CS 541: Artifical Intelligence

Assignment 4 - Logic(Spring 2023)

Professor Xueqing Liu

Teaching Assistants: Girish Budhrani, Moksha Dave

This assignment has a written part and a programming part.
The full assignment with our supporting code and scripts can be found in logic.zip.

Introduction

- This assignment is due on **12th May 2023** at 11:59 pm EST.
- Collaboration or plagiarism will result in a 0 and being reported to the Graduate Academic Integrity Board. This includes all sorts of online resources and AI tools.
- Please submit the code only after checking it. If the code does not compile, the assignment will be graded as 0.

Submission Details:

You shall submit a zip file named CS541_firstname_lastname.zip which contains:

- A submission.py file which contains your code for problems 1, 2, and 3.
- A PDF file which contains the answer to your problem 4.

Problem Description

You should modify the code in `submission.py` between the given lines only. Any other changes to the file may cause the autograder to fail. Your code will be evaluated on basic test cases, which you can see in grader.py. Basic tests, which are fully provided to you, do not stress your code with large inputs or tricky corner cases. To run the tests, you will need to have all the files in the same directory as your code and grader.py. Then, you can run all the tests by typing commands in terminal:

```
python grader.py
```

This will tell you only whether you passed the basic tests. You can also run a single test (e.g., 3a-0-basic) by typing

```
python grader.py 3a-0-basic
```

In this assignment, you will get some hands-on experience with logic. You'll see how logic can be used to represent the meaning of natural language sentences, and how it can be used to solve puzzles. Most of this assignment will be translating English into logical formulas, but in Problem 4, we will delve into the mechanics of logical inference.
To get started, launch a Python shell and try typing the following commands to add logical expressions into the knowledge base.

Here is a table that describes how logical formulas are represented in code. Use it as a reference guide:

Name	**Mathematical notation**	**Code**
Constant symbol	Stevens	Constant('stevens') (must be lowercase)
Variable symbol	x	Variable('$x') (must be lowercase)
Atomic formula (atom)	deadline, LocatedIn(stevens, x)	Atom('deadline') (predicate must be uppercase) Atom('LocatedIn', 'stevens','$x') (arguments are symbols)
Negation	¬deadline	Not(Atom('deadline'))
Conjunction	deadline ∧ watching TV	And(Atom('deadline'), Atom('tv'))
Disjunction	deadline ∨ watching TV	Or(Atom('deadline'), Atom('tv'))
Implication	deadline → (¬productive)	Implies(Atom('deadline'), Not(Atom('productive')))
Equivalence	deadline ↔ productive(Syntactic sugar for deadline → productive ∧ productive → deadline)	Equiv(Atom('deadline'), Atom('productive'))
Existential quantification	$\exists x$.LocatedIn(stevens,x)	Exists('$x',Atom('LocatedIn','stevens', '$x'))
Universal quantification	$\forall x$.MadeOfAtoms(x)	Forall('$x',Atom('MadeOfAtoms','$x'))

The operations **And** and **Or** only take two arguments. If we want to take a conjunction or disjunction of more than two, use **AndList** and **OrList**. For example: **AndList([Atom('A'), Atom('B'), Atom('C')])** is equivalent to **And(And(Atom('A'), Atom('B')), Atom('C'))**.

Note: In case you still have doubts, you may refer to the example.py file available in the logic.zip

Problem 1: Propositional logic

Write a propositional logic formula for each of the following English sentences in the given function in **submission.py**.

a) If I have a deadline tomorrow and I'm watching TV, then I'm not being very productive.

b) Either I'll go to the gym or go for a run (but not both).

c) The store is open if and only if the sign says "open" and the lights are on.

You can run the following command to test each formula:

```
python grader.py 1a
```

Problem 2: First-order logic

Write a first-order logic formula for each of the following English sentences in the given function in submission.py.

a) The world that contain two types of objects: Student, and Teacher. There is one relations between these objects: a teacher teaches a student. Write first-order logic formulas to describe the following world:

   – Every object is either a student or a teacher;
   – There exists at least one student and at least one teacher;
   – Only student can learn, and only teacher can teach;
   – Every student must have at least one teacher;

   For example, {Teacher(o3), Teacher(o2), Teaches(o3,o1), Student(o1)} is a valid model for ['o1', 'o2', 'o3'], while {Teacher(o3), Teacher(o2), Teaches(o3,o2), Student(o1)} is not, because teachers cannot learn;

b) Write the first-order logic formulas to describe the world that is the same as 2a), except that teacher can learn from teacher, but student still cannot teach, i.e.,

   – Every object is either a student or a teacher;
   – There exists at least one student and at least one teacher;
   – Only teacher can teach, both student and teacher can learn;
   – Every student must have at least one teacher;

   In this case, {Teacher(o3), Teacher(o2), Teaches(o3,o1), Student(o1)} is a valid model for ['o1', 'o2', 'o3'], {Teacher(o3), Teacher(o2), Teaches(o3,o2), Student(o1)} is also a valid model, but {Teacher(o3), Teacher(o2), Teaches(o1,o2), Student(o1)} is not;

You can run the following command to test each formula:

```
python grader.py 2a
```

Problem 3: Liar puzzle

There are four friends - Luke, John, Levi, and Adam. Each one of them is wearing a different colored shirt: red or blue. One person is telling the truth and one person is wearing a red shirt. Based on the following statements, you can determine who is telling the truth and who is wearing the red shirt:

- **Adam says:** "My shirt is not blue."
- **Levi says:** "Adam's shirt is red."
- **John says:** "Levi's shirt is not blue."
- **Luke says:** "John's shirt is blue.
- You know that exactly one person is **telling the truth**
- And exactly one person is **wearing a red shirt**.

  Fill out **liar()** to return a list of 6 formulas, one for each of the above facts. Be sure your formulas are exactly in the order specified.
  You can test your code using the following commands:

```
python grader.py 3a-0
python grader.py 3a-1
python grader.py 3a-2
python grader.py 3a-3
python grader.py 3a-4
python grader.py 3a-5
python grader.py 3a-all # Tests the conjunction of all the formulas
```

  To solve the puzzle and find the answer, tell the formulas to the knowledge base and ask the query TellTruth('$x'), by running:

```
python grader.py 3a-run # Tests the conjunction of all the formulas
```

Problem 4: Logical Inference

Having obtained some intuition on how to construct formulas, we will now perform logical inference to derive new formulas from old ones. Recall that:

- Modus ponens asserts that if we have two formulas, A $\rightarrow$ B and A in our knowledge base, then we can derive B.
- Resolution asserts that if we have two formulas, A $\vee$ B and $\neg$B $\vee$ C in our knowledge base, then we can derive A $\vee$ C.
- If A $\wedge$ B is in the knowledge base, then we can derive both A and B.
  Some inferences that might look like they're outside the scope of Modus ponens are actually within reach. Suppose the knowledge base contains the following formulas:

a) KB = A$\rightarrow$ (B $\vee$ C), B $\rightarrow$ D, C $\rightarrow$ E
Your task: Convert the knowledge base into CNF and apply resolution rule repeatedly to derive E. Please show how your knowledge base changes as you apply derivation rules.

b) KB = (B $\rightarrow$ A), (B $\rightarrow$ C) $\rightarrow$ D
Your task: Convert the knowledge base into CNF and apply resolution rule repeatedly to derive D. Please show how your knowledge base changes as you apply derivation rules.

Hint: You may use the fact that P $\rightarrow$ Q is equivalent $\neg$P $\vee$ Q.

Late policy

90% if late by 1 date, 70 % if late by 2 days, 50 % if late by 5 days and 0 there after.