Homework 03: Logic Representation

Courses `CSC14003`: Intro to Artificial Intelligence
`18CLC6`, `FIT - HCMUS`.
`20/08/2020`

This is an INDIVIDUAL assignment:

- `18127231`: Đoàn Đình Toàn (GitHub: @t3bol90)

Problem:

Problem 1. (3.0pts) For each English sentence below, write the FOL sentence that best expresses its intended meaning. Use Cat(x) for "x is a cat," Mouse(x) for "x is a mouse," and Chases(x, y) for "x chases y."

1. Every cat chases every mouse.

   Ans:

$$\forall x \; \forall y \; \text{Cat}(\text{x}) \land \text{Mouse}(\text{y}) \implies \text{Chases}(\text{x}, \text{y})$$

2. For every cat, there is a mouse that the cat chases.

   Ans:

$$\forall x \; \exists y \; Cat(x) \implies Mouse(y) \land Chases(x, y)$$

3. There is a cat who chases every mouse.

   Ans:

$$\exists x \; \forall y \; Cat(x) \land (Mouse(y) \implies Chases(x, y))$$

4. Some cat chases some mouse.

   Ans:

$$\exists x \; \exists y \;\; Cat(x) \land \;\; Mouse(y) \land Chases(x, y)$$

5. There is a mouse that every cat chases.

   Ans:

$$\exists x \; \forall y \; Mouse(x) \land (Cat(y) \implies Chases(y, x))$$

6. For every mouse, there is a cat who chases that mouse.

   Ans:

$$\forall x \; \exists y \; Mouse(x) \implies Cat(y) \land Chases(y, x)$$

Problem 2. (2.0pts) Given a knowledge base as follows

$$P \vee Q, Q \rightarrow (R \wedge S), (P \vee R) \rightarrow U \ (*)$$

Check whether each of the following sentences is entailed by KB using PL-Resolution

Solution:
Extract from $(*)$, KB:

I. $P \vee Q$

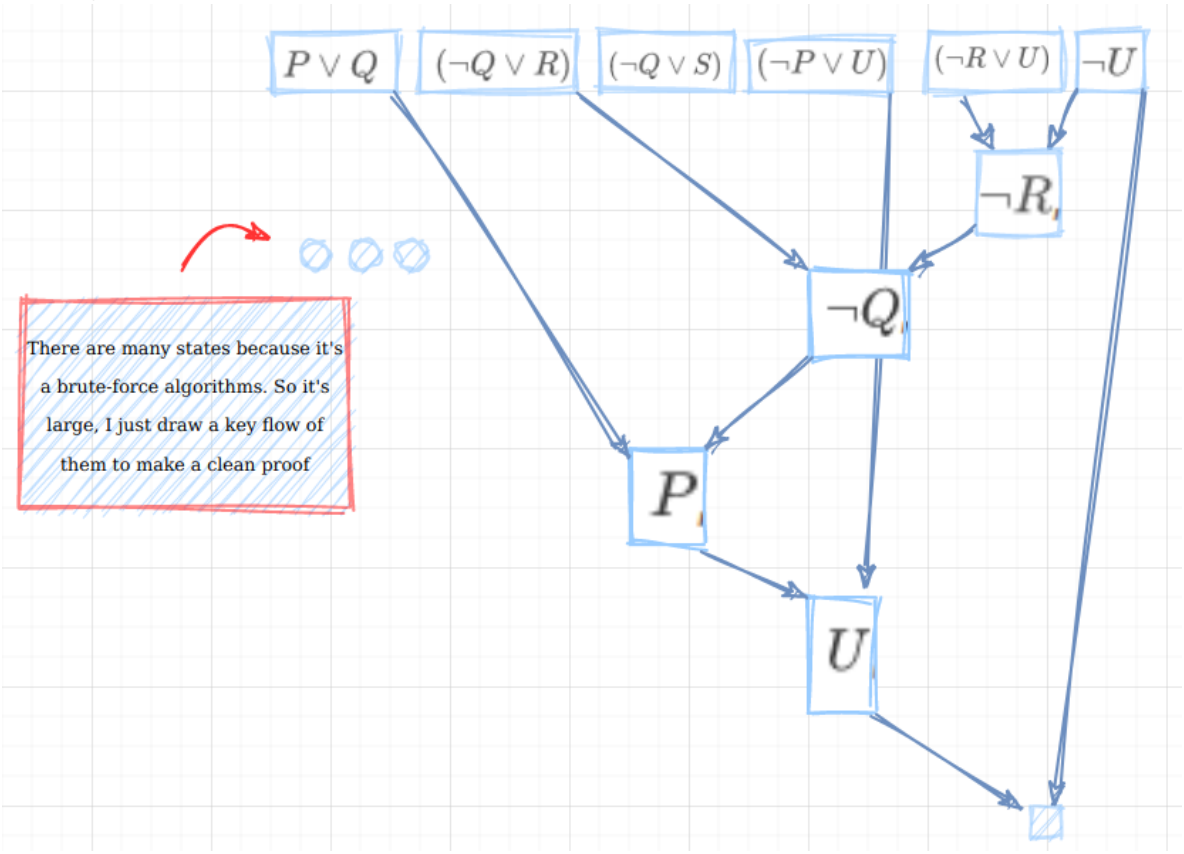
II. $Q \rightarrow (R \wedge S) \iff \neg Q \vee (R \wedge S) \iff (\neg Q \vee R) \wedge (\neg Q \vee S)$

