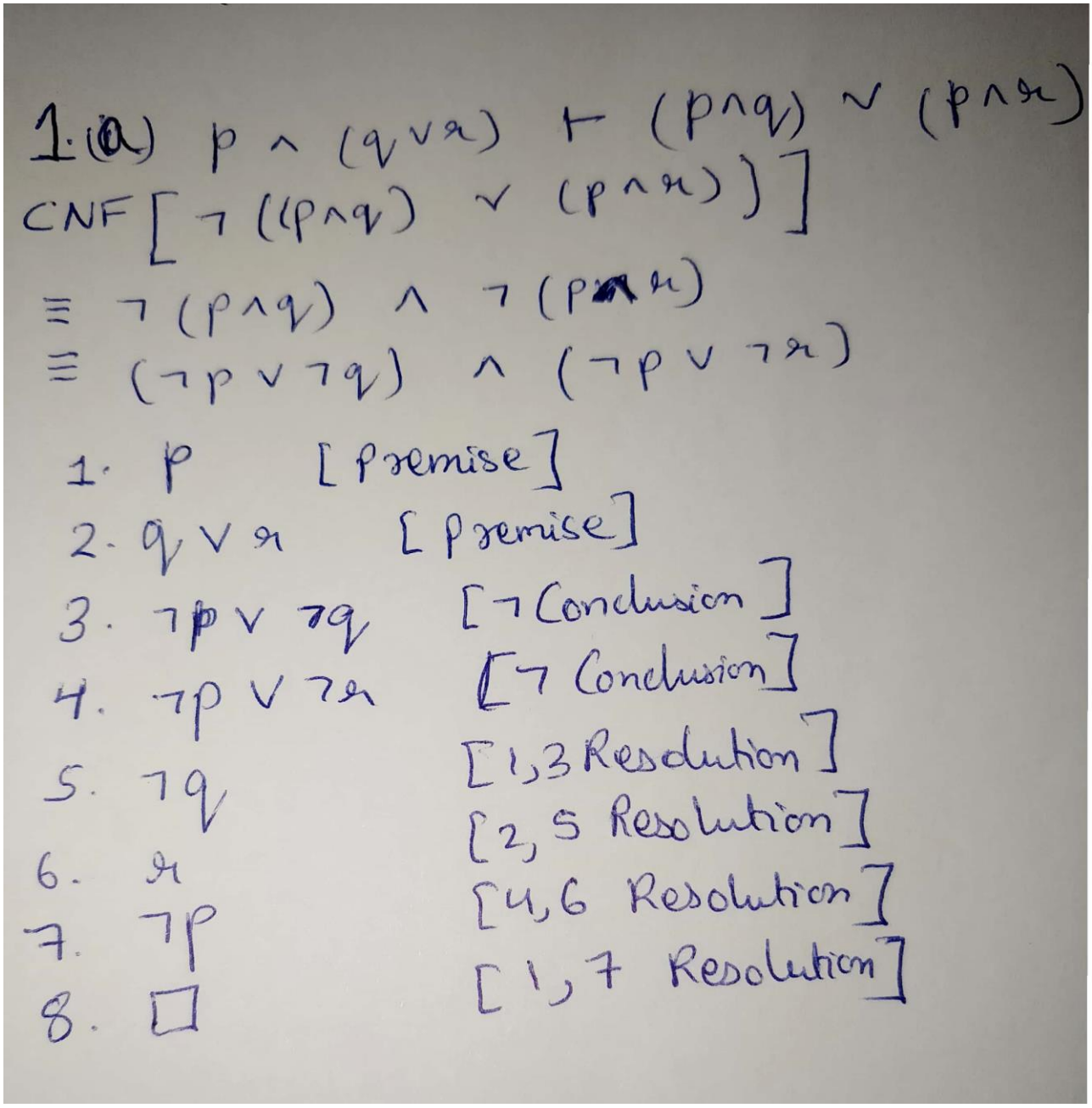
# REPORT

z5151301

**ANSWER – 1**

(a)

