Heaven's Light is Our Guide

Rajshahi Universiy of Engineering and Technology

Course Code

ECE 2214

Course Title

Numerical Methods and Discrete Mathematics Sessional

Experiment Date: October 2, 2023

Submission Date: October 9, 2023

Lab Report 4: Python codes to find logical contradiction, contraposition & equivalence

Submitted to	Submitted by
Md. Nahiduzzaman	Md. Tajim An Noor
Lecturer	Roll: 2010025
Dept of ECE, Ruet	

Contents

Python codes to find logical contradiction, contrapositition & equivalence.

Md Tajim An Noor

1 Introduction

1.1 Contradiction

A statement that is always false is known as a contradiction. A statement or notion that is logically or intrinsically false is referred to as a contradiction. It occurs when a logical argument leads to a situation where two opposing statements cannot both be true at the same time. One common example of a contradiction in discrete mathematics is:

"There exists an integer n such that n is both even and odd."

This statement is a contradiction because by the definition of even and odd integers, no integer can be both even and odd simultaneously. Even integers are divisible by 2, whereas odd integers are not.

1.2 Contraposition

Contraposition refers to the inference of going from a conditional statement into its logically equivalent contrapositive. The contrapositive of a statement is formed by negating both the hypothesis and the conclusion of the original conditional statement and reversing their order. Here is the general form of a contrapositive:

Original statement: If A, then B.

Contrapositive: If not B, then not A.

In other words, the contrapositive switches the roles of the antecedent (A) and the consequent (B) and negates both of them.

1.3 Logical Equivalence

Compound propositions that have the same truth values in all possible cases are called logically equivalent. We can also define this notion as follows:

The compound propositions p and q are called logically equivalent if $p \leftrightarrow q$ is a tautology. The notation $p \equiv q$ denotes that p and q are logically equivalent.[1]

2 Tools Used

- Python
- VS Code - for running python code
- MacTeX -LaTeXcompiler
- VS Code with LaTeX workshop extension as a text editor

3 Process

3.1 Code:

3.1.1 Contradiction

```python
import sympy
from itertools import product
from sympy.logic.boolalg import truth_table
from sympy.abc import x, y

p, q = sympy.symbols("p q")

# Defining logical expressions
expr = ~(p | ~p)

table = truth_table(expr, [p, p])
contradiction = True
for t in table:
    # printing the truth table
    print("{0} -> {1}".format(*t))
    if t[1] == True:
        contradiction = False
        break

if contradiction:
    print("The statement is a contradiction.")
else:
    print("The statement is not a contradiction.")
```

3.1.2 Contraposition

```python
def find_contrapositive(sentence):
    # Split the sentence into words
    words = sentence.split()

    # Find the position of "if" and "then" in the sentence
    if_index = -1
```

```python
    then_index = -1
    for i, word in enumerate(words):
        if word.lower() == "if":
            if_index = i
        elif word.lower() == "then":
            then_index = i

    # Check if both "if" and "then" were found
    if if_index != -1 and then_index != -1:
        # Identify the p (if) and q (then) parts of the statement
        p = " ".join(words[if_index + 1 : then_index])
        q = " ".join(words[then_index + 1 :])

        # Negating both the p and the q
        np = "it is not true that " + p
        nq = "it is not true that " + q

        # Form the contrapositive statement
        contrapositive = f"If {np}, then {nq}"

        return contrapositive
    else:
        return "Invalid input sentence. Please use 'if' and 'then' in your sentence."


# Example usage:
original_sentence = "If he comes then I will go."
contrapositive_sentence = find_contrapositive(original_sentence)
print("Original:", original_sentence)
print("Contrapositive:", contrapositive_sentence)
```

3.1.3 Logical Equivalence

```python
import sympy

# Define symbolic variables
p, q = sympy.symbols("p q")

# Define your logical expressions. Here we enter the logical expressions we want to check equivalance of.
# Here for testing, De Morgan's theorem is used.
expr1 = ~(p & q)
expr2 = ~p | ~q

# Check for logical equivalence: Simplifying the logical expressions and then storing the boolean value in a variable.
```

```
equivalent = sympy.simplify_logic(expr1) ==
    sympy.simplify_logic(expr2)
# From the previous statement, we get if the expressions are
    logically equivalant.
if equivalent:
    print("The expressions are logically equivalent.")
else:
    print("The expressions are not logically equivalent.")
```