III. $(P \vee R) \rightarrow U \iff (\neg P \wedge \neg R) \vee U \iff (\neg P \vee U) \wedge (\neg R \vee U)$

a) U
So the proof of $KB \models \alpha, \ \alpha = U$
I will proof by contradiction, we show that $KB \wedge \neg U$ is unsatisfiable.:

1. $\neg U$, from our hypothesis.
2. $\neg R \vee U$, from (II).
3. $\neg R$, combined from 1 and 2.
4. $\neg Q \vee R$, from (II).
5. $\neg Q$, combined from 4 and 3.
6. $P \vee Q$, from (I).
7. P, from 6 and 5.
8. $\neg P \vee U$, from (III).
9. U, from 8 and 7 (It's unsatisfiable). (**)

Proof by PL-Resolution:.



Because of (**), and two clauses resolve to yield the empty clause, in which case KB entails U so we can proof $KB \models U$.
b) $\neg U$
So the proof of $KB \models \alpha, \ \alpha = \neg U$

We need to show that $KB \wedge U$ is not satisfiable:
By using PL-Resolution, there are no new clauses that can be added, in which case KB does not entail $\neg U$.

Problem 03. (5.0pts) Given a Puzzle game board as below (Figure 01-a)

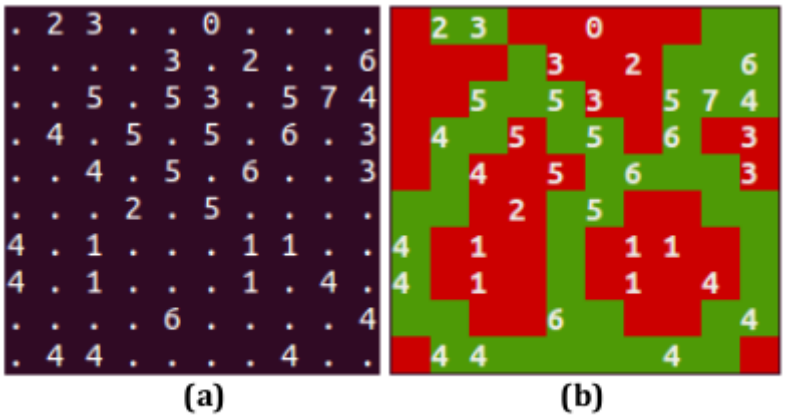


Figure 01. *A sample of Puzzle game board*

Each cell in the board is assigned a number or a blank (dot cells). The player needs to fill all cells in green or red color so that the number of green cells around a cell with a number (including adjacent ones and itself) is exactly the number (Figure 01-b).

a) (3.0 pts) Assign a logical variable to each cell in which the cell is green if the variable is True and red if False. Write CNF clauses to restrict the game rule above for the given cell below. Justify your procedure.

.	.	.
.	2	.
.	.	.

cell

a	b	c
d	e	f
g	h	i

variables

(**b) (2.0 pts)** Implement a program using Python and Glucose3 solver (Python-SAT: https://pypi.org/project/python-sat/0.1.1dev7/) to solve the game board in Figure 01-a.
• Input: **input.txt** (tab separated)

```
-   First line: 2 integers as the matrix shape
-   Rest line: a matrix of numbers and dots
```

• Output: **output.txt** (tab separated)
  - a matrix of letters G and R (green/red cells)
• Source code: attached one file only, filename is formatted like [student_number].py*
  E.g. 12345678.py

```
# Example for Glucose3
from pysat.solvers import Glucose3
g = Glucose3()
g.add_clause([-1, 2])
g.add_clause([-2, 3])
print (g.solve())
#True
print (g.get_model())
#[-1, -2, -3]
```

Solution:

a.

There is $\binom{9}{2}$ ways to assign logical values to the the given cells. Assume that we assign values $True$ (green) to a and b. So the rest need to be $False$.

The clause for this case:

$$\begin{gathered}[(A \wedge B) \implies (\neg C \wedge \neg D \wedge \neg E \wedge \neg F \wedge \neg G \wedge \neg H \wedge \neg I)] \wedge \\ [(\neg C \wedge \neg D \wedge \neg E \wedge \neg F \wedge \neg G \wedge \neg H \wedge \neg I) \implies [(A \wedge B)]\end{gathered}$$

Then, it's equal to:

$$\big[(A \wedge B) \iff (\neg C \wedge \neg D \wedge \neg E \wedge \neg F \wedge \neg G \wedge \neg H \wedge \neg I)\big](*)$$

Because the term "assigning True values to cells" depend on "assigning False values to cells" and otherwise. So we need to keep the constraints on both terms.