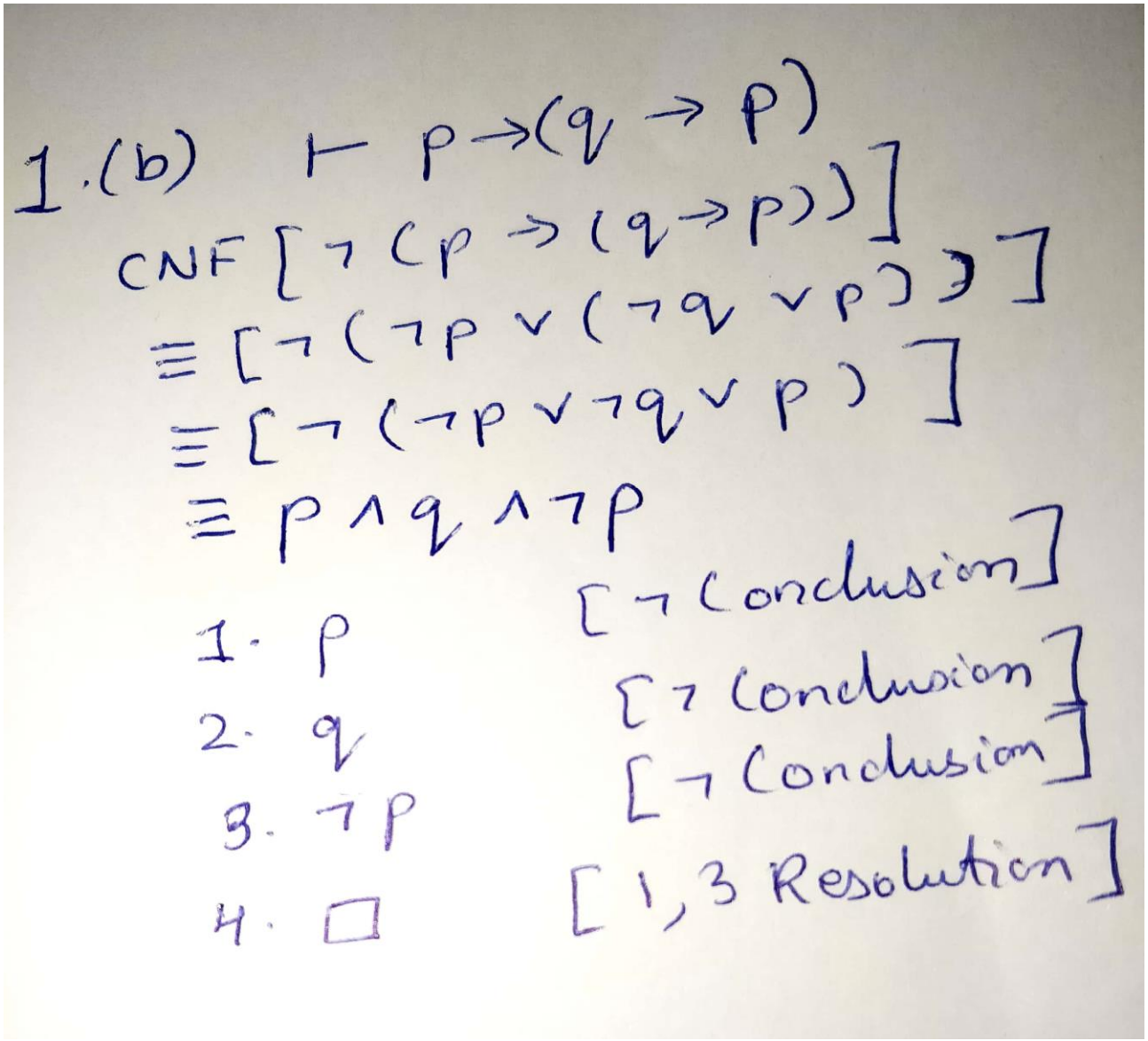
| P | Q | R | (Q V R) | P ∧ (Q V R) | (P ∧ Q) V (P ∧ R) |
|---|---|---|---|---|---|
| FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| FALSE | FALSE | TRUE | TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE | TRUE | FALSE | FALSE |
| FALSE | TRUE | TRUE | TRUE | FALSE | FALSE |
| TRUE | FALSE | FALSE | FALSE | FALSE | FALSE |
| TRUE | FALSE | TRUE | TRUE | TRUE | TRUE |
| TRUE | TRUE | FALSE | TRUE | TRUE | TRUE |
| TRUE | TRUE | TRUE | TRUE | TRUE | TRUE |




Thus, it is holds semantically using truth table and is proven syntactically using resolution.

(b)