3.2 Output

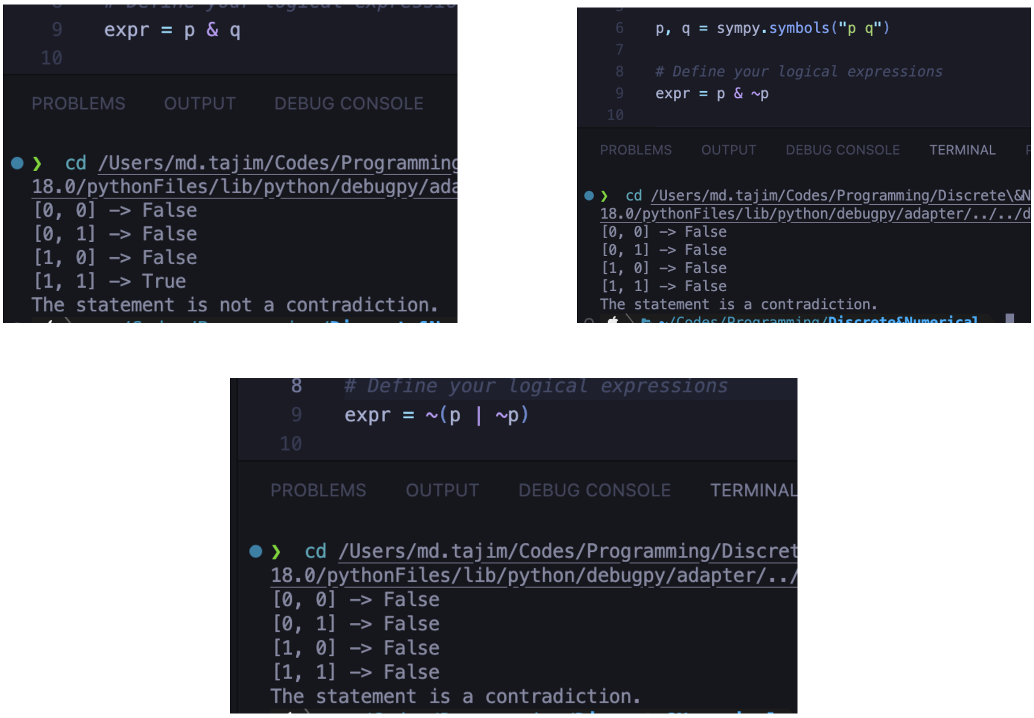

Figure 1: Outputs for Contradiction

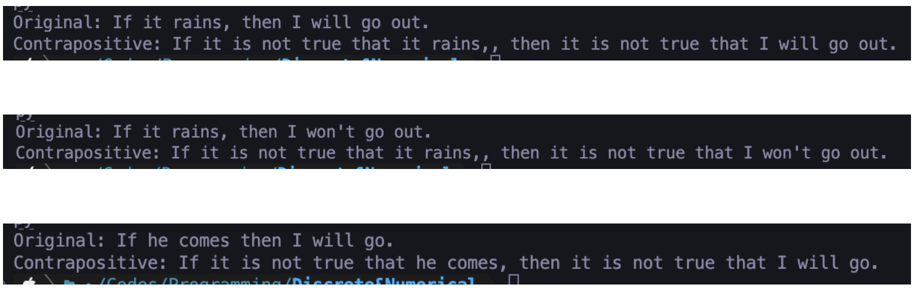

Figure 2: Outputs for Contraposition

Figure 3: Outputs for Logical Equivalence

4 Discussion

To solve the three problems, the python library *Sympy* was used. SymPy is an open source computer algebra system written in pure Python. It is built with a focus on extensibility and ease of use, through both interactive and programmatic applications.[2] Using SymPy, the input logical sequence can easily be processed. otherwise there would be too many cases to take care of. Also it'll be harder to process big logical inputs.

Using SymPy's built-in functions, the inputs, which are logical sentences, are processed to a simplification, and then using other functions, the logical operations were done.

References

[1] K. H. Rosen, *DISCRETE MATHEMATICS AND ITS APPLICATIONS,SEVENTH EDITION.* McGraw-Hill.

[2] R. P. M. GRANGER, F. BONAZZI, H. GUPTA, S. VATS, F. JOHANSSON, and M. J. FABIAN PEDREGOSA, "Sympy: Symbolic computing in python."