CS156 In-Class Exercise #2: Logic-Based Representation

GROUP MEMBER NAMES:

1. Tanmay Singal
2. Alisha Rath
3. Rashmi Vishwanath Bhat
4. Ian Chavez

ENTER “COMPLETE” OR “INCOMPLETE” **COMPLETE**

# Exercise in Propositional Logic With The Wumpus World

## This in-class exercise involves implementing a propositional logic-based representation of data and information about the Wumpus World. The exercise consists of three parts. Part 1 involves becoming familiar with the code in the logic.py file. Part 2 involves implementing and running the code to reproduce Rule 1 through Rule 17 in the lecture slides and textbook. Part 3 involves writing one or more functions that call various functions in logic.py to carry out the logical operations to determine at least one location that you can move to. That is a location that does not contain the Wumpus (W) and not a pit (P).

Before you leave class, you MUST upload your .py file with your code (finished or not) to the Canvas In-Class Exercise 2 Assignment in order to get credit for the in-class exercise.

**NOTE: YOU WILL NEED PYTHON 3.6 DO NOT TRY TO RUN IN PYTHON 3.8 OR LATER.**

Download and unzip the file named “CS156\_In\_Class\_Exercise\_2.zip” on the Module 3 Week 7 section on Canvas.

**PART 1: In the file named** logic.py

Line 44: Class KB defines a Knowledge Base (KB) to which you can tell and ask propositional sentences. The KB that you will/should use to save and retrieve propositions from is called PropKB on line 76.

Line 76: A KB for Propositional Logic. To store a sentence in a PropKB, you will use the Tell member function. To retrieve info from a PropKB you will use the ASK member function. To remove a sentence from the PropKB you will use the retract member function. For example:

kb = PropKB() # create a KB

kb.tell(A & B) # Put sentences A and B in the kb instance of a PropKB print "3: kb.clauses = ", kb.clauses # dumps all clauses in kb

kb.tell(B >> C) # Put B -> C in kb

print "5: kb.clauses = ", kb.clauses # # dumps all clauses in kb just like above

kb.ask(C) # Ask if sentence C is true.

# The result {} means true, with no substitutions kb.ask(P) # Ask if sentence P is true

kb.retract(B) # Remove sentence B from kb

# In the utils.py file

Line 435: class Expr: Is used for logical expressions, and for terms within logical expressions. For example,

Nullary (no args) type expressions

Expr(42) => 42 # A number, representing the number itself. Expr('F') => F # A symbol, representing a variable or constant

Unary (1 arg) type expressions

Expr('~', Expr('P')) => ~P # A symbol represent not P

Binary (2 arg) type expressions

Expr(‘>>’, A, B) => A -> B # An expression representing A implies B

Line 616: def expr(S): Creates a logical express by using Expr above. It parses the input String S. Symbols and numbers are automatically converted to Exprs. For example,

expr("~P12") # Create the expression ~P12 which means there is

# no pit at location 1,2

expr(“K>>L”) # Create the implication K -> L , K implies L

Lines 131 to 154: Various Boolean functions to answer various questions as indicated by the names of the functions. For example,

is\_symbol(s) # A string s is a symbol if it starts with an alphabetic char. is\_var\_symbol # A logic variable symbol is an initial-lowercase string.

…. and so forth

# Back to the logic.py file

Line 188: tt\_entails(kb, alpha): Use truth tables to determine if KB entails sentence alpha. For example,

tt\_entails(expr('P & Q'), expr('Q'))

Line 200: tt\_check\_all(kb, alpha, symbols, model): Helper function to tt\_entails above

Line 245: tt\_true(alpha): Is the sentence alpha a tautology? (alpha will be coerced to an expr.)

tt\_true(expr("(P >> Q) <=> (~P | Q)"))

**You will need to finish your pl\_true code to finish Part 3 of this In-Class Exercise.**

Line 254: pl\_true(exp, model={}): Return True if the propositional logic expression is true in the model, and False if it is false. If the model does not specify the value for every proposition, this may return None to indicate 'not obvious'

Line 345: to\_cnf(s): Convert a propositional logical sentence to conjunctive normal form. That is, of the form ((A | ~B | ...) & (B | C | ...) & ...) For example,

to\_cnf("~(B|C)") # returns (~B & ~C)

Line 359: eliminate\_implications(s): Change >>, <<, and <=> into &, |, and ~. That is, return an Expr that is equivalent to s, but has only &, |, and ~ as logical operators. For example

eliminate\_implications(A >> (~B << C)) # returns ((~B | ~C) | ~A)

Line 491: pl\_resolution(KB, alpha): Propositional Logic Resolution: It determines if alpha follows from KB.