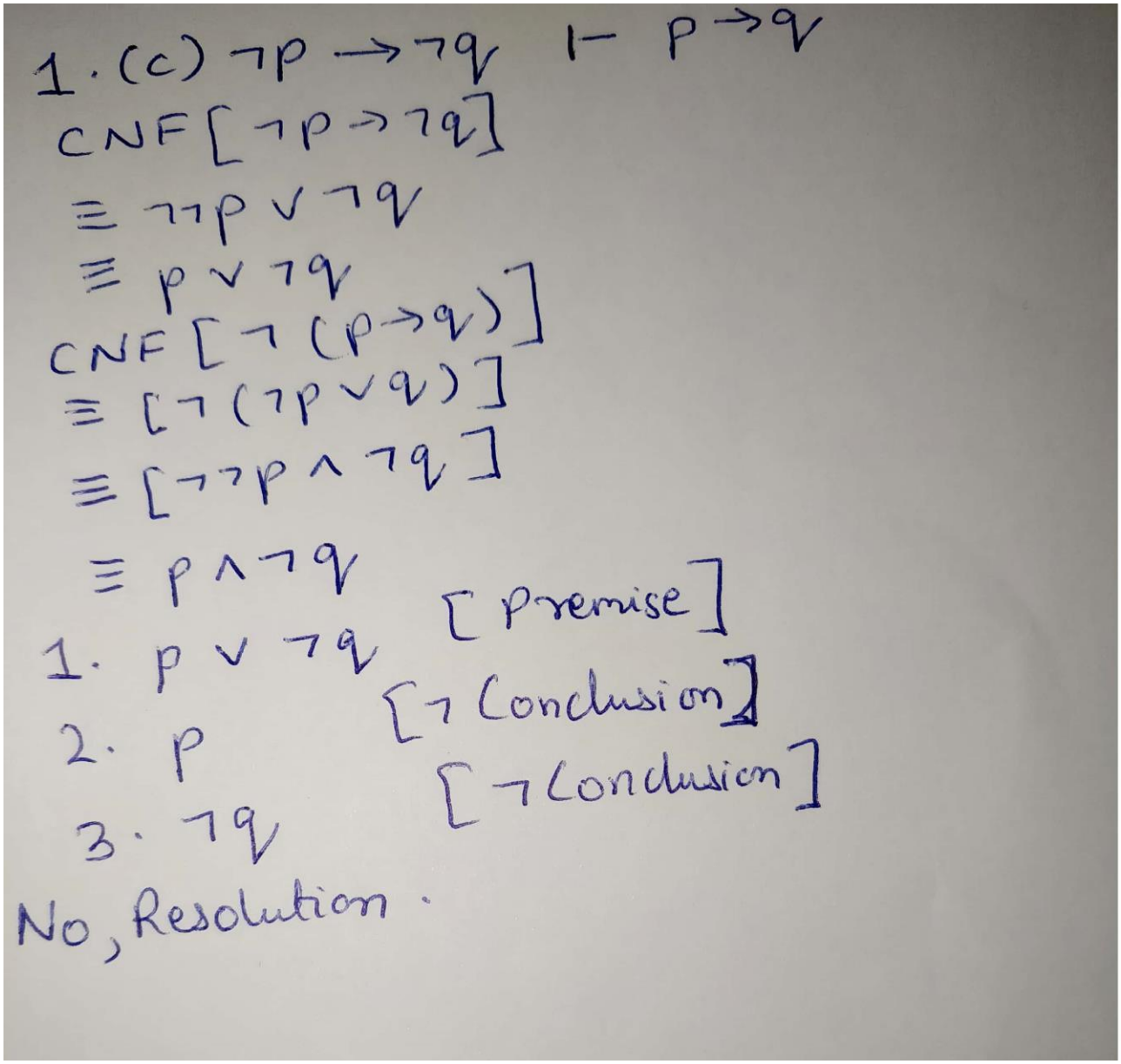
| P | Q | (Q → P) | P → (Q → P) |
|---|---|---|---|
| FALSE | FALSE | TRUE | TRUE |
| FALSE | TRUE | FALSE | TRUE |
| TRUE | FALSE | TRUE | TRUE |
| TRUE | TRUE | TRUE | TRUE |




Thus, it is holds semantically using truth table and is proven syntactically using resolution.

(c)

| P | Q | ¬ P | ¬ Q | ¬ P → ¬ Q | P → Q |
|---|---|---|---|---|---|
| FALSE | FALSE | TRUE | TRUE | TRUE | TRUE |
| FALSE | TRUE | TRUE | FALSE | FALSE | TRUE |
| TRUE | FALSE | FALSE | TRUE | TRUE | FALSE |
| TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |




Does not holds semantically using truth and cannot be proven syntactically using resolution.
(d)

| P | Q | ¬ P → ¬ Q | ¬ Q → ¬ P | P ↔ Q |
|---|---|---|---|---|
| FALSE | FALSE | TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE | TRUE | FALSE |
| TRUE | FALSE | TRUE | FALSE | FALSE |
| TRUE | TRUE | TRUE | TRUE | TRUE |