So $(*)$ equals to:

$$\begin{gathered}(\neg A \vee \neg B \neg C) \wedge (\neg A \vee \neg B \neg D) \wedge (\neg A \vee \neg B \neg E) \wedge \\ (\neg A \vee \neg B \neg F) \wedge (\neg A \vee \neg B \neg G) \wedge (\neg A \vee \neg B \neg H) \wedge (\neg A \vee \neg B \neg I) \wedge \\ (A \vee C \vee D \vee E \vee F \vee G \vee H \vee I) \wedge (B \vee C \vee D \vee E \vee F \vee G \vee H \vee I)\end{gathered}$$

This is CNF clause from of this case.

In generality:

$\forall x, y \in P,\ (x \neq y)$. With P is a set of cells in this problem.

$Q = \{x, y\}$ and $K = P \setminus Q$.

Call p_i is clause of a case in $\binom{9}{2}$ cases which x_i, y_i is assign $True$ and the rest is $False$ or $\forall k \in K_i, k := False$.

So the CNF form for p_i is:

$$[\bigwedge_{k \in K_i} (\neg x_i \vee \neg y_i \vee \neg k)] \wedge \big[[x_i \vee (\bigvee_{k \in K_i} k)] \wedge [y_i \vee (\bigvee_{k \in K_i} k))]\big]$$

b.

I have done 2 things on the templates file to complete this homework:

1. Complete the `toCNF` function, by read `pysat` documentation many times and research on Google "what is this function return?", "what is SAT model?", "How can solve SAT problem?", "What is 2-SAT, 3-SAT problems? Are they NP problems?",... Then, I learn somethings new but I realize that I just need to complete this function:

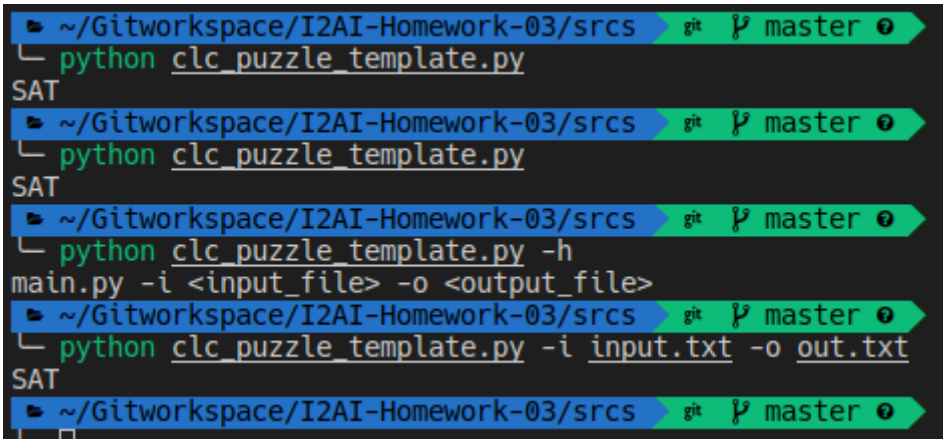
```
def toCNF(mat, lvars):
    '''
        input:  mat: the matrix of puzzle's map/grid.
                lvars: the matrix of labels in mat.
        output: pre :: list[l1,l2..],
                the CNF clauses, combined all of cell's clause in the map.

    '''
    height = len(mat)
    width = len(mat[0])
    pre = []
```

```
    for i in range(height):
        for j in range(width):
            clauses = getClauses(mat,i,j) ## Get all the clauses then add to
one CNF clause!
            if clauses[0]: ## Maybe null :)))
                for clause in clauses:
                    pre.append(clause)
    return pre
```

Thanks for this assignment, it's really helpful for our next project.

2. The input, output parser is not good enough, and I think you want to write a arguments style. So, I modify it:

```
    infile = 'input.txt'  # sys.argv[1]
    outfile = 'output.txt'  # sys.argv[2]
    try:
        opts, args = getopt.getopt(argv, "hi:o:", ["ifile=", "ofile="])
    except getopt.GetoptError:
        print('main.py -i <input_file> -o <output_file>')
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print('main.py -i <input_file> -o <output_file>')
            sys.exit()
        elif opt in ("-i", "--ifile"):
            infile = arg
        elif opt in ("-o", "--ofile"):
            outfile = arg
    mat = readMat(infile)
    lvars, num = initVars(mat)
    clauses = toCNF(mat, lvars)
    sol, res = solveCNF(clauses)
```



Now, it's so cool :))).