Artificial Intelligence
COMP 5600/ COMP 6600/ COMP 6600 - D01

Instructor: Dr. Shubhra ("Santu") Karmaker
TA 1: Souvika Sarkar
TA 2: Sanjoy Kundu
Department of Computer Science and Software Engineering
Auburn University
Spring, 2024

February 6, 2024

Assignment #2
Logic and Semantics

Submission Instructions

This assignment is due on **February 13, 2024, at 11:59 pm**. Please submit your solutions via Canvas (https://auburn.instructure.com/). You should submit your assignment as a PDF file. Please do not include blurry scanned/photographed equations, as they are difficult for us to grade.

Late Submission Policy

The late submission policy for assignments will be as follows unless otherwise specified:

1. 75% credit within 0-48 hours after the submission deadline.
2. 50% credit within 48-96 hours after the submission deadline.
3. 0% credit after 96 hours after the submission deadline.

Tasks

In this assignment, you will gain hands-on experience with logic. You'll explore how logic can represent the meaning of natural language sentences and how it can be applied to solve puzzles and prove theorems. Most of this assignment will involve translating English into logical formulas.

The full assignment, along with our supporting code and scripts, has been uploaded in the file section as **logic.zip**. Download the zip file and locate `submission.py`. For this assignment, you should write your code in `submission.py`.

You are expected to modify the code in `submission.py` between the lines:

```
BEGIN_YOUR_CODE
```

and

```
END_YOUR_CODE
```

However, you can add other helper functions outside this block if needed. **Do not make changes to files other than `submission.py`.**

Your code will be evaluated on two types of test cases: basic and hidden, which you can find in `grader.py`. Basic tests, which are provided to you, do not stress your code with large inputs or tricky corner cases. Hidden tests are more complex and do stress your code. The inputs of hidden tests are provided in `grader.py`, but the correct outputs are not. To run the tests, you will need to have `graderUtil.py` in the same directory as your code and `grader.py`. Then, you can run all the tests by typing:

```
python grader.py
```

This will inform you only whether you passed the basic tests. For the hidden tests, the script will alert you if your code takes too long or crashes, but it will not indicate whether you got the correct output. You can also run a single test (e.g., `3a-0-basic`) by typing:

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

We strongly encourage you to read and understand the test cases, create your own test cases, and not just blindly run `grader.py`.

Below is a table describing how logical formulas are represented in code. Use it as a reference guide:

Name	**Mathematical notation**	**Code**
Constant symbol	*stanford*	`Constant('stanford')` (must be lowercase)
Variable symbol	x	`Variable('$x')`(must be lowercase)
Atomic formula (atom)	*Rain* *LocatedIn(stanford,x)*	`Atom('Rain')` (predicate start with uppercase) `Atom('LocatedIn', 'stanford', '$x')` (arguments are symbols)
Negation	¬Rain	`Not(Atom('Rain'))`
Conjunction	Rain ∧ Snow	`And(Atom('Rain'), Atom('Snow'))`
Disjunction	Rain ∨ Snow	`Or(Atom('Rain'), Atom('Snow'))`
Implication	Rain → Wet	`Implies(Atom('Rain'), Atom('Wet'))`
Equivalence	Rain ↔ Wet (syntactic sugar for Rain→Wet∧Wet→Rain)	`Equiv(Atom('Rain'), Atom('Wet'))`
Existential quantification	$\exists x$.LocatedIn(stanford,x)	`Exists('$x', Atom('LocatedIn', 'stanford', '$x'))`
Universal quantification	$\forall x$.MadeOfAtoms(x)	`Forall('$x', Atom('MadeOfAtoms', '$x'))`

Table 1: Mathematical notation and code representation in first-order logic.

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

Problem 1 [20 points]: Propositional Logic

Write a propositional logic formula for each of the following English sentences in the given function in submission.py. For example, if the sentence is **"If it is raining, it is wet,"** then you would write **Implies(Atom('Rain'), Atom('Wet'))**, which would be **Rain→Wet** in symbols (see examples.py). Note: Don't forget to return the constructed formula!

- "If it's summer and we're in California, then it doesn't rain."
- "It's wet if and only if it is raining or the sprinklers are on."
- "Either it's day or night (but not both)."

You can run the following command to test each formula:

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

If your formula is wrong, then the grader will provide a counterexample, which is a model that your formula and the correct formula don't agree on. For example, if you accidentally wrote And(Atom('Rain'), Atom('Wet')) for "If it is raining, it is wet,", then the grader would output the following: Your formula (And(Rain,Wet)) says the following model is FALSE, but it should be TRUE: * Rain = False * Wet = True * (other atoms if any) = False

Problem 2 [20 points]: First-Order Logic

Write a first-order logic formula for each of the following English sentences in the given function in `submission.py`. For example, if the sentence is "There is a light that shines," then you would write `Exists('x', And(Atom('Light', 'x'), Atom('Shines', 'x')))`, which would be $\exists x.\text{Light}(x) \wedge \text{Shines}(x)$ in symbols (see `examples.py`).