1. (d) $\neg p \to \neg q, \neg q \to \neg p \vdash p \leftrightarrow q$

CNF$[\neg p \to \neg q]$ , CNF$[\neg q \to \neg p]$

$\equiv \neg\neg p \vee \neg q$ $\equiv \neg\neg q \vee \neg p$

$\equiv p \vee \neg q$ $\equiv q \vee \neg p$

CNF$[\neg (p \leftrightarrow q)]$

$\equiv [\neg ((p \to q) \wedge (q \to p))]$

$\equiv [\neg ((\neg p \vee q) \wedge (\neg q \vee p))]$

$\equiv [\neg (\neg p \vee q) \vee \neg (\neg q \vee p)]$

$\equiv [(p \wedge \neg q) \vee (q \wedge \neg p)]$

$\equiv (p \vee q) \wedge (p \vee \neg p) \wedge (\neg q \vee q)$

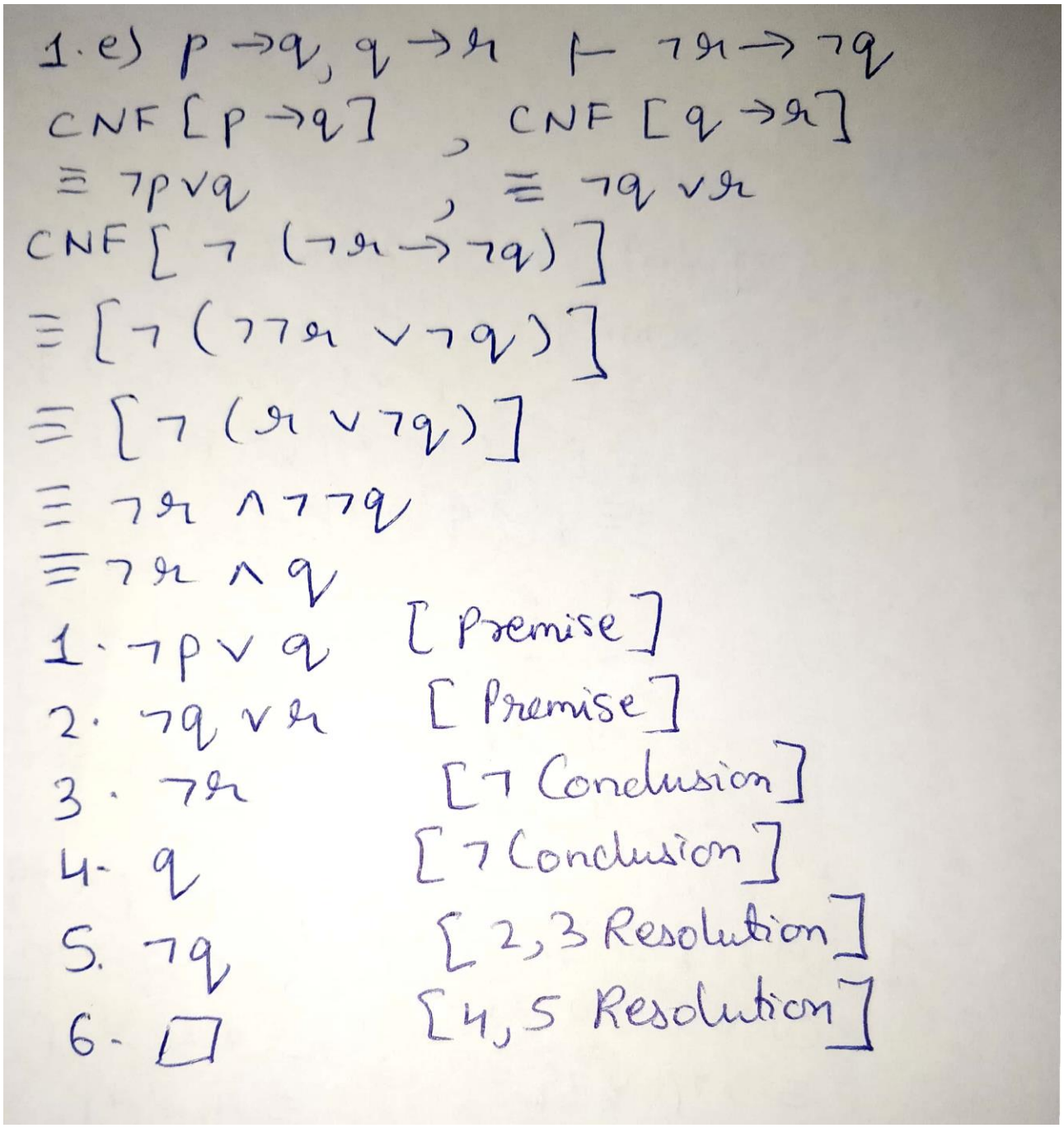
$\wedge (\neg q \vee \neg p)$

1. $p \vee \neg q$ [Premise]
2. $q \vee \neg p$ [Premise]
3. $p \vee q$ [¬ Conclusion]
4. $p \vee \neg p$ [¬ Conclusion]
5. $\neg q \vee q$ [¬ Conclusion]
6. $\neg q \vee \neg p$ [¬ Conclusion]
7. $\neg q$ [1, 6 Resolution]
8. $q$ [2, 3 Resolution]
9. $\square$ [7, 8 Resolution]

It holds semantically and can be proven syntactically using resolution.

