Greg Scott

ECE 4524/5984 - Spring 2015 Problem Set #2 Due: 8am 3/23/2015(via Scholar)

The purpose of this problem set is to gain experience with logic-based methods. You can complete the exercises by either directly marking up this pdf, or by printing, completing, and scanning as a pdf. In the engineering design section you will be simulating the wumpus world. You will then design and implement a knowledge-based agent to navigate the environment.

**Submission:** Upload a zip file containing your exercises.pdf, program files, and program documentation through Scholar at the assignment link using the "Submitted Attachments" button. Be sure to record the the confirmation code, which is known as the "Submission ID", on the next screen. You will also receive an email confirmation with the Submission ID, but don't always rely on email. Keep the Submission ID with your records as this is the only way to verify to me that an assignment has been submitted successfully.

---

# Exercises

1. Given two propositions $A$ and $B$, show that $A \Leftrightarrow B$ is logically equivalent to $A \vee B \Rightarrow (A \wedge B)$ using

   (a) truth tables

| A | B | A∨B | A∧B | (A∨B)→(A∧B) | A↔B |
|---|---|---|---|---|---|
| F | F | F | F | T | T |
| F | T | T | F | F | F |
| T | F | T | F | F | F |
| T | T | T | T | T | T |

logically equivalent by truth table

   (b) substitutions using the basic logical identities

$(A \vee B) \rightarrow (A \wedge B)$

$\sim((A \vee B) \wedge \sim(A \wedge B))$

$\sim(A \vee B) \vee (A \wedge B)$

$(\sim A \wedge \sim B) \vee (A \wedge B)$

Therefore $A \leftrightarrow B$ since they are either both false or both true

2. Given the following PL KB

$$(A \Rightarrow B) \wedge (B \wedge D \Rightarrow C) \wedge (C \Rightarrow E) \wedge (A \wedge D)$$

Prove $E$ or show it cannot be proven using

(a) modus ponens

$$\frac{A,\ A \rightarrow B}{B}$$

$$\frac{D,\ B,\ B \wedge D \rightarrow C}{C}$$

$$\frac{C,\ C \rightarrow E}{E}$$

(b) resolution

$(\sim A \vee B) \wedge (\sim(B \wedge D) \vee C) \wedge (\sim C \vee E) \wedge A \wedge D$

$(\sim A \vee B) \wedge (\sim B \vee \sim D \vee C) \wedge (\sim C \vee E) \wedge A \wedge D$

add negation

$(\sim A \vee B) \wedge (\sim B \vee \sim D \vee C) \wedge (\sim C \vee E) \wedge A \wedge D \wedge \sim E$

$\equiv \emptyset$

so E is true

3. Translate the following sentences into FOL. Use single lowercase characters to denote variables (e.g. x,y), lowercase words to denote functions (e.g. func()), and TitleCase words to denote predicates or constants (e.g. Foo() or Bar).

   (a) some roses have thorns

   $\exists r,t: Rose(r) \wedge Thorn(t) \wedge has(r,t)$

   (b) every night has its dawn

   $\forall n, \exists d: Night(n) \rightarrow Dawn(d) \wedge has(n,d)$

   (c) every cowboy sings his same sad old song

   $\forall c, \exists s: Cowboy(c) \rightarrow Song(s) \wedge isSad(s) \wedge isOld(s) \wedge has(c,s)$

   (d) no mammal except bats can fly

   $\forall m: Mammal(m) \wedge canFly(m) \rightarrow Bat(m)$

   (e) some problems have no optimal solution

   $\exists p: Problem(p) \wedge \sim hasOptimalSolution(p)$

4. Given the following arbitrary FOL sentences, $S_1$ and $S_2$, determine the MGU if one exists. Variables are lowercase, constants uppercase, functions are denoted by ().

   (a) $S_1 = g(f(A, B), A)$ and $S_2 = g(x, y)$

   $\Theta = \{x/f(A,B), y/A\}$

   (b) $S_1 = f(g(A, y))$ and $S_2 = f(g(B, A))$

   $\Theta = \{\}$ fails because A cannot equal B

   (c) $S_1 = g(A, f(x), f(y))$ and $S_2 = g(x, f(B), y)$

   $\Theta = \{\}$ fails because when x is assigned to A it cannot then be assigned to B

   (d) $S_1 = g(A, f(B, C))$ and $S_2 = g(x)$

   $\Theta = \{\}$ fails because functions g do not have the same number of arguments

   (e) $S_1 = f(g(x), h(g(A), B, y))$ and $S_2 = f(x, h(x, y, C))$

   $\Theta = \{\}$ fails because assigning g(x) to x causes an infinite recursion of g(g(g(g.....

