Programming Assignment 5: Logical Agent based on First Order Logic

Problem Description

A knowledge base (KB) consists of real world rules and is populated with the rules written in their own original form expressed in First Order Logic (FOL). Develop a logical agent to infer FOL queries that uses following two different approaches of inference -

a. Forward Chaining
b. Resolution technique.

You are given three logical problems in this document. Formulate the predicates, write the rules and formulate queries using appropriate predicates from the vocabulary of the problem domain using human intelligence and knowledge of the real world. Test the implementation of above techniques using these sets of rules and facts as specified in the logic problem description.

Support

You are given a support of basic functionalities written in Python in the three files uploaded earlier **(link: http://aima.eecs.berkeley.edu/python/).** The file logic.py provides basic support for propositional logic which is required to be extended for FOL in your implementation. You are allowed to use the functions of logic.py directly in your program as well. While most of the functionalities support propositional logic, you are expected to make changes in these to work for FOL.

The important *data types* used are as follows

KB	Abstract class holds a knowledge base of logical expressions
KB_Agent	Abstract class subclasses agents.Agent
Expr	A logical expression
substitution	Implemented as a dictionary of var:value pairs, {x:1, y:x}

The *functions* for doing logical inference available in logic.py are:

pl_true	Evaluate a propositional logical sentence in a model
tt_entails	Say if a statement is entailed by a KB
pl_resolution	Do resolution on propositional sentences
dpll_satisfiable	See if a propositional sentence is satisfiable
to_cnf	Convert to conjunctive normal form
unify	Do unification of two FOL sentences and returns a substitution
diff, simp	Symbolic differentiation and simplification

Other *functions* that support propositional logic are eliminate_implications(s), move_not_inwards(s), distribute_and_over_or(s), conjuncts(s) and disjuncts(s), pl_resolve( ),PropHornKB( ), etc.

Application implemented in logic.py using propositional logic is PLWumpusAgent(agents.Agent)

Functions to support FOL are
fol_fc_ask(KB, alpha), standardize_apart(sentence, dic) and fol_bc_ask(KB, goals, theta). You may have to see that they work for you if they are not in proper shape. The modifications in the provided code are also part of your creative contribution. On the other hand, you are free to develop all functions on your own without having to modify the functions in logic.py.

Implementation

Use Python version 3.7 (Windows 10) for implementing your solution. Only standard Python Libraries along with Matplotlib, PyQT and Tkinter should be used. Support from external sources or libraries such as github will not be accepted in your submissions. Each student must design own solution and write own code. [Refer handout to understand the malpractice policies.]

The basic assumptions of the three algorithms must be considered. You are allowed to make changes in the functions given in logic.py, agent.py or util.py to suit your needs. However, any code taken from any other source (internet, file, book etc.) will be treated as plagiarized. You can create objects of FOL such as **predicates** and **functions** appropriately. Specify two quantifiers as **For_every** and **There_exists** to define universal and existential quantifiers.

objects

1. **predicate** - object consisting of name and arguments of a predicate. Facts are the instances of predicates and rules. Also represent the variable names used in the predicate. Remember that these names of variables are one form of data for your algorithms which will take different values on substitution.

2. **rule**- is an object consisting of a body and head (body==>head) where body consists of sentences written in FOL.

3. **clause**- as used in the file logic.py

Note: You can directly use *clause* in place of *rule* if the formulation does not clash. I would advise to keep *rules* and *clauses* as two separate entities. In the text book terminology, a clause refers to the disjunction of literals (refer section 7.5) while a rule may not be a single clause. As you know that any rule can be converted to the CNF form, a rule should be understood as a collection of many clauses combined by conjunction. Hence, rule and clause differ theoretically.

4. **unifier**- It represents the substitution and is represented as a list of pair of values as discussed in the class.