Line 511: pl\_resolve(ci, cj): Return all clauses that can be obtained by resolving clauses ci and cj. For example,

pl\_resolve(to\_cnf(A|B|C), to\_cnf(~B|~C|F)) # returns [(A | C | ~C | F), (A | B | ~B | F)]

Line 574: dpll\_satisfiable(s): Check satisfiability of a propositional sentence.

Line 823: unify(x, y, s): Unify expressions x,y with substitution s; return a substitution That would make x,y equal, or None if x,y can not unify. x and y can be variables (e.g. Expr('x')), constants, lists, or Exprs. [Fig. 9.1] For example,

unify(x + y, y + C, {}) # returns {y: C, x:y}

# PART 2:

Write a function named wumpus\_1() that reproduces the steps to go from the first rule (R1 on page 220 section 7.4.3) to (R17 on page 225 section 7.5.2) in the textbook and as discussed in lecture. Print out the clauses of the KB at the end of your function.

If you would like to see how to call various functions to perform various logical operations such as

1. Creating an instance of a KB
2. Adding (asserting) a sentence into the KB
3. Finding out if a sentence is true in the KB
4. … etc.

Then execute the module/file named logic\_driver.py and examine the code and output.

**Answer**:

1. Reading completed for Part 1.
2. Running of logic\_driver.py with current information.
3. Function change required for pl\_true logic for ⇒ and ⇐

Changes done are below:

A screenshot of a computer program

Description automatically generated

1. There is a wumpus function created in logic.py which is used to add to kb
2. The output after running the wumpus world logic from R1 to R17 with the final clause as follows:

A screenshot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

R1 ~P11 means no pit in 11

R2 B11 ⬄ P12 | P21 means if there is breeze in B11 then either pit in 12 or 21

R3 B21 ⬄ P11 | P22 | P31 means there is breeze in B21 then either pit in 11 or 22 or 31

R4 ~B11 means there is no breeze in 11

R5 B21 means there is a breeze in B21

R6 B11 -> P12 | P21 ^ P12 | P21 -> B11 , obtained after double implication removal from R2

R7 P12 | P21 -> B11 obtained after and elimination from R6

R8 ~B11-> ~(P12 | P21) contrapositive in R7

R9 ~(P12 | P21), modus poneus on R8 using R4

R10 ~P12 & ~P21 de morgan in R9, means no pit in 12 and no pit in 21

R11 ~B12 means no breeze in 12

R12 B12 ⬄ P11 | P12 | P13 means there is breeze in B12 then either pit in 11 or 12 or 13

R13 ~P22 means detected on pit in 22. From R3, we do double implication elimination, and elimination, contrapositive , modus poneus and de morgan

R14 ~P13 From R3, we do double implication elimination, and elimination, contrapositive , modus poneus and de morgan R15 P11 | P22 | P31 , means either there is pit in 11, 22 or 31. From R3

R16 P11 | P31 , using R15 and R13, resolution using ~P22

R17 P31, as we have R1 as ~P11, resolution applied to R16. There is a pit in 31

**PART 3:**

**NOTE: The code you should use is in the aima3.zip file. See Week 8 Code folder on Canvas.**

**CAUTION: There will be mode functions and code that you need to be concerned about.**

This part will probably take the longest amount of time. If you complete this part within the allotted class time, that is GREAT! If not, then that is NOT A PROBLEM. It is not expected that you will be able to complete this part during class. Completing this part will be via Programming Assignment #2. Check Canvas for the due date. Do not wait until the last minute to complete this assignment.