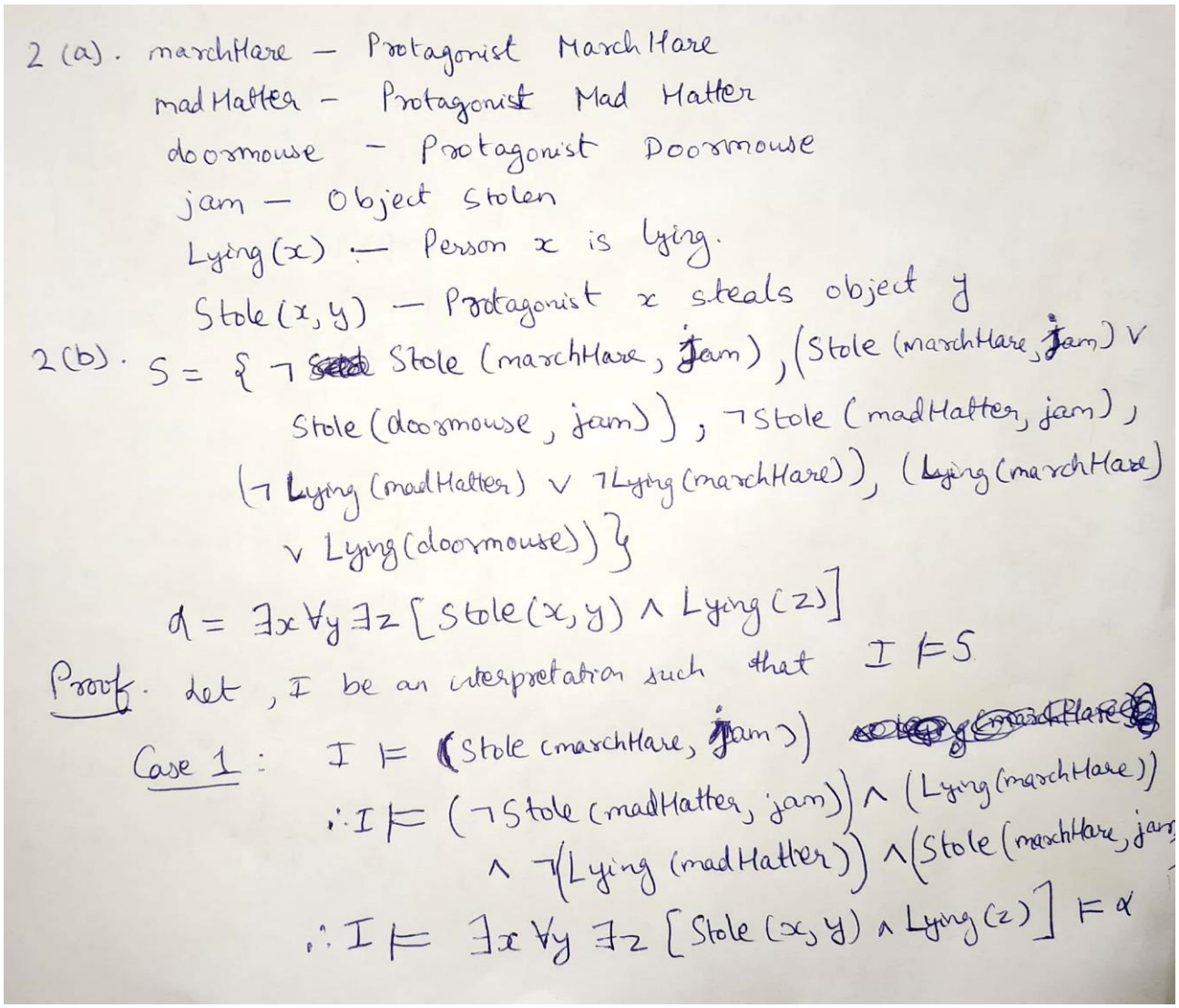
(e)

| P | Q | R | ¬ Q | ¬ R | P → Q | Q → R | ¬ R → ¬ Q |
|---|---|---|---|---|---|---|---|
| FALSE | FALSE | FALSE | TRUE | TRUE | TRUE | TRUE | TRUE |
| FALSE | FALSE | TRUE | TRUE | FALSE | TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE | FALSE | TRUE | TRUE | FALSE | FALSE |
| FALSE | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE | TRUE | TRUE | FALSE | TRUE | TRUE |
| TRUE | FALSE | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| TRUE | TRUE | FALSE | FALSE | TRUE | TRUE | FALSE | FALSE |
| TRUE | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE | TRUE |




It semantically follows and holds as well using truth table and can be proven using resolution.