Modules to implement FOL

A. Knowledge Base related

1. readPredicate(): returns a predicate(X, Y, Z,...) with a list of variables. The name of the predicate is specified by the user. The function must take from each line of the file named predicateFile#.txt, the names of predicate and variables. Consider the following file predicateFile0.txt with the description of the predicates

Friend(X,Y): X is friend of Y
Sells(X, Y, Z): X sells Z to Y
Person(X): X is a person

The function returns the name of the predicate as "Friend" and the list of variable symbols as "["X","Y"] for the first time it is invoked. When repeatedly invoked this function it returns the other predicate "Sells" and variables ["X","Y","Z"], and later predicate "Person" and variable ["X"]. Remember that names of variables are important for substitution and unification processes.

2. populate_FOL_KB(KB, filename), takes as input an empty KB and adds the rules from file *filename* to KB. It returns the new KB. You should initialize the KB to an empty structure and populate one by one reading the rules from the file ruleFile#.txt. Initial KB receives rules in their original form. For example, a rule describing "brothers are siblings" expressed in FOL is

$\forall X \forall Y (Brother(X,Y) \Rightarrow Sibling(X,Y))$

This rule is expressed in file ruleFile#.txt as

For_every X **For_every** Y (Brother(X,Y)⇒Sibling(X,Y))

Once the rule is populated in the KB, one can process the rule for different purposes such testing whether the rule is a horn clause or not. It is expected that the user writes all rules in the file rulesFile#.txt after manually standardizing the variables. The programmer (you) is expected to process the quantifiers For_every and There_exists in the function that converts the rules to CNF.

B. Inference Techniques for FOL

3. forwardChaining_FOL(KB, facts, query): returns the truth value of the query. Refer FOL_FC_ASK( ) algorithm given in Fig. 9.3 of the text book and implement the algorithm. You can use the support given as fol_fc_ask(KB, alpha) in logic.py.

4. resolution_FOL(KB, newKB, facts, query): This function converts the rules, facts and query to CNF after checking the basic property of clauses needed for resolution. The variable newKB is the

conjunction of all CNF clauses converted from the rules in KB. The facts represented initially as predicates and rules for specific instances are also added to the newKB in CNF form . Refer section 9.5 to implement the technique.

Other functions needed as helper function must be used (after modification, if needed) from the files given as support or created new.

The purpose of this assignment is to give you sufficient exposure of the domain independent techniques for FOL inference as above. Also, testing of the techniques in real world logic problems to infer queries would give an insight into the implementation aspects of the techniques. Students will also get sufficient exposure of mapping (using human intelligence) the real world problems (Logic problems 1-3) into the FOL symbolic logic and posing the queries in FOL.

First Order Logic Inference

Write the predicates clearly for the following three logic problems in files predicateFile#.txt with their corresponding English description.

Logic Problem 1

Marcus was a human. Marcus was a Pompeian. Marcus was born in 40 A.D. All humans are mortal. All Pompeians died when the volcano erupted in 79 A.D. No mortal lives longer than 150 years. It is now 2019. Alive means not dead. If someone dies, then he is dead at all later times. All Pompeians were Roman. Ceaser was a ruler. All Romans were either loyal to Ceaser or hated him. Everyone is loyal to someone. People only try to assassinate rulers they are not loyal to. Marcus assassinated Ceaser. Is Marcus alive now? Who assassinated Ceaser? Was Marcus loyal to Ceaser? Who was the ruler? When did the volcano erupt? Was Marcus dead in 60 A.D.? Did Marcus hate Ceaser? Was Marcus alive in 35 A.D.?

Logic Problem 2

Pumpkin-Patch Kids: Last Saturday, each of five children, accompanied by one of his or her parents, went to Goldman's Farm to pick out a pumpkin. Once there, however, each child (including Raven) became far more fascinated with a different aspect of the farm (in one case, feeding the animals) than he or she was in the pumpkin patch! Fortunately, each parent was able to recapture his or her child's interest just long enough for them to pick out the perfect pumpkin for their Halloween jack-o'-lantern! From the information provided, match each child with his or her parent (one is Mr. Maier) and determine the aspect of the farm each found more intriguing than pumpkin picking.

