LAB 11 Propositional Logic

If you use python go to https://github.com/aimacode/aima-python/blob/master/intro.ipynb for instructions and links to the files you need. As always, look at https://github.com/aimacode/aima-python/blob/master/tests/test_logic.py for examples on how to use the code. The code (which includes examples) is at https://github.com/aimacode/aima-python/blob/master/logic.py. [1]

1.1 Propositions

A *propositional statement* or *proposition* is a statement that is unambiguously either true or false in a given context. Examples include: "*Quaid-e-Azam was born in July*" (false), *2 + 3 equals 5* (true), and "*Some humans are reptiles*" (false). The truth or falsity of a given proposition is called its truth value. Statements that involve a subjective judgment, such as *"Ali is a good guy"*, are not propositional statements.

Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "Where is Ali", "How are you", "What is your name", are not propositions.

1.1.1 Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

i. **Atomic Propositions**

ii. **Compound propositions**

[1] Notes: The aima.logic package requires that propositional variables (i.e., those whose values must be True or False) begin with an uppercase letter and logic variables (i.e., those that can take on any value) begin with a lowercase letter. See here for a an example of using logic.py on gl. The Python examples below assume you are using Python 3.X and have done "from logic import *". The aima's logic.py package is well commented, so take a look if you have questions. The answer might be in the comments.

Atomic Proposition: Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false. For example

- 2+2 is 4, it is an atomic proposition as it is a **true** fact.
- "The Sun is cold" is also a proposition as it is a **false** fact.

Compound proposition: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives. For example:

- "It is raining today, and street is wet."
- "Ankit is a doctor, and his clinic is in Mumbai."

A well-formed Boolean expression in a programming language such as Python is a concrete embodiment of a proposition. The following example uses an equality test to construct a Boolean expression:

1.1.2 Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has **∧** connective such as, **P ∧ Q** is called a conjunction. For example “Ali is intelligent and hardworking”. It can be written as,
   P= Ali is intelligent,
   Q= Ali is hardworking. → P∧ Q.
3. **Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions. For example: "Qasim is a doctor or Engineer**"**, Here
   P= Qasim is Doctor.
   Q= Qasim is an Engineer, so we can write it as **P ∨ Q**.
4. **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
   If it is raining, then the street is wet.
   Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
5. **Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence,** for example If I am breathing, then I am alive
   P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

1.1.3 Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called **Truth table**. Following are the truth table for all logical connectives:

1.1.3.1 Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

P	Q	R	¬R	P∨Q	P∨Q→¬R
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

1.1.3.2 Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

1.1.4 Logical equivalence:

Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

Let's take two propositions A and B, so for logical equivalence, we can write it as A⇔B. In below truth table we can see that column for ¬AV B and A→B, are identical hence A is Equivalent to B

A	**B**	**¬A**	**¬AV B**	**A→B**
T	T	F	**T**	**T**
T	F	F	**F**	**F**
F	T	T	**T**	**T**
F	F	T	**T**	**T**

1.2 Lab Tasks

Exercise 11.1. Checking Validity

Use the functions in aima's logic.py to see which of the following are valid, i.e., true in every model. You will have to

i. convert these sentences to the appropriate string form that the python code uses (see the comments in the code) and
ii. use the `expr()` function in `logic.py` to turn each into an Expr object, and
iii. use the `tt_true()` function to check for validity. Provide text from a session that shows the result of checking the validity of each. We've done the first one as an example.

a. P v ¬P

```
>>> tt_true(expr('P | ~P'))

True
```

b. P → P

c. P→ (P v Q)

d. (P v Q) → P

e. ((A ∧B) →C) ↔ (A → (B → C))

f. ((A →B) →A) →A

Exercise 11.2. Satisfiability

Use the functions in logic.py to see which of the following are are satisfiable. We've done the first one as an example.

a. P ∧ Q

```
>>> dpll_satisfiable(expr('P & Q'))

{P: True, Q: True}
```

b. ALIVE→ ¬DEAD ∧ ¬ALIVE ∧ ¬DEAD

c. P → ¬ P v P

d. ~ (P v ¬ P)

Exercise 11.3. Propositional Consequence

For each of the following entailment relations, say whether or not it is true. The text on the left of the entailment symbol (⊨) represents one or more sentences (separated by commas) that constitute a knowledge base. We've done the first one for you.

a. P ∧ Q ⊨ P

b. P ⊨ P ∧ Q

c. $P \vDash P \lor Q$

d. $P \vDash \neg\neg P$

e. $P \rightarrow Q \vDash \neg P \rightarrow \neg Q$

f. $\neg P \vDash P \rightarrow Q$

g. $\neg Q \vDash P \rightarrow Q$

h. $P \land (P \rightarrow Q) \vDash Q$

i. $(\neg P) \land (Q \rightarrow P) \vDash \neg Q$