5. Given the following FOL KB

$$(P1(x) \wedge P2(x) \rightarrow P3(x)) \wedge (P3(x) \wedge P4(x) \rightarrow P5(x))$$

(a) Suppose the agent is told $P1(A) \wedge P2(A)$, use forward chaining to update the KB

since $P1(A) \wedge P2(A) \rightarrow P3(A)$ KB can infer $P3(A)$

KB: $(P1(x) \wedge P2(x) \rightarrow P3(x)) \wedge (P3(x) \wedge P4(x) \rightarrow P5(x)) \wedge P1(A) \wedge P2(A) \wedge P3(A)$

(b) Suppose now the agent is told $P5(A)$, using backward chaining what is the result of the query $P4(x)$?

The result is a null set since we cannot imply that $P4(A)$ or of any other constant is true given the added information.

6. You have likely encountered the Tower of Hanoi problem before. There are three rods and a number of disks of different sizes which can slide onto any rod. The problem begins with a stack of disks in ascending order of size on one rod. The goal is to move the entire stack to another rod according to the following rules:

   - only one disk can be moved at a time.
   - only the top-most disk in a stack can be moved
   - disks must always be stacked in ascending order according to size

   x = start
   y = other z = other

   (a) Define a KB that can represent this problem for arbitrary number of disks

$$Tower(x) \wedge Tower(y) \wedge Tower(z) \wedge isAscending(x) \wedge isAscending(y) \wedge isAscending(z)$$
$$\wedge ((hasDisks(y) \wedge \sim hasDisks(x) \wedge \sim hasDisks(z)) \vee (hasDisks(z) \wedge \sim hasDisks(x)$$
$$\wedge \sim hasDisks(y)) \leftrightarrow gameOver()) \wedge (StartTower(s) \wedge Disk(d) \wedge isTopOf(d,s)$$
$$\wedge EndTower(e) \wedge (\sim hasDisks(e) \vee (Disk(k) \wedge isTopOf(k,e) \wedge smallerThan(k,d))))$$
$$\leftrightarrow LegalMove(d,e)$$

   (b) To use your KB above to solve a specific problem instance what additional KB entries are needed?

For solving a specific problem, KB entries giving specific states of the towers are needed in order to know the condition of the game and legal moves.

# Engineering Design Problem

The purpose of this problem set is to gain experience with first-order logic and knowledge-based agents. The problem is specified more loosely than in previous assignments and so this assignment has a larger program design element and some room for creativity. Note, there is also a documentation requirement. You can download starter code in the `ps2.zip` file from the website.

The basis of this assignment is the wumpus world as defined in section 7.2 of the AIAMA text. You will design and implement both a simulator for this world and an agent to navigate it. You have considerable freedom with respect to design and implementation details, subject to the following functional specifications. The most important of these is that the simulator and agent can **only** communicate through calls to update the agent's percept and to get the agent's action. There can be no other communication between the modules, in particular no "peeking" from the agent into the simulation module.

## Wumpus World Simulator

In a python file called `wwsim.py` implement a system to populate valid random wumpus worlds and simulate an agent's interaction. Your code should be able to represent a given wumpus world, determine a correct percept sequence given the current state, and simulate the result of an agents action.

Provide a graphical user interface (GUI) using Tkinter to show the current state of the simulated environment, including the relevant status of pits, wumpus, gold, and agent in iconic form. Provide buttons to start a new simulation and make one percept-action time-step. Your GUI should show the current percept sequence, the last action emitted by the agent, and the current performance score. Focus on basic functionality and good user interface design rather than appearance (i.e. ugly glyps are fine as long as they clearly indicate status).

Your file should import the wwagent module described below. The basic loop of your program should be to obtain the current percept sequence and pass it to the agent as a tuple or list in a function call. Then call the agent action function to get the next action. See the agent definition below. Update the performance score and simulate the agent given the returned action, providing an updated percept. This should continue until the agent dies, climbs from the pit (with our without the gold), or a new game is started.

Your program should also be able to run in a non-gui unattended mode, writing a human readable text file describing the simulation to standard output. This mode should be selectable as a command-line flag.

## Wumpus World Agent

In a python file called `wwagent.py` implement an agent to navigate the simulated wumpus world. The agent should use a knowledge-base to represent the rules of the environment and the status of its current model. Inference should be used in conjunction with state-space search to choose a rational action. Your agent can use memory external to the knowledge base.

Your agent module should provide two functions, `update` taking a single tuple, named tuple, or list of percepts and `action` taking no arguments and returning a single action. No other communication between the modules is allowed.

You can (but do not have to) use code from the book, either directly, or for ideas. I have provided the relevant files in the starter code.

## Documentation

You are required to fully document your program at the module, class, and method/function level. You should also include a narrative explaining your design choices, how your code works, and typical results you get when using it. In particular how rational is your agent. Describe any weaknesses or improvements you might make given more time.

How you do your documentation is up to you, however I suggest that you look into the Sphinx document generator as it was designed for literate programming with python. See `http://sphinx-doc.org/` for details.

The final documentation file format should be html or pdf.