(a) Neither Lauren nor Zach was the child accompanied by Ms. Reed. Tara was more interested in shopping at the country store than in picking pumpkins.

(b) Xander and his father, Mr. Morgan, didn't go on the hay ride. Ms. Fedor's child (who isn't Zach or Lauren) was fascinated by the cider making process

(c) Zach is neither the child who went on the hay ride nor the one who wanted to go apple picking. Mr. Hanson's child didn't go on the hay ride.

Logic Problem 3

Consider the following models of the wumpus world. The percepts are breeze, stench and glitter to represent the presence of pit, wumpus and gold. A wumpus creates stench in all its 4 horizontal and vertical neighboring squares. A pit sends breeze in all 4 of its horizontal and vertical squares. The glitter percept is only true in the square containing gold. The termination condition of the game is that the logical agent gets either caught by the Wumpus, or falls in the pit or finds the gold. Produce message appropriately about one of the three conditions reached. The agent moves using its knowledge either available initially, through the facts (percepts' truth values when logical agent reaches the particular square) and by intermediate additions of inferences in the knowledge base. The environment is partially observable and the logical agent is able to sense only the square which it is in, but has a complete access to its evolving knowledge base through which it uses its logic to move right, move left, move forward or backward. For example, if it senses a stench in a square, it must step backward as there could be a wumpus in any of the three squares on its left, right or front, as was discussed during propositional logic discussions in the class. You can refer to the wumpus world description in FOL given in section 8.3 for more clarity. As a programmer, you are expected to create the wumpus world rules in FOL, and display the path taken by the logical agent using the inference algorithms. You can either hardcode the percepts for these two models alone with specified number of wumpuses and pits, in their specified locations and capture the stench and breeze as two dimensional arrays, or you can randomize the environment creation. Remember that the logical agent is able to perceive the percepts only in the square it is in.

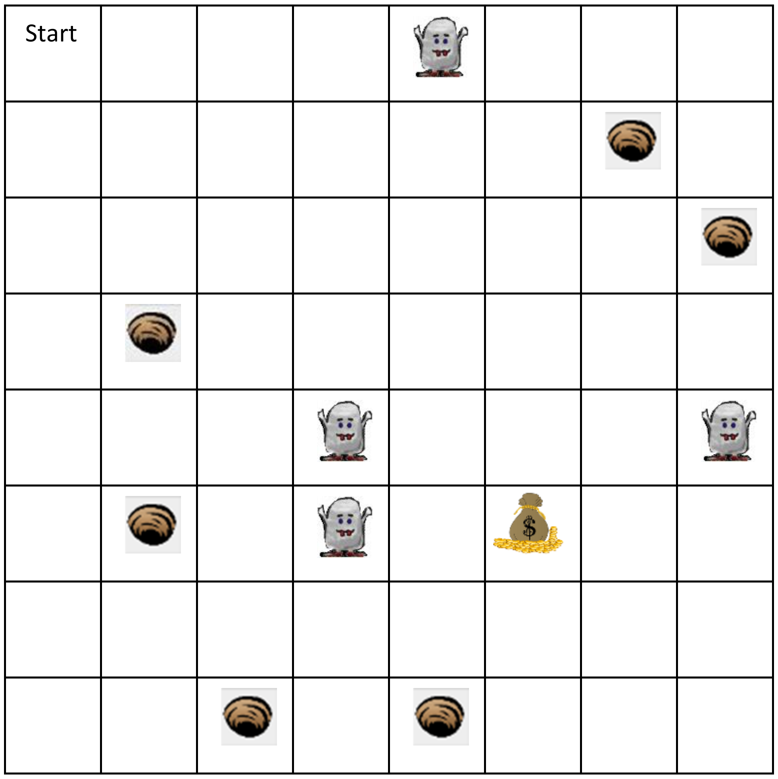


Fig.1. Wumpus World (Model 1)

