COMP4418, 2018–Assignment 1

Student ID: Z5097690 Name: Baixiang Guan

Question 1.

1.

(i) using truth table:

(a). $\boxed{\text{Yes}}$: $p \wedge (q \vee r) \models (p \wedge q) \vee (p \wedge r)$

p	q	r	$q \vee r$	$p \wedge q$	$p \wedge r$	$p \wedge (q \vee r)$	$(p \wedge q) \vee (p \wedge r)$
T	T	T	T	T	T	T	T
T	T	F	T	T	F	T	T
T	F	T	T	F	T	T	T
T	F	F	F	F	F	F	F
F	T	T	T	F	F	F	F
F	T	F	T	F	F	F	F
F	F	T	T	F	F	F	F
F	F	F	F	F	F	F	F

when $p \wedge (q \vee r)$ is true, $(p \wedge q) \vee (p \wedge r)$ is also true. Therefore, inference is vaild.

(b) $\boxed{\text{Yes.}}$ $\models p \to (q \to p)$

p	q	$q \to p$	$p \to (q \to p)$
T	T	T	T
T	F	T	T
F	T	F	T
F	F	T	T

we can see from the ~~true~~ trueth table. which shows the $p \to (q \to p)$ is always true no matter what p or q values. Therefore, $p \to (q \to p)$ is a tautology.

(c). $\boxed{\text{NO.}}$

p	q	$\neg p \to \neg q$	$p \to q$
T	T	T	T
T	F	T	F
P	T	F	T
F	F	T	T

In the. second row where $\neg p \to \neg q$ is true, but $p \to q$ is false. therefore, inference is not vaild.

(d) $\boxed{\text{Yes.}}$

p	q	$\neg p \to \neg q$	$\neg q \to \neg p$	$p \to q$	$q \to p$	$p \leftrightarrow q$
T	T	T	T	T	T	T
T	F	T	F	F	T	F
F	T	F	T	T	F	F
F	F	T	T	T	T	T

we can see from the trueth table, when both $\neg p \to \neg q$ and $\neg q \to \neg p$ are true, then $p \leftrightarrow q$ is true. Therefore, $\neg p \to \neg q, \neg q \to \neg p \models p \leftrightarrow q$ is vaild.

(e). $\boxed{\text{Yes.}}$

p	q	r	$p \to q$	$q \to r$	$\neg r \to \neg q$
T	T	T	T	T	T
T	T	F	T	F	F
T	F	T	F	T	T
T	F	F	F	T	T
F	T	T	T	T	T
F	T	F	T	F	F
F	F	T	T	T	T
F	F	F	T	T	T

When both $p \to q$, and $q \to r$ are true, $\neg r \to \neg q$ is also. true. ~~Hence~~ Hence, Inference is vaild.

And also can proof by $q \to r$. is equal to $\neg r \to \neg q$.

So, no matter $p \to q$ is true or false.

(ii) Resolution

1. using resolution:

(a). $p\wedge(q\vee r) \vdash (p\wedge q)\vee(p\wedge r)$ Yes

$CNF[p\wedge(q\vee r)] \equiv p\wedge(q\vee r)$

$CNF[(p\wedge q)\vee(p\wedge r)] \equiv p\wedge(q\vee r)$

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

1. p [premise]
2. $q\vee r$ [premise]
3. $\neg p\vee\neg q$ [¬conclusion]
4. $\neg p\vee\neg r$ [¬conclusion]
5. $\neg q$ [1.3 resolution]
6. r [2.5 resolution]
7. $\neg p$ [4.6 resolution]
8. $\square$ [1.7 resolution]

(b). $\vdash p\rightarrow(q\rightarrow p)$. Yes.

$CNF[\neg(p\rightarrow(q\rightarrow p))] \equiv p\wedge q\wedge\neg p$

1. p [¬conclusion]
2. q [¬conclusion]
3. $\neg p$ [¬conclusion]
4. $\square$ [1.3. resolution]

(c) $\neg p\rightarrow\neg q \vdash p\rightarrow q$ No.

~~$CNF[\neg p\rightarrow\neg q] \equiv$~~

$p\rightarrow q \equiv \neg p\vee q$.

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

1. $p\vee\neg q$ [premise]
2. p [¬conclusion]
3. $\neg q$ [¬conclusion]
4. $p\vee\neg q$ [1.2. resolution].
5. $p\vee\neg q$ [3.4 resolution].

⋮

we can not resolution to $\square$, ~~know (p or ¬q)~~ ~~is true or both~~. a ~~p and ¬q are~~ ~~true~~.

(d). $\neg p\rightarrow\neg q,\ \neg q\rightarrow\neg p \vdash p\leftrightarrow q$ Yes.

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

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

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

1. $p\vee\neg q$ [premise]
2. $q\vee\neg p$ [premise]
3. $p\vee q$ [¬conclusion]
4. $\neg q\vee\neg p$ [¬conclusion]
5. p [1.3 resolution]
6. q [2.5 resolution]
7. $\neg p$ [4.6 resolution]
8. $\square$ [5.7 resolution]

(e). $p\rightarrow q,\ q\rightarrow r \vdash \neg r\rightarrow\neg q$ Yes

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

$q\rightarrow r \equiv \neg q\vee r$

$CNF[\neg(\neg r\rightarrow\neg q)] \equiv \neg r\wedge q$

1. $\neg p\vee q$ [premise]
2. $\neg q\vee r$ [premise]
3. $\neg r$ [¬conclusion]
4. q [¬conclusion]
5. $\neg q$ [2.4 resolution]
6. $\square$ [4.5 resolution]

Question 2.