**ANSWER – 2**



2 (a). marchHare – Protagonist MarchHare
madHatter – Protagonist Mad Hatter
doormouse – Protagonist Doormouse
jam – Object Stolen
Lying(x) – Person x is lying.
Stole(x, y) – Protagonist x steals object y

2(b). $S = \{ \neg$ Stole(marchHare, jam), (Stole(marchHare, jam) $\vee$ Stole(doormouse, jam)), $\neg$Stole(madHatter, jam), ($\neg$ Lying(madHatter) $\vee$ $\neg$Lying(marchHare)), (Lying(marchHare) $\vee$ Lying(doormouse)) $\}$

$\alpha = \exists x \forall y \exists z [Stole(x, y) \wedge Lying(z)]$

Proof. Let, I be an interpretation such that $I \models S$

Case 1: $I \models$ (Stole(marchHare, jam))

$\therefore I \models (\neg Stole(madHatter, jam)) \wedge (Lying(marchHare)) \wedge \neg(Lying(madHatter)) \wedge (Stole(marchHare, jam$

$\therefore I \models \exists x \forall y \exists z [Stole(x, y) \wedge Lying(z)] \models \alpha$

CONTINUED BELOW……

Case 2: $I \not\models (Stole(marchHare, jam))$

$\therefore I \models \neg(Stole(marchHare, jam)) \wedge Stole(doormouse, jam)$

$\wedge (\neg Lying(madHatter) \vee \neg Lying(marchHare)) \wedge$

$(Lying(doormouse))$

$\therefore I \models \exists x \forall y \exists z [Stole(x,y) \wedge Lying(z)]$

In both cases for $I$, $I \models S$, then $I \models \alpha$, therefore $S \models \alpha$

Thus, we can say that the case of Stole(marchHare, jam) follows logical interpretation and conclude that marchHare Stole the jam.

2 (d.)

1. $\neg Stole(marchHare, jam)$ [Premise]
2. $Stole(marchHare, jam) \vee Stole(doormouse, jam)$ [Premise]
3. $\neg Stole(madHatter, jam)$ [Premise]
4. $\neg Lying(madHatter) \vee \neg Lying(marchHare, jam)$ [Premise]
5. $Lying(marchHare, jam) \vee Lying(doormouse, jam)$ [Premise]
6. $\therefore$ The conclusion is $[(Stole(marchHare, jam) \wedge \neg Stole(doormouse, jam) \wedge \neg Stole(madHatter, jam)) \rightarrow (Lying(marchHare) \wedge \neg Lying(madHatter) \wedge \neg Lying(doormouse))]$
7. $Stole(marchHare, jam)$ [$\neg$ Conclusion]
8. $\neg Stole(doormouse, jam)$ [$\neg$ Conclusion]
9. $\neg Stole(madHatter, jam)$ [$\neg$ Conclusion]
10. $\neg Lying(marchHare) \vee Lying(madHatter) \vee Lying(doormouse)$
11. $\square$ [1, 6 Resolution]

**ANSWER – 4**

(a)

DATR is a language for lexical knowledge representation. Here, the information is organised as a network of nodes information is organised as a network of nodes, where a node is essentially just a collection of closely related information. In the context of lexical description, a node typically corresponds to a word, a lexeme or a class of lexemes. For example, we might have a node describing an abstract verb, another for the subcase of a transitive verb, another for the lexeme love and still more for the individual words that are instances of this lexeme (love, loves, loved, loving, etc.). Each node has associated with it a set of path/value pairs where a path is a sequence of atoms (which are primitive objects), and a value is an atom or a sequence of atoms. We will sometimes refer to atoms in paths as attributes.

The syntax of DATR uses four classes of lexical token: nodes, atoms, variables and reserved symbols. The complete list of reserved symbols is as follows:

: " < > = == . ' % #

Single quotes can be used to form atoms that would otherwise be ill-formed as such; % is used for end-of-line comments, following the Prolog convention; # is used to introduce declarations and other compiler directives. The other classes, nodes, atoms and variables, must be distinct, and distinct from the reserved symbols, but are otherwise arbitrary. For this, we have already adopted the convention that both nodes and atoms are simple words, with nodes starting with uppercase letters. We extend this convention to variables, which we require to start with the character $. And we take white-space (spaces, newlines, tabs, etc.) to delimit lexical tokens but otherwise to be insignificant. Thus right hand side expressions are more like *atom*,$var,etc.

Then there are three kinds of local inheritance descriptor: a node, an (evaluable) path, and a node/path pair. Nodes are primitive tokens, paths are descriptor sequences (defined below) enclosed in angle brackets and node/path pairs consist of a node and a path separated by a colon:

*Node1*

*<desc1 desc2 desc3 …>*

*Node1:<desc1 desc2 desc3 …>*

We put a global descriptor by putting the above inside two double quotes(“ start-end).

A descriptor sequence is a (possibly empty) sequence of descriptors. The recursive definition of evaluable paths in terms of descriptor sequences allows arbitrarily complex expressions to be constructed, such as:

*“Node1:<”<atom1>” Node2:<atom2>>”*

*“<”<”<Node1:<atom1 atom2> atom3>” Node2 “<atom4 atom5>” <> >”>”*

All the DATR definitional sentences are of the form –

*Node: Path == Def*

Here, a node can have more than just one path and definitions can be multiple too.

