D2 Implementation

from nnf import Var

from lib204 import Encoding

# Variables

t1 = Var('t1') #Olivia

t2 = Var('t2') #Ulric

t3= Var('t3') #Alice

t4= Var('t4') #Yates

t5= Var('t5') #Bob

t6= Var('t6') #Zara

t7= Var('t7') #Sam

t8= Var('t8') #Jeff

t9= Var('t9') #Quentin

t10= Var('t10') #Valerie

t11= Var('t11') #Roslina

t12= Var('t12') #Wallace

t13= Var('t13') #Nate

t14= Var('t14') #Harry

t15= Var('t15') #Karen

t16= Var('t16') #Eva

t17= Var('t17') #Gale

t18= Var('t18') #Chloe

t19= Var('t19') #Larry

t20= Var('t20') #Farrel

t21= Var('t21') #Michelle

t22= Var('t22') #Perry

# Build an example full theory for your setting and return it.

# There should be at least 10 variables, and a sufficiently large formula to describe it (>50 operators).

# This restriction is fairly minimal, and if there is any concern, reach out to the teaching staff to clarify

# what the expectations are.

def example\_theory():

E = Encoding()

1. E.add\_constraint(t1 >> (~t2 & ~t3))
2. E.add\_constraint(t4 >> (~t4 & t5))
3. E.add\_constraint(t1 >> (t6))
4. E.add\_constraint(t7 >> (~t8 | ~t9))
5. E.add\_constraint(t10 >> (~t11))
6. E.add\_constraint(t12 >> ((~t7 | t1) & ~(~t7 & t1)))
7. E.add\_constraint(t13 >> (t14 | t15))
8. E.add\_constraint(t16 >> (~t7))
9. E.add\_constraint(t7 >> (t13))
10. E.add\_constraint(t10 >> (t17 >> (~t16 & t4)))
11. E.add\_constraint(t9 >> (~t18 | ~t19))
12. E.add\_constraint(t20 >> (t14 & t21))
13. E.add\_constraint(t5 >> (t10 | t22))
14. E.add\_constraint(t11 >> (t15 >> (~t9 | t14)))
15. E.add\_constraint(t12 >> (t10))
16. E.add\_constraint(t10 >> (~t21 & t4))
17. E.add\_constraint(t19 >> (t22))
18. E.add\_constraint(t18 >> (~t13 & ~t5) & t7)
19. E.add\_constraint(t3 >> (~t21 & (t16 & t22)))

return E



D3: Jape Proofs

* + Knights(T) can say that another Knight is a liar as long as it’s in an ‘**or**’ statement where the other half is the truth, thus making the Knight a truth teller
    - T >> ~T | ~L
  + Knaves(L) can say that another Knave is a liar as long as it’s in an ‘**and**’ statement where the other half is a lie, thus making the Knave a liar
    - L >> ~L & ~T
  + Two Knaves can say each other are truthful
  + Two Knights can say each other are truthful
  + No inhabitant, whether Knight or Knaves, can claim that they are lying (a Knave)
  + You cannot determine whether someone is a Knight or Knave within the context of their statement alone
  + If two inhabitants make contradictory statements, one must be a Knave and one must be a Knight

D5: Documentation

**Project Summary**

This puzzle takes place on an island inhabited by two groups of people: Knights, who always tell the truth, and Knaves, who always tell lies. The object of the game is to determine who the Knaves and Knights are by analyzing the statements they make about the other inhabitants.

A model will determine whether each person is truthful (Knight) or a liar/is untruthful (Knave).

**Propositions**

The original proposal for this project had many more categories of variables, however, it was suggested that we could simplify the problem to only use the following category and many of the original proposal’s variables could be absorbed into the constraints of a given “version” of the problem.

ti: This is true when person i is truthful (Knight).

**Constraints**

Knights and Knaves will make statements about whether other inhabitants are truthful or not. Using our proposition ti we can view person i's statement as an implication. If person i is truthful (ti), that implies that their statement about the other inhabitants must be true. For the general case, we can say person i makes statement p, and thus if person i is telling the truth (a Knight), we have:

ti  → p

Where p is a statement about the other inhabitant

ex. *tj & ~tk* : person j is truthful and person k is lying

~tj | ~tk : at least one of person j and person k is lying

( tj & ~tk ) | ( ~tj & tk ) : either person j or person k is lying

( tj & p ) | ( ~tj & ~p ) : person j would say p

The people can make statements in a number of different forms, which follow the structure of logical arguments (AND, OR, NOT, IF AND ONLY IF, etc...).

**Model Exploration**

There are various ways these types of problems can be solved. One way is making the assumption that they are all telling the truth, then examining the consequences make amendments as the problem is being solved. A solver would likely follow a similar pattern, assigning a True or False value to the proposition variables and examining the consequences before deciding on a proper assignment.

Considering this, we can explore different ways of making the problems more or less complex. Adding more people to the problem creates more variables that need to be given a boolean variable, thus making the problems more complex. The same may hold for adding more statements, leading to more assumptions about the people in the problem. This is one way we explored the difficulty of problems.

Complexity of the different types of statements that the people make can also be explored further. We can examine the number of steps/comparisons a solver makes in order to find the solution to a problem, then compare that with different variations of statements. We may see that certain statements, for example and Exclusive Or with regards to the truthfulness of two people, make the problem more difficult for the solver to assign values to the propositions.

**First-Order Extension**

To extend the model to predicate logic, a basic extension would be to create a predicate for both the set of truthful people and untruthful people. This would take the form of Knight(t) and Knave(t). This, however, is not an incredibly useful extension

A more complex extension would involve perhaps a predicate that defines whether a person is making a statement involving another person, however given that these statements have many different forms, that may become overly complex very quickly. We would love to hear if anyone has a suggestion as to what aspect of this problem we might be able to better model with predicate logic.