CSCI 561 Fall 2003 – HOMEWORK 2 Due 10/14/2003

Question 1(10 pts):

Please answer the following questions in 1 or 2 short sentences.

1. What does it mean for a search algorithm to be optimal?

2. What does it mean for a search algorithm to be complete?

3. What does it mean for a logical inference method to be sound?

4. What does it mean for a logical inference method to be complete?

5. Consider an intelligent alarm system that has the following sentence in its knowledge base:

((Motion => Intruder) v (Noise => Intruder))

Assume that the propositions Motion and Noise take on their respective truth-values directly from information provided by the agent's sensors. Given this sentence, show precisely the circumstances under which this agent can conclude the presence of an intruder (i.e., conclude whether or not the proposition Intruder is true).

Question 2(10 pts):

Assume that we have an A^* search with the evaluation function f = g + Random( ) * h, where g is the actual cost of the path thus far, h is an admissible heuristic, and Random( ) returns random real values between 0 and 1.
Consider the behavior of an A^* search with this evaluation function as compared to the same search without the multiplication by Random( ). Be sure to discuss the following topics:

1. The qualitative effect of the multiplication by Random( ) on A^*'s performance (comparison to other search algorithms would be appropriate here).
2. The effect it has on the optimality of A^*.

Question 4 (30 pts):

Agent Specification –

You are going to specify three agents that you would find in a restaurant, the Customer, Waiter, and Cook. The agents will have the same architecture as specified in the discussion board topic, "Intelligent Agents" Project. Here is an overview of the architecture:

1) An agent is an independent process that receives percepts from the outside world through messages. It will map the messages into internal state (perhaps with

variables, perhaps with queues).

2) The scheduling thread will select an action for execution when the agent is not busy and, of course, only if there is something for it to do. It might decide to terminate a running action and do something else.

3) An action is some task the agent is capable of doing. An agent can only carry out one task at a time. If you find the agent needs to do two actions concurrently, maybe you need two agents. Since our actions do not control real-world outputs, they must send messages to other agents as their only output.

So, for each agent:

- Design the messages and how they are mapped to internal state.
- Describe the rules the scheduler uses in deciding what action the agent should perform.
- Design the actions of each agent, what they do, what internal state they change, what messages they send.

For this assignment assume the restaurant layout simply has some number of tables, each of which is occupied or not occupied.

Elaborate 2 scenarios: 1) Show what would occur if one customer comes in. (2) Show what would happen if 2 customers come in simultaneously.

In a scenario, you show in sequential form, how the agents would behave. You must show how the agent decides what to do (the scheduler decisions), and then what it does.

Here is a possible beginning to the first scenario:

1) Customer1 Agent:
   - Customer1.hungry=True; //initial state
   - if hungry then queueAction(GotoRestaurant); //scheduler rule
   - Executing GotoRestaurant:
     Customer1.location=Restaurant;
     Message to Waiter- PartyOfOne(name=Customer1)
2) Waiter1 Agent:
   - Receives message: PartyOfOne(name=Customer1) which executes
     waitingCustomers.add(Customer1)
   - if tables.isTableEmpty() then queueAction(GreetAndSeat(Customer1))
     (if no table were empty, then the action might be
     GreetAndWait(Customer1,10minutes);
   - etc.
3) etc.

Don't worry about movement actions. As you can see from the above sample, the GotoRestaurant action just assumes the customer is there. You might set some state in the Customer agent that stores its location.

Don't be overly detailed. Try to do everything using object-oriented pseudo-code. [For students who are implementing the agents, you might as well design Java classes.]

Question 5 (50 pts): Programming

Consider First-order-logic without quantifiers. You will be dealing with proposition variables and logical connectives, implication, equivalence and, of course, parenthesis. You have to write a program that can show whether a given sentence is either valid or unsatisfiable or neither. You have to account for precedence and associativity.

Your operators will be defined with the following symbols:

```
NOT X                ~X
X AND Y              X && Y
X OR Y               X || Y
X IMPLIES Y          X => Y
X EQUIVALENT_TO Y    X <=> Y
```

And the variables will be denoted by strings of letters (see example below).

Input: You will be given a text file called **problem.txt**. The file will be located in the current directory. The first line contains an integer **n** specifying the number of problems in the file. Each line will contain a problem. In each line there is a sentence composed of propositional variables, logical connectives, implication, equivalence and parenthesis. The sample file **problem.txt,** is as follows

Line number:	***File contents:***
1.	3
2.	P && (Q && R) <=> (P && Q) && R
3.	Smoke => Fire
4.	P && ~P

Output: All the output should be written to a file called **solution.txt** in the current directory. For each problem in **problem.txt**, there will be a **corresponding line** in **solution.txt** and it will contain your program output for that sentence: one of the following three: **valid, unsatisfiable** or **neither**.

Line number:	***File contents:***
1.	valid
2.	neither
3.	unsatisfiable

Caution: Please follow the following guidelines
1. Do not expect any command line argument(s).
2. There can be any number of white spaces between fields in the input file.
3. Assume that only space or tab will be used for separating the fields in the input file.
4. There will be no line number or "***Line number***" string in **problem.txt**. Don't look for them.