Assignment 8. From Language to Logic

Hwanjo Yu
CSED342 - Artificial Intelligence

Contact: TA Byoungwoo Kang (bykang@postech.ac.kr)
Deadline: 9th June 2024 at 2:00 pm

General Instructions

This assignment has a written part and a programming part.

✎ This icon means a written answer is expected in `writeup.pdf`. Refer to `writeup.tex` for pdf file generation.

⌨ This icon means you should write code in `submission.py`.

You should modify the code in `submission.py` between

```
# BEGIN_YOUR_CODE
```

and

```
# END_YOUR_CODE
```

but you can add other helper functions outside this block if you want. Do not make changes to files other than `submission.py`.
Your code will be evaluated based solely on **hidden** test cases, which you can see in `grader.py`. This is because if you have followed the instructions for each problem correctly, the answer will be deterministic.
To run all the tests, type

```
python grader.py
```

This will tell you only whether you passed the hidden tests. On the hidden tests, the script will alert you if your code takes too long or crashes, but does not say whether you got the correct output. You can also run a single test (e.g., `1a-1-hidden`) by typing

```
python grader.py 1a-1-hidden
```

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

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 and prove theorems. Most of this assignment will be translating English into logical formulas, but in Problem 2, 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**
Atomic formula (atom)	Rain LocatedIn(postech, x)	`Atom("Rain")` (predicate must be uppercase) `Atom("LocatedIn", "postech", "$x")` (arguments are symbols)
Negation	$\neg$Rule	`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"))`

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 **Rain** $\rightarrow$ **Wet** in symbols (see `examples.py`). Note: Don't forget to return the constructed formula!

Problem 1a [2 points] ⌨

"If it's summer and we're in California, then it doesn't rain."

Problem 1b [2 points] ⌨

"It's wet if and only if it is raining or the sprinklers are on."

Problem 1c [2 points] ⌨

"Either it's day or night (but not both)."

Problem 1d [2 points] ⌨

"One can access campus server only if she is a computer science major or not a freshman."

Problem 1e [3 points] ⌨

"There are 10 students (i.e. student1, ..., student10) and they all pass AI course."

Problem 2. Logical Inference

Having obtained some intuition on how to construct formulas, we will now perform logical inference to derive new formulas from old ones. Also, we will use conjunctive normal form (CNF). A CNF formula is a conjunction of clauses. Example: $(A \vee B \vee \neg C) \wedge (\neg B \vee D))$. Every formula f in propositional logic can be converted into an equivalent CNF formula $f' : M(f) = M(f')$. Some conversion rules are below:

- Modus Ponens: $\frac{f \rightarrow g, f}{g}$
- Resolution: $\frac{f \vee h, g \vee \neg h}{f \vee g}$
- Eliminate $\wedge$: $\frac{f \wedge g}{f, g}$
- Eliminate $\leftrightarrow$: $\frac{f \leftrightarrow g}{(f \rightarrow g) \wedge (g \rightarrow f)}$
- Eliminate $\rightarrow$: $\frac{f \rightarrow g}{\neg f \vee g}$
- Move $\neg$ inwards: $\frac{\neg(f \wedge g)}{\neg f \vee \neg g}$
- Move $\neg$ inwards: $\frac{\neg(f \vee g)}{\neg f \wedge \neg g}$
- Eliminate double negation: $\frac{\neg\neg f}{f}$
- Distribute $\vee$ over $\wedge$: $\frac{f \vee (g \wedge h)}{(f \vee g) \wedge (f \vee h)}$

Problem 2a [5 points]

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 two formulas:

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

Your task: First, convert the knowledge base into conjunctive normal form (CNF). Then apply Modus ponens to derive **D**. Please show how your knowledge base changes as you apply derivation rules.

Remember, this isn't about you as a human being able to arrive at the conclusion, but rather about the rote application of a small set of transformations (which a computer could execute).

Problem 2b [5 points]

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. However, recall that the resolution rule is complete.

Your task: Convert the knowledge base into CNF and apply the resolution rule repeatedly to derive **D**.