EECS 543 Final Exam Example

Note: The following contains an example of what you might expect the final exam to look like. The questions are of the type that you should expect. The real final exam will, however, have more questions in each part than in this example exam!

General Instructions

This exam is open book. Write your answers in the space provided, using the backs or attaching extra sheets if needed. Make sure you write your name at the top of every page, and keep your answers on the same page as the question appears.

This course operates under the rules of the College of Engineering Honor Code. Sign the pledge below:

I have neither given nor received aid on this examination, nor am I aware of any violations of the honor code.

signed ______________________________

You have 1.5 hours. There are 4 questions, worth a total of 100 points. This exam is worth 20% of your total grade for the course. Be careful to pace yourself; a pace of less than one point per minute is strongly advised.

1

2

3

4

total
Problem 1: Matching

For each of the items on the left, indicate the letter of an item on the right that best (most exactly or uniquely) matches it. Note that there are more items on the right than on the left, and an item on the right can match more than one item on the left.

_____ breadth-first  a. special variable
_____ solution to goal clobbering  b. optimal
_____ admissible  c. backquote
_____ defparameter  d. augmentation
_____ lambda function  e. goal protection
_____ cut  f. operator subgoaling
          g. prevent backtracking
          h. anonymous
          i. specificity
Problem 2: True/False and Justify

For each of the statements below, say (explicitly) whether you think the statement is true or false, and briefly justify why beneath it.

1. In chess, forward chaining is better than backward chaining.

2. C++ is more truly object-oriented than CLOS.

3. Soar uses “chunking” to resolve impasses.
Problem 3: Code Examples

1. Assume that the following has been typed into the Lisp listener:

\texttt{\textgreater (setf L1 '(a b))}

\texttt{\textgreater (setf L2 '(c d))}

\texttt{\textgreater (setf L3 '(e f))}

Now, consider what happens when the following s-expression is typed into Lisp:

\texttt{\textgreater (setf L4 (list L1 (rest L2) (nreverse L3)))}

What is L4? _________________________________________________________

How many new cons cells were allocated to carry out these last 2 s-expressions?________

Draw L4 in cons-cell form below and show where (if at all) in its structure L1, L2, and/or L3 point. (Put check-marks (√) next to the new cons cells counted above if you want to increase your chances of partial credit.)
2. Consider the following Prolog code for determining whether integer \( N \) is a positive or negative number.

\[
p(N, \text{positive}) :- \text{not } p(N, \text{negative}).
\]

\[
p(-1, \text{negative}).
\]

\[
p(N, \text{negative}) :- N1 \text{ is } N + 1,
\]
\[
\text{p}(N1, \text{negative}).
\]

This seems to work fine when I pass it a negative integer. But when I pass it a positive number, it seems to have trouble.

a. What is the trouble that it has, and why?

b. Can reordering the clauses and/ or inserting cut(s) help? Explain.
3. Let's consider Eater agents who can do (simplified) "knight's jump" actions. A simplified knight's jump is a jump that goes one cell in one direction, and then one cell in a sideways direction to the first.

Assume that Soar includes productions for applying the relevant operators. Given the initialization rule below, and the example of the move rule, fill in the conditions of the propose*knightjump rule.

\[
\text{sp } \{\text{initialize*state*directions} \\
\quad \text{(state } \langle s \rangle \ ^{\text{type state}}) \\
\quad \rightarrow \\
\quad \quad \langle s \rangle \ ^{\text{directions } n \langle s \rangle < e \langle w \rangle} \\
\quad \quad \quad \langle n \rangle \ ^{\text{value north } ^{\text{opposite south } ^{\text{sideways east west}}}} \\
\quad \quad \quad \langle s \rangle \ ^{\text{value south } ^{\text{opposite north } ^{\text{sideways east west}}}} \\
\quad \quad \quad \langle e \rangle \ ^{\text{value east } ^{\text{opposite west } ^{\text{sideways north south}}}} \\
\quad \quad \quad \langle w \rangle \ ^{\text{value west } ^{\text{opposite east } ^{\text{sideways north south}}}}\}
\]

\[
\text{sp } \{\text{propose*move} \\
\quad \text{(state } \langle s \rangle \ ^{\text{io.input-link.my-location.} \langle \text{dir} \rangle .\text{content } \{ \langle \text{co} \rangle \ = \text{wall} \}} \\
\quad \quad \quad \ ^{\text{directions } d} \}
\quad \rightarrow \\
\quad \quad \langle s \rangle \ ^{\text{operator } o \ + \ =} \\
\quad \quad \quad \langle o \rangle \ ^{\text{name move } ^{\text{direction } \langle \text{dir} \rangle ^{\text{content } \langle \text{co} \rangle}}}\}
\]

\[
\text{sp } \{\text{propose*knightjump} \\
\quad \rightarrow \\
\quad \quad \langle s \rangle \ ^{\text{operator } o \ + \ =} \\
\quad \quad \quad \langle o \rangle \ ^{\text{name knightjump } ^{\text{firstdir } \langle \text{dir1} \rangle ^{\text{seconddir } \langle \text{dir2} \rangle ^{\text{content } \langle \text{co} \rangle}}}}\}
\]
**Problem 4: Longer Answer**

1. OPS5 decides on the rule to apply next using the following 4 criteria: (1) refraction (it doesn’t fire the same rule on the same WMEs again); (2) recency (it prefers rules that match more recent WMEs); (3) specificity (it prefers rules that have more LHS conditions); and finally (4) randomly. How easy/hard would it be to specify operator selection rules in Soar that consider each of these criteria?

2. Is “least-constrained-variable-first” a good heuristic for CSPs? Why or why not?
3. From this course (and/or your other experience), do you tend to believe in the Physical Symbol System Hypothesis? Why or why not?