In this part,

1. Modify the WumpusEnvironment class on line 774 in the file agents.py to change the size/dimension of the world. The WumpusEnvironment class inherits from the XYEnvironment class (fined on line 379). You will likely need to first review the code in the agents.py file to understand what the XYEnvironment class does and what the WumpusEnvironment class will need to do in order to create a Wumpus environment. I suggest limiting the dimension of the Wumpus matrix to a 5x5 sized world (i.e., change the value of 10 for width and height on line 388 to 5). You can place Pits, Wumpuses (I suggest only 1 Wumpus to start), Gold,…, in any location you prefer … except don’t place the gold in the same location as a Pit. It is OK to place gold in the same location as a Wumpus, but it will require your agent to detect and kill the Wumpus before it enters that location to pick up the gold. READ the comments in every function very carefully. They provide helpful information.
2. You will need to modify the “percept” that is returned when you “sense” the environment. It should contain the results of sensing the environment, as discussed in class and in the textbook, when the agent is facing a particular direction.
3. Next you will need to identify which functions from the XYEnvironment class will need to be overridden in the WumpusEnvironment class. For example. The percept member function of XYEnvironment class will need to be overridden to be specific for the Wumpus World. A Wumpus World Agent will not make percepts within a given radius of its current location, which is the default for the XYEnvironment class. In addition, each precept that a Wumpus World Agent makes will need to return the current precept values for “Location, Direction, Stench, Breeze, Glitter, Bump, and Scream” Similarly, the execute\_action member function of the XYEnvironment class (line 413). Will need to be overridden or modified to perform one of the allowed actions in addition to Forward, TurnRight, TurnLeft, such as Grab, Shoot, …, and so forth.

There are several additional modifications of the XYEnvironment class member functions, as well as new member functions for the WumpusEnvironment class that will need to be designed and implemented. Which ones are up to you. Before you start to code, I suggest making sure you specify (i.e., create a set of notes) what exactly needs to be done with respect to the environment, and what needs to be done with the Wumpus World Agent. Then make the necessary code changes.

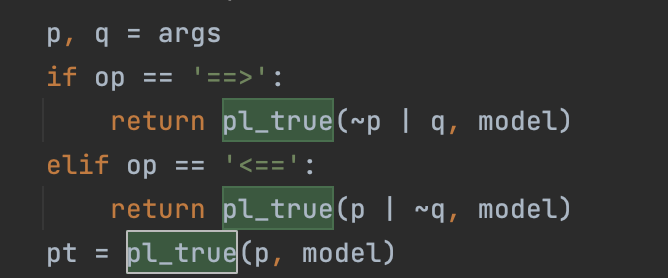
1. Then take a look at the PLWumpusAgent(agents.Agent): class in the logic.py file (NOTE: The PL in PLWumpusAgent means Propositional Logic).
   1. First you will need to figure out the role each variable and function plays in the PLWumpusAgent.
2. To run your agent, use the test\_PLWumpusAgent (Line 124) in the test\_agents.py file. You will need to modify the “run(self, num\_moves=3)” member function in the WumpusEnvironment class in the agents.py file to perform the necessary moves and percepts of the agent.
3. If requested, the instructor is willing to host at least one and possibly two Zoom help sessions to address any questions.
4. FOR THIS IN-CLASS EXERCISE TODAY:
   1. You should be able to complete most if not all of PART 2: If not, just upload whatever code you have written to reproduce Rules R1 thru R17. Your grade for this assignment will require you to make sufficient and reasonable progress. You can work in teams JUST for today’s in-class exercise, but all team members must upload the same file with the names of all members at the top of the file.
   2. When finished with PART 3, upload your code to the Programming Assignment 2 assignment by the specified due date. Although you can collaborate with class partners RE: what needs to be done and understanding what the code in the provided files accomplish, the actual writing of the code to complete PART 3 CAN be done individually or with the original members of your In-Class Exercise #2 group members.

**DONE**

**Things done during class period:**

1. Reading completed for Part 1.
2. Running of logic\_driver.py with current information.
3. Function change required for pl\_true logic for ⇒ and ⇐

Changes done are below:



1. There is a wumpus function created in logic.py which is used to add to kb
2. The output after running the wumpus world logic from R1 to R17 with the final clause as follows:

