From Language to Logic

Stanford CS221 Fall 2014-2015

Owner TA: Adam Goldberg

Note: grader.py only provides basic tests. Passing grader.py does not by any means guarantee full points.

In this assignment, you will get some hands-on experience with logic and see how logic can be used to represent the meaning of natural language sentences, and how it can be used to solve puzzles and prove theorems. 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.

```
from logic import *
Rain = Atom('Rain')            # Shortcut
Wet = Atom('Wet')              # Shortcut
kb = createResolutionKB()      # Create the knowledge base
kb.ask(Wet)                    # Prints "I don't know."
kb.ask(Not(Wet))               # Prints "I don't know."
kb.tell(Implies(Rain, Wet))    # Prints "I learned something."
kb.ask(Wet)                    # Prints "I don't know."
kb.tell(Rain)                  # Prints "I learned something."
kb.tell(Wet)                   # Prints "I already knew that."
kb.ask(Wet)                    # Prints "Yes."
kb.ask(Not(Wet))               # Prints "No."
kb.tell(Not(Wet))              # Prints "I don't buy that."
```

To print out the contents of the knowledge base, you can call `kb.dump()`. For the example above, you get:

```
==== Knowledge base [3 derivations] ===
* Or(Not(Rain),Wet)
* Rain
- Wet
```

In the output, '*' means the fact was explicitly added by the user, and '-' means that it was inferred. 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	stanford	`Constant('stanford')` (must be lowercase)
Variable symbol	x	`Variable('$x')` (must be lowercase)
Atomic formula (atom)	Rain LocatedIn(stanford, x)	`Atom('Rain')` (predicate must be uppercase) `Atom('LocatedIn', 'stanford', '$x')` (arguments are symbols)
Negation	$\neg$Rain	`Not(Atom('Rain'))`
Conjunction	Rain $\wedge$ Snow	`And(Atom('Rain'), Atom('Snow'))`
Disjunction	Rain $\vee$ Snow	`Or(Atom('Rain'), Atom('Snow'))`
Implication	Rain $\rightarrow$ Wet	`Implies(Atom('Rain'), Atom('Wet'))`
Equivalence	Rain $\leftrightarrow$ Wet (syntactic sugar for Rain $\rightarrow$ Wet $\wedge$ Wet $\rightarrow$ Rain)	`Equiv(Atom('Rain'), Atom('Wet'))`
Existential quantification	$\exists x.$ LocatedIn(stanford, x)	`Exists('$x', Atom('LocatedIn', 'stanford', '$x'))`

Universal quantification	$\forall x. \mathrm{MadeOfAtoms}(x)$	`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'))`.

Problem 1: 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 $\mathrm{Rain} \to \mathrm{Wet}$ in symbols (see `examples.py`). Note: Don't forget to return the constructed formula!

a. `formula1a`: *"If it's summer and we're in California, then it doesn't rain."*

b. `formula1b`: *"It's wet if and only if it is raining or the sprinklers are on."*

c. `formula1c`: *"Either it's day or night (but not both)."*

You can run the following command to test each formula:

```
python grader.py formula1a
```

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
```

In this model, it's not raining and it is wet, which satisfies the correct formula $\mathrm{Rain} \to \mathrm{Wet}$ (TRUE), but does not satisfy the incorrect formula $\mathrm{Rain} \wedge \mathrm{Wet}$ (FALSE). Use these counterexamples to guide you in the rest of the assignment.

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`. 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 $\exists x. \mathrm{Light}(x) \wedge \mathrm{Shines}(x)$ in symbols (see `examples.py`).

a. `formula2a`: *"Every person has a mother."*

b. `formula2b`: *"At least one person has no children."*

c. `formula2c`: Create a formula which defines `Daughter(x,y)` in terms of `Female(x)` and `Child(x,y)`.

d. `formula2d`: Create a formula which defines `Grandmother(x,y)` in terms of `Female(x)` and `Parent(x,y)`.

Problem 3: 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:

- John says: "It wasn't me!"
- Susan says: "It was Nicole!"
- Mark says: "No, it was Susan!"
- Nicole says: "Susan's a liar."
- You know that exactly one person is telling the truth.
- You also know exactly one person crashed the server.

a. 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 liar-0
...
python grader.py liar-5
python grader.py liar-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 `CrashedServer('$x')`, by running:

```
python grader.py liar-run
```

Problem 4: Modus ponens inference

For this problem, we will implement a generalized Modus ponens rule of inference for propositional logic. Recall that Modus ponens asserts that if we have two formulas, $A \rightarrow C$ and A in our knowledge base, then we can derive C. Generalized Modus ponens allows for multiple antecedents. If we have the conjunction of k antecedents $A_1, \dots, A_k$ and we also have one of those symbols A_i ($1 \leq i \leq k$), then we derive the implication formula where A_i is removed. Formally, if the following two formulas are in the knowledge base:

- $A_1 \wedge \cdots \wedge A_k \rightarrow C$
- A_i

Then we derive

$$A_1 \wedge \cdots A_{i-1} \wedge A_{i+1} \wedge A_k \rightarrow C.$$

In the following, we will simply refer to generalized Modus ponens as Modus ponens.

a. In this part, you will see that 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:

$$\text{KB} = \{(A \vee B) \rightarrow C, A\}.$$

Convert the knowledge base into conjunctive normal form (CNF).

Define a new rule that derives $P \rightarrow Q$ to $\neg P \vee Q$, as well as an analogous one from $\neg P \vee Q$ to $P \rightarrow Q$. Then use only Modus ponens and these two new rules to derive C. Remember, this isn't about you as a human being able to reason through the manipulations, but rather about the rote application of a small set of transformations which a computer could execute. Please show how your knowledge base changes as you apply the rules.

b. Recall that Modus ponens is not complete, meaning that we can't use it to derive everything that's true. Suppose the knowledge base contains the following formulas:

$$\text{KB} = \{A \vee B, B \rightarrow C, (A \vee C) \rightarrow D\}.$$

In this example, Modus ponens cannot be used to derive D, even though D is entailed by the knowledge base.

Recall that the resolution rule is complete though. Convert the knowledge base into CNF and apply the resolution rule repeatedly to derive D.

Problem 5: Odd and even integers

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

If the following conditions hold:

- Each number x has a unique 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 greater than it.

a. Fill out `ints()` to construct 6 formulas for each of the conditions. The consequence has been filled out for you (`query` in the code). You can test your code using the following commands:

```
python grader.py ints-0
...
python grader.py ints-5
python grader.py ints-all  # Tests the conjunction of all the formulas
```

To finally prove the theorem, `tell` the formulas to the knowledge base and `ask` the query by running model checking (on a finite model):

```
python grader.py ints-run
```

b. Suppose we added another constraint:

   - A number is not larger than itself.

Briefly argue that there is no finite model for which the resulting set of 7 constraints is consistent. This means that if we try to prove this theorem by model checking only finite models, we will find that it is false, when in fact the theorem is true for a countably infinite model (where the objects in the model are the numbers).

Problem 6: Utterance Parser (Extra Credit)

How can we parse utterances from a human into first order formulas? A couple have been provided in the code for you in `createNLIGrammar()`. Parsing is hard, so we have simplified the set of possible words for you to make your job easier.

a. Please populate rules for the following sentence structures in `createNLIGrammar()` (X and Y are atoms):

   1. Every X has a Y.
   2. No X has a Y.
   3. If a X has a Y, then it has a Z.