(b)

**Example –**

A node describing “love” word as Word1 can be -

```
Word1:
```

```
        <syn cat>  = verb
        <syn type> = main
        <syn form> = present participle
        <mor form> = love ing.
```

Similarly, a passive form of “love” can be –

```
Word2:
        <syn cat>  = verb
        <syn type> = main
        <syn form> = passive participle
        <mor form> = love ed.
```

This can easily be represented into the DATR definitional sentences by putting one more = sign in between the left hand side and the right hand side of each line.

Although this change does not itself make the description more concise, it allows us to introduce other ways of describing values in definitional statements, in addition to simply specifying them. Such value descriptors will include inheritance specifications which allow us to gather together the properties that Word1 and Word2 have solely by virtue of being verbs. We start by introducing a VERB node:

```
VERB:
        <syn cat>  == verb
        <syn type> == main.
```

And then after introducing this concept for inheritance which the DATR language offers is –

```
    Word1:
        <syn cat>  == VERB:<syn cat>
        <syn type> == VERB:<syn type>
        <syn form> == present participle
        <mor form> == love ing.
    Word2:
        <syn cat>  == VERB:<syn cat>
        <syn type> == VERB:<syn type>
        <syn form> == passive participle
        <mor form> == love ed.
```

Since, in the example the following appears to be leading to inherit VERB it can be reduce to just one case using –

```
Word1:
        <> == VERB.
```

Also, simplifying more to adjust the above discussed value descriptors we finally make changes to both as –

```
    VERB:
        <syn cat> == verb
        <syn type> == main.
    Love:
        <> == VERB
        <mor root> == love.
    Word1:
        <> == Love
        <syn form> == present participle
        <mor form> == <mor root> ing.
    Word2:
        <> == Love
        <syn form> == passive participle
        <mor form> == <mor root> ed.
```

Thus, by keeping the atom statements of the node can changing them we can create a sequence more complex than what we created above and form a much more detailed

knowledge representation Since, inference is simple as just expanding each node until we reach an atom.
(c)
This method is similar to a graph problem as it uses the analogy of it. Thus advantage is that it is easy to develop, debug and delete. But the issues can be when you have to update, insert in middle of the path while changing the definition of it as well as deleting a node path which has an inheritance for a different node path value descriptor.

More information is available at : - http://www.ccl.kuleuven.ac.be/LKR/html/datr.html

**ANSWER – 3**

# INTRODUCTION

Satisfiability for 4-SAT problems exhibit easy-hard-easy pattern. Just like a 3-SAT problem we made use of the satisfiability problem of sets of clauses that are small in relation to the total number of distinct propositional variables and of the sets of clauses that are large in relation to the total number of distinct propositional variables. Since both the scenarios have either fewer constraints to set the truth values for the variables thus making the problem easily satisfiable or too many constraints to assign truth values for the variables making it unsatisfiable.

Prerequisites – Use of exactly 4 literals in disjunctions. For example, { p v q v r v s, ¬ p v ¬t v ¬ a v ¬ b } but no constraint on the number of variables, clauses to use.

# TESTING

I have tried the 6 different cases for which the satisfiability was limited to both the extremes and was not hard to solve. They all contain the testing result of the minisat program. Even then some cases have un-satisfiability problem, but the time taken to solve the problem is very less. Thus, specifying that the satisfiability was easy to determine in the below cases and use the steps given below to run the program.

Executing,
$ python generate.py
Enter the number of propositional variables:
Yourvalue
Enter the number of clauses to generate randomly:
Yourvalue
$ ~morri/bin/minisat file.cnf
Will generate the below results

## Case 1:

Number of variables: 100
Number of clauses: 1473
restarts: 284
conflicts: 107873 (108306 /sec)
decisions: 124131 (0.00 % random) (124630 /sec)
propagations: 1845984 (1853398 /sec)

conflict literals: 1159895 (21.09 % deleted)
Memory used: 5.00 MB
CPU time: 0.996 s
UNSATISFIABLE

## Case 2:

Number of variables: 50
Number of clauses: 1450
restarts: 2
conflicts: 198 (inf /sec)
decisions: 213 (0.00 % random) (inf /sec)
propagations: 1630 (inf /sec)
conflict literals: 856 (22.32 % deleted)
Memory used: 5.00 MB
CPU time: 0 s
UNSATISFIABLE

## Case 3:

Number of variables: 30
Number of clauses: 464
restarts: 2
conflicts: 125 (inf /sec)
decisions: 138 (0.00 % random) (inf /sec)
propagations: 920 (inf /sec)
conflict literals: 517 (21.19 % deleted)
Memory used: 5.00 MB
CPU time: 0 s
UNSATISFIABLE

## Case 4:

Number of variables: 150
Number of clauses: 150
restarts: 1
conflicts: 0 (-nan /sec)
decisions: 1 (0.00 % random) (inf /sec)
propagations: 0 (-nan /sec)
conflict literals: 0 (-nan % deleted)
Memory used: 5.00 MB
CPU time: 0 s
SATISFIABLE