Tips:

You can directly write 'x' instead of `Variable('x')` to represent a variable symbol.

Python tuples can span multiple lines, which helps with readability when you are writing logic expressions (some of them in this homework can get quite large).

- "Every person has a mother."
- "At least one person has no children."

Problem 3 [30 points]: Liar Puzzle

Someone crashed the server, and accusations are flying. For this problem, we will encode the evidence in first-order logic formulas to find out who crashed the server. You've narrowed it down to four suspects: John, Susan, Mark, and Nicole. You have the following information:

Mark says: "It wasn't me!"
John says: "It was Nicole!"
Nicole says: "No, it was Susan!"
Susan says: "Nicole's a liar."

You know that exactly one person is telling the truth. You also know exactly one person crashed the server. Fill out ***liar()*** to return a list of 6 formulas, one for each of the above facts. The grader is set up such that you could run individual parts 3a-0 to 3a-5 to debug each formula only if you implement them 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, run:

```
    python grader.py 3a-run
```

Problem 4 [30 points]: Odd and Even Integers

In this problem, we will see how to use logic to prove mathematical theorems automatically. We will focus on encoding the theorem and leave the proving part to the logical inference algorithm. Here is the theorem:

If the following constraints hold:

- Each number x has exactly one successor, which is not equal to x.
- Each number is either odd or even, but not both.
- The successor of an even number is odd.
- The successor of an odd number is even.
- For every number x, the successor of x is larger than x.
- Larger is a transitive property: if x is larger than y and y is larger than z, then x is larger than z.

Then we have the following consequence:

- For each number, there is an even number larger than it.

Note: in this problem, "larger than" is just an arbitrary relation, and you should not assume it has any prior meaning. In other words, don't assume things like "a number can't be larger than itself" unless explicitly stated.

Fill out ***ints()*** to construct 6 formulas for each of the constraints. The consequence has been filled out for you (query in the code). The grader is set up such that you could run individual parts 4a-0 to 4a-5 to debug each formula only if you implement them in the order specified. You can test your code using the following commands:

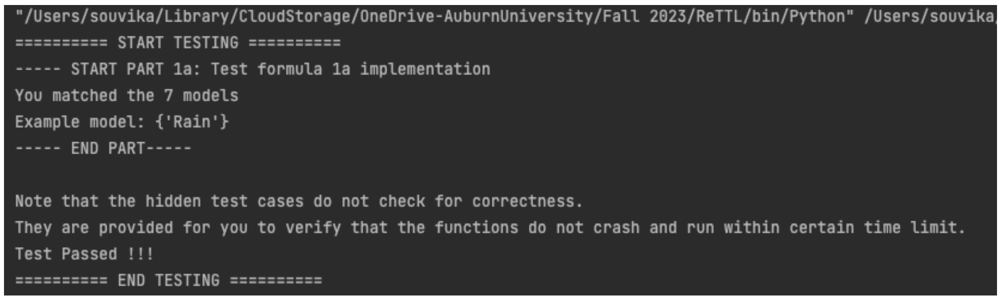
```
    python grader.py 4a-0
    python grader.py 4a-1
    python grader.py 4a-2
    python grader.py 4a-3
    python grader.py 4a-4
    python grader.py 4a-5
    python grader.py 4a-all  | Tests the conjunction of all the formulas
```

To finally prove the theorem, run:

```
    python grader.py 4a-run
```

Deliverables: Please ensure that your submission includes the following:

1. A PDF file containing:
   - A screenshot of the output for each test. For example, here is the output for **Problem 1a**.



```
"/Users/souvika/Library/CloudStorage/OneDrive-AuburnUniversity/Fall 2023/ReTTL/bin/Python" /Users/souvika
========== START TESTING ==========
----- START PART 1a: Test formula 1a implementation
You matched the 7 models
Example model: {'Rain'}
----- END PART-----

Note that the hidden test cases do not check for correctness.
They are provided for you to verify that the functions do not crash and run within certain time limit.
Test Passed !!!
========== END TESTING ==========
```

   - Additionally, include screenshots of the respective implemented code inside,

     `BEGIN_YOUR_CODE`

     and

     `END_YOUR_CODE`

2. Your `sumission.py` file.

If you encounter a scenario where a test case fails, please provide an explanation of the logic you have implemented in the PDF document. This explanation should detail how your logic works, and any specific reasoning behind your approach. This ensures that even if a test case fails, the reviewers can understand your thought process and reasoning behind the implemented logic.