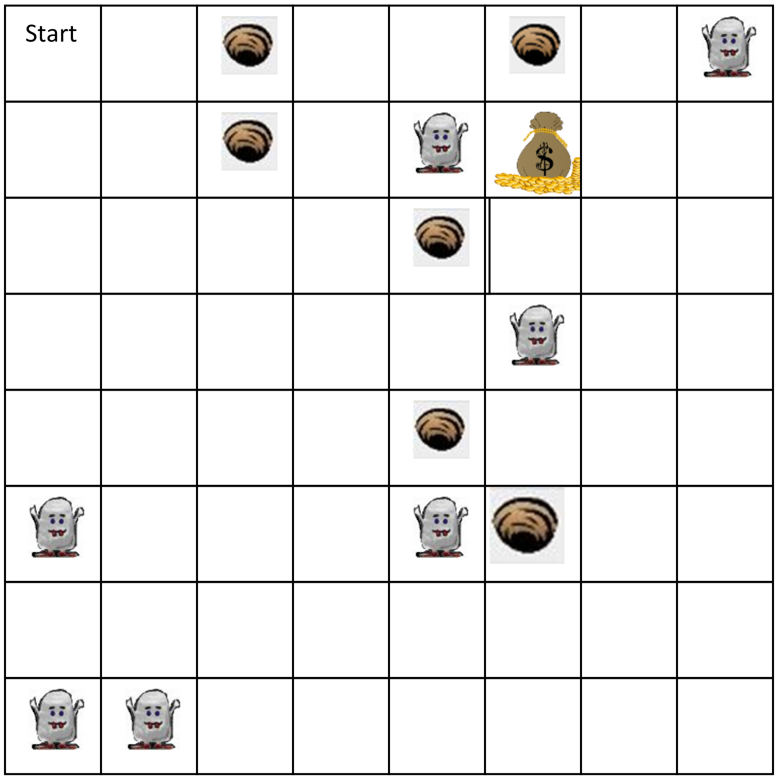


Fig.2. Wumpus World (Model 2)

