatkum(x) ∧ tent(y) ∧ owns(Me,y) ∧ in(x,y)`. Most of you got this, as well. Once again, the sentence cannot be represented in a semantic network because we have no way of making general claims about the existence of a particular object. There is a hack along the following lines: Give the quatkum class a location property, then instantiate a particular quatkum Bill, and fill Bill's location slot with my tent. This is a practical work-around that actually turns up in various places in the literature. You got credit for coming up with this, though only one or two people did.

Question 4

Part (a)

Almost every state representation got full credit. A popular choice was to index the leftward and rightward diagonals, then specify each peg as located at the intersection of two such diagonals. The goal test is then that the length of the peg list is one. The easiest approach to operators is using four: `moveNW`, `moveNE`, `moveSW`, and `moveSE`. Each takes a node and a peg location and returns either nothing (if no peg is at that location, or the peg cannot move) or the board resulting from jumping the argument peg and removing the intermediate peg.

An example of a sufficient answer is the following:

A state is a list of ordered pairs, `(x,y)`, where each pair is the location of a peg on the game board indexed by downward diagonals from the left and upward diagonals from the bottom. The initial state is the list of all board locations except `(2,2)`. The goal test returns true iff a state's list is singleton.

There are four operators: `move[NW/NE/SW/SE](node,(x,y))`. Take as an example, `moveNW`. If `(x,y-2)` is not in the state's list, and `(x,y-1)` is, then the new node corresponds to a state with `(x,y)` and `(x,y-1)` both removed but `(x,y-2)` added to the list. Otherwise the operator returns `nil`. The other three operators are analogous.

Part (b)

Path costs are not useful in this problem. However, this is *not* because we are uninterested in knowing the steps which lead to a solution or how many such steps there are; we are in fact quite interested in this, as the problem explicitly states. Rather, the point is that all valid solutions have exactly the same length, since we need to get rid of 13 pegs and each valid step removes exactly one. Thus path cost provides no way of discriminating among solutions and is hence not at all useful.

Part (c)

An incorrect heuristic which showed up in many exams was to count the number of pegs (or the number of pegs minus one) in a given board state. The problem states explicitly that the heuristic must guide search so that it does not proceed exactly as it would in an uninformed algorithm. But consider any board state with `n > 1` pegs. Every child of that state will have exactly `n - 1` pegs, so every child will get exactly the same heuristic evaluation. Thus, the heuristic has no impact on search, and therefore isn't really heuristic at all.

Good heuristics emphasized making moves which didn't leave pegs isolated, for example, by counting the number of pegs which had an adjacent hole filled by a different peg, or by counting the diagonal distance between the two pegs which are furthest away from each other. The latter is actually a reasonable "end-game" heuristic for when the board is only sparsely populated by pegs. Each of these is admissible, since neither returns a number greater than the number of jumps remaining to the nearest goal state (verify this for yourselves).

Question 5

General Comments

An optimal solution is a solution that has the highest quality; in this case, that means it has the shortest path cost. Optimality does *not* refer to the time it takes to find the goal. That's time efficiency and relates to measuring time complexity.

Solution

(a) `g(n) = 0` and `h(n) = h'(n)`: Not optimal. $O(2)$

(b) `g(n) = path cost` and `h(n) = 0`: Optimal. $O(2^{129})$

(c) `g(n) = path cost` and `h(n) = h'(n)`: Optimal. $O(2^{7})$

Explanation

Part (a) is greedy search. Of all three scenerios, this returns a solution path to us the fastest. We only have to expand the node of `init_state` and the node of `state_0` to end up in Goal A $O(2)$. However, Goal A is a worse solution than Goal B because it takes us 137 units to get to Goal A, but only 129 units to get to Goal B. So, we don't get an optimal answer in this case.

Part (b) is uniform cost search, so we do get the optimal solution. The optimal path has a total cost of 129 (leading to Goal B). However, the search will explore all the nodes on the right until exceeded path length of 129 in all possible ways. The time complexity is $O(2^{129})$. So, while this evaluation function gives an optimal answer, the search takes a long time.

Part (c) is the general A* heuristic function. We get optimal solution because `h(n)` never overestimates. Since we've factored in the path cost, once we get to State 0, we'd pick Goal B (which has a much better path cost) rather than Goal A. `f(state_0) = 127 + 1 = 128`. But before we get to State 0, we will first have to check the nodes on the right branch. We'll only come back to State 0 after the path cost of 128 has been exceeded. Since we have to go k levels deep on the right side to beat 128—i.e., $f(n) = k + 2^k$ and $f(n) > 128$ happens when k is 7. So, the running time is about $O(2^7)$. Here, we see that A* gives us an optimal solution much faster than when we used uniform cost.

Question 6

Part 1

The most direct naive encoding is propositional:

(a) `have-dollar` ⇒ `can-buy-coke`

(b) `have-dollar` ⇒ `can-buy-pretzels`

(c) `have-dollar`

A slightly less naive version using FOPC:

(a) `have($)` ⇒ `cb(coke)`

(b) `have($)` ⇒ `cb(pretzels)`

(c) `have($)`

The proof:

(d): (a), (c) `----- modus ponens cb(coke)`

(e): (b), (c) `----- modus ponens cb(pretzels)`

(d), (e) `----- and introduction cb(coke)` $\wedge$ `cb(pretzels)`

(Note: Full credit was given if you showed all the rules, even if you left out the names; but you needed to show `and-introduction`.)

Part 2

Intuitively, the problem is that after you buy coke or buy pretzels, it's no longer (necessarily) true that you have a dollar.

Further Explanations

For full credit you needed to mention that an action changes the state of the world.

Note that it doesn't make sense to write:

(1') `have-dollar` $\Rightarrow$ `(can-buy-coke` $\wedge$ $\neg$`have-dollar)`

Since that's logically inconsistent. What you have to do is recognize that the act of buying coke (or chips) changes the state of the world: What's true in the state before the action (you have a dollar, you have no coke) is not true in the state after (you may or may not have a dollar, but you have coke).

Part 3

Background

You can encode the above constraint using a rule like:

$\forall$`s have-dollar(s)` $\Rightarrow$ `have-coke(result(buy-coke,s))`

where `result` is a function that maps from actions and states to new states. Note that this axiom makes no assertion one way or the other about whether you still have a dollar in the resulting state, and so you can't conclude that you can buy pretzels in that state. Note also that for this to work, we have to assume that only one action at a time can be performed.

<u>Solution</u>

For this problem, you needed to modify each of the statements to include a situation variable, and give a WFF that showed how the buying action changes state. For example:

```
Have($,s) ⇒ Buy-coke(s)
Have($,s) ⇒ Buy-pretzels(s)
```
$\texttt{Have(\$,S}_1\texttt{)}$

Also have WFF encoding loss of a dollar:

```
¬Have($,Result(Buy-pretzels,s))
```

(Note: We gave full credit for any answer to part three that showed use of the situation variable and the result function; three points were given if the explanation was right, but the logic had mistakes. Note that we didn't ask for proof or complete encoding of domain. That would have taken an hour by itself.)