## Case 5:

Number of variables: 1486
Number of clauses: 150
restarts: 1
conflicts: 0 (-nan /sec)
decisions: 1 (0.00 % random) (inf /sec)
propagations: 0 (-nan /sec)
conflict literals: 0 (-nan % deleted)

Memory used: 5.00 MB
CPU time: 0 s
SATISFIABLE

## Case 6:

Number of variables: 150
Number of clauses: 1400
restarts: 7676
conflicts: 4331290 (72439 /sec)
decisions: 5191849 (0.00 % random) (86832 /sec)
propagations: 113728245 (1902065 /sec)
conflict literals: 90315255 (15.31 % deleted)
Memory used: 5.00 MB
CPU time: 59.792 s
SATISFIABLE

# CONCLUSION

Since, the assumption that 4-SAT is like 3-SAT problem we now had to determine the in-between of range of the ratio. Finally, I came up with the empirical value for a constant $C \approx 9.75$ because I was getting multiple values like 9.9,10 but I chose the minimum average value out of all the random testing scenarios. Since I believe that the value that I have selected is the minimum value found empirically. Thus, the ratio of number of clauses to the number of variables is used to determine the constant value mentioned above.

The cases below shows the range of cases where the sat-solver was hard to determine, and it was stopped after a long time of just generating more conflicts/program not terminating as well. Thus the curve went to the peak in the graph of c value and cpu time.

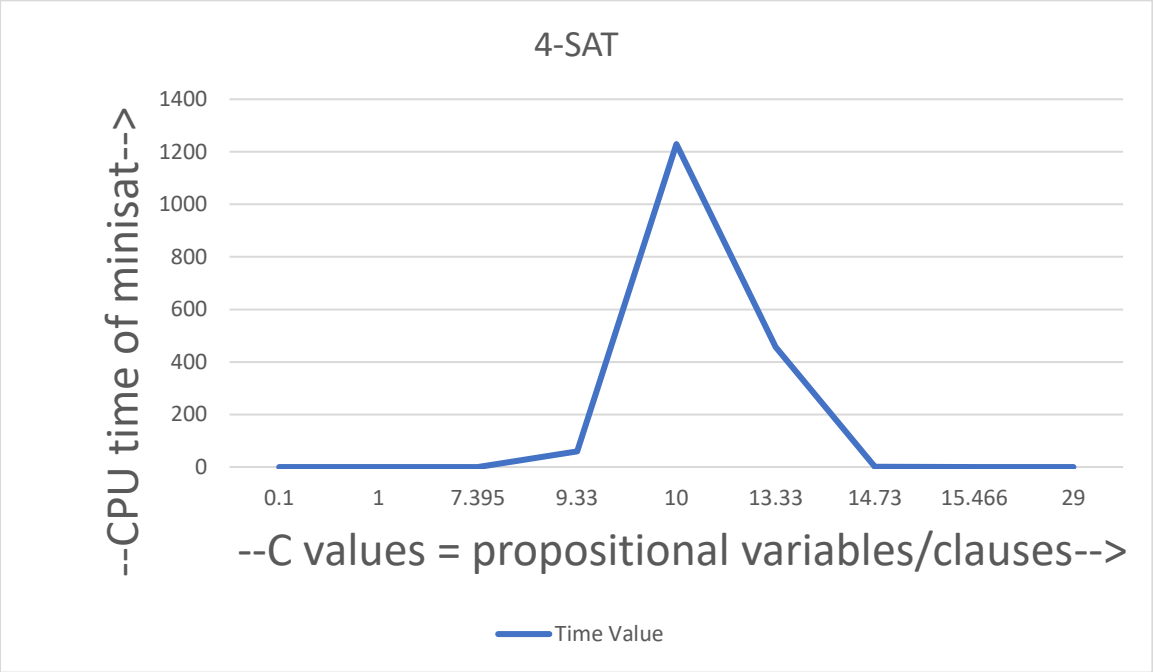
## CASE 1:

Number of variables: 150
Number of clauses: 2000
restarts: 32767
conflicts: 23480882 (51416 /sec)
decisions: 27469981 (0.00 % random) (60150 /sec)
propagations: 548136379 (1200243 /sec)
conflict literals: 388488027 (19.00 % deleted)
Memory used: 11.00 MB
CPU time: 456.688 s
INDETERMINATE

## CASE 2:

Number of variables: 150
Number of clauses: 1500
restarts: 85432
conflicts: 65763643 (53495 /sec)
decisions: 77513578 (0.00 % random) (63053 /sec)
propagations: 1670099599 (1358538 /sec)
conflict literals: 1300132115 (17.30 % deleted)

Memory used: 12.00 MB
CPU time: 1229.34 s
INDETERMINATE




Although, the cpu time is not an important measuring aspect of determining the pattern but it helps us identify the co-relation with the constant value of "C". In the given test cases the progress of resolving a conflict gives us a good estimate of the sat-solver actually solving the problem.

The most important measure were the propositional variables and the randomly generated clauses.