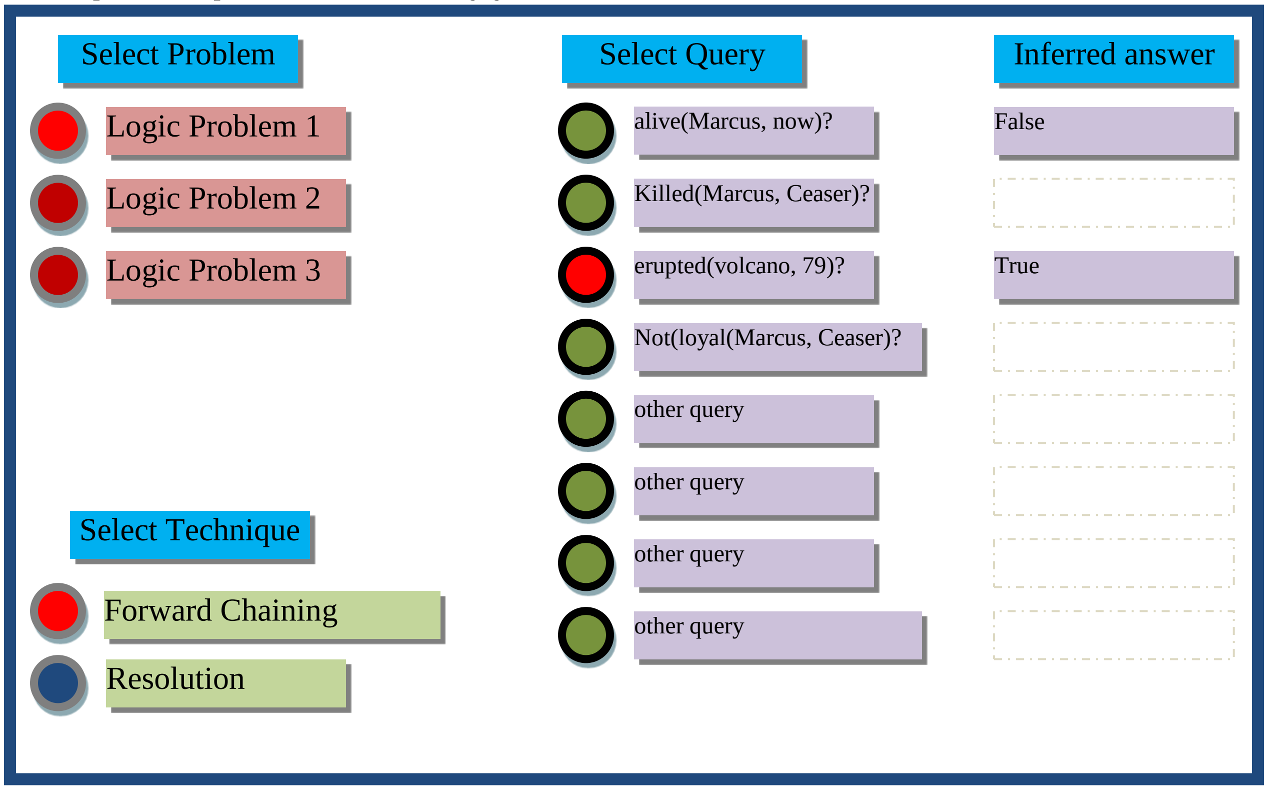
File handling

Generate English descriptions of predicates and rules in FOL in the files *manually* by typing the details in files *predicateFile#.txt* and *ruleFile#.txt* respectively where # is the number of the logic problem. The naming must follow the following scheme. The file named as *KB_inCNF* is the text file containing all rules in CNF form in different lines, and is generated once the knowledge base (KB) is in CNF form. The *output file* contains the name of the technique used, queries, the results of the queries and the time taken by the technique for the corresponding logic problem. Make sure that the results for the same set of queries for a corresponding logic problem are executed by each of the two techniques Forward Chaining (FC) and Resolution to write (append) the results in output#.txt. For example, the file output3.txt will contain name of the technique used, queries, the results of the queries and the time taken by the FC technique for logic problem 3, followed by the similar details obtained by the resolution technique for the same logic problem. The files *KB_inCNF* and *output files* must be generated *automatically* using appropriate functions.

Logic Problem	English description of predicates	Rules in FOL	KB_inCNF	outputFiles
1	predicateFile1.txt	ruleFile1.txt	KB1.txt	Output1.txt
2	predicateFile2.txt	ruleFile2.txt	KB2.txt	Output2.txt
3	predicateFile3.txt	ruleFile3.txt	KB3.txt	Output3.txt

Graphics

Create a graphical user interface (GUI) to use buttons for the selection of logic problem (1-4), techniques and queries formulated by you.



Leave the answers to the queries once invoked/selected on the screen so as to enable the visibility of the answers to all the queries till the end. Ensure that the selections display highlighted buttons (as shown with Red in the above sample GUI). When a different logic problem is selected, clear the query and inferred answer area and refresh with set of queries pertaining to the selected logic problem. If all the queries are not fit in the space, use next button to refresh the previous set of queries and display remaining set of queries pertaining to the same problem. If the logic problem 3 of wumpus world is selected, then after the queries are displayed, prompt the user to see the path. Display the logical path taken by the logical agent in problem 4 after refreshing the query area.

Driver

The driver must integrate all functionalities and execute the functions appropriately using the selections made through the above GUI.

Writeup, evaluation and submission

Write up details will be made available two days before the submission. Evaluation will be out of 16 marks (8% weight). Students are advised to inform me immediately, if any discrepancy exists in this document. The assignment is due for submission on November 17, 2019 (Sunday) by 6:00 p.m. The students are expected to read the text book chapters 8 and 9 and clarify all their doubts pertaining to the problem specification, explanations given above, conceptual understanding, doubts related to individual logic problem or the data structures to be used, and the doubts relating other aspects of implementation.

Please feel free to meet me and discuss your doubts.

Vandana
November 6, 2019