


Assignment 8. From Language to Logic


Hwanjo Yu
CSED342 - Artificial Intelligence

Contact: TA Seongje Lee (lsj720@postech.ac.kr), Jaehyun Lee (jminy8@postech.ac.kr)

General Instructions

This (and every) assignment has a written part and a programming part.

 This icon means a written answer is expected in `writeup.pdf`.

 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 on two types of test cases, **basic** and **hidden**, 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. 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 all the tests, type

```
python grader.py
```

This will tell you only whether you passed the basic 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-0-basic`) by typing

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

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. 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	<code>Atom("Rain")</code> (predicate must be uppercase)
Negation	\neg Rule	<code>Not(Atom("Rain"))</code>
Conjunction	$\text{Rain} \wedge \text{Snow}$	<code>And(Atom("Rain"), Atom("Snow"))</code>
Disjunction	$\text{Rain} \vee \text{Snow}$	<code>Or(Atom("Rain"), Atom("Snow"))</code>
Implication	$\text{Rain} \rightarrow \text{Wet}$	<code>Implies(Atom("Rain"), Atom("Wet"))</code>
Equivalence	$\text{Rain} \leftrightarrow \text{Wet}$ (syntactic sugar for $\text{Rain} \rightarrow \text{Wet} \wedge \text{Wet} \rightarrow \text{Rain}$)	<code>Equiv(Atom("Rain"), Atom("Wet"))</code>

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).”

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`

In this model, it's not raining and it is wet, which satisfies the correct formula **Rain** \rightarrow **Wet** (TRUE), but does not satisfy the incorrect formula **Rain** \wedge **Wet** (FALSE).

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. 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**.
- If **A** \wedge **B** is in the knowledge base, then we can derive both **A** and **B**.

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.

- 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 formulas:

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

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

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

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).