2(a).

1. "I never stole the jam!" pleaded the March Hare.

   $\neg Stole(marchHare, jam)$.

2. "No, no" pleaded the Hatter. "One of us stole it, but it wasn't me!".

   $\exists x\,[Stole(x, jam) \wedge \neg Stole(madHatter, jam)]$

3. "At least one of them did" replied the Dormouse.

   $\exists x\,[(x = marchHare \vee x = madHatter) \rightarrow \neg lying(x)]$

4. the March Hare and the Dormouse were not both speaking the truth.

   $\exists x\,[(x = marchHare \vee x = dormouse) \rightarrow lying(x)]$

2(b):

(b). $\boxed{\text{Yes, marchHare stole the jam}}$

Firstly, I need ensure that because the paragraph describe the jam was stole which means that only one stealer stole the jam. It can be proved by using formalisation in part (2a).

Proof Semantically:

$S = \{\neg stole(marchHare, jam),\ \exists x[stole(x, jam) \wedge \neg Stole(madHatter, jam)] \rightarrow [(Stole(marchHare, jam) \vee Stole(dormouse, jam)) \wedge \neg Stole(madHatter, jam)],$

$\exists x\,[(x = marchHare \vee x = madHatter) \rightarrow \neg lying(x)] \rightarrow [\neg lying(marchHare) \vee \neg lying(madHatter) \vee (\neg lying(marchHare) \wedge \neg lying(madHatter))]$

$\exists x\,[(x = marchHare \vee x = dormouse) \rightarrow lying(x)] \rightarrow [lying(marchHare) \vee lying(dormouse) \vee (lying(marchHouse) \wedge lying(doormouse))].\}$

$\alpha = \exists x\,[stole(x, jam)]$

claim: $S \models \alpha$

Proof. Let I be any Interpretation such that $I \models S$

Case 1: $I \models \neg lying(marchHare)$

because $lying(marchHare) \vee lying(dormouse)$ (form 4)

$\therefore I \models lying(doormouse)$

because doormouse is lying.

$\therefore \neg \exists x\,[(x = marchHare \vee x = madHatter) \rightarrow \neg lying(x)] \equiv \forall x\,[(x = marchHare \vee x = madHatter) \wedge lying(x)]$

which means that both marchHare and madHatter are lying.

$\therefore I \models lying(marchHare)$

Hence, it is conflict with the asuption.

Case 2: $I \models lying(marchHare) \land lying(dormouse)$.

because. $lying(marchHare)$

$\therefore \neg\neg Stole(marchHare, iam) \equiv stole(marchHare, iam)$.

$I \models \alpha$.

but. $I \models lying(dormouse)$

which means that:

$I \models lying(marchHare) \land lying(madHatter)$

$\therefore I \not\models \neg \exists x [Stole(x, iam) \land \neg Stole(madHatter, iam)] \equiv \forall x [\neg stole(X, iam) \lor stole(madHatter, iam)]$

This is conflict with stole(marchHare, iam). because. there is only one stealer.

$\therefore$ Case 2 can not be success.

Case 3: $I \models lying(marchHare) \land \neg lying(dormouse)$.

because. $lying(marchHare)$.

$\therefore \neg\neg Stole(marchHare, iam) \equiv Stole(marchHare, iam)$

$\therefore I \models \alpha$

And

$I \models \neg lying(doormouse)$

because. marchHare is lying and dormouse is not lying.

which means that ~~¬(madHatter~~ $\neg lying(madHatter)$ must be True.

due to the $\exists x [\cancel{stole} (x = marchHare \lor x = madHatter) \rightarrow \neg lying(x)]$.

$\therefore I \models \neg lying(madHatter)$.

$\therefore$ madHatter is Trueth. then. he is not stoler. and one stoler is marchHare because ~~they~~ he lying.

$\therefore I \models \alpha$

Either way, for any I, if $I \models S$ the $I \models \alpha$.

so $S \models \alpha$. QED.

PS: This poof iuse assume only one stealer to do, If it can be more than one stealer. then we just add a condition that limit number of stealer. ~~which means~~ After that, the poof is the same with above.

Question 3.

3\. Firstly because 3-SAT is a NP-complete problem, and we can use that to prove 4-SAT is also a NP-complete problem.

3-SAT : $A = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge \cdots \wedge (x_{n-2} \vee x_{n-1} \vee x_n)$

4-SAT : $B = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_5 \vee x_6 \vee x_7 \vee x_8) \wedge \cdots \wedge (x_{n-3} \vee x_{n-2} \vee x_{n-1} \vee x_n)$

It is easily to prove that:

$$(x_1 \vee x_2 \vee x_3) \equiv (x_1 \vee x_2 \vee x_3 \vee y) \wedge (x_1 \vee x_2 \vee x_3 \vee \neg y)$$

Hence.

3-SAT can be changed to

$$A = (x_1 \vee x_2 \vee x_3 \vee y_1) \wedge (x_1 \vee x_2 \vee x_3 \vee \neg y_1) \wedge (x_4 \vee x_5 \vee x_6 \vee y_2) \wedge (x_4 \vee x_5 \vee x_6 \vee \neg y_2) \cdots$$

which the formular is same with 4-SAT

$\therefore$ 4-SAT is also a NP-complete problem.

Today, we assume in the 4-SAT, the clauses just use Horn clauses. And I code a program which can random generate the test file ~~SAT~~. And the content is DIMACS format. The out come of programming at ~~blow~~ below.

When random generate number of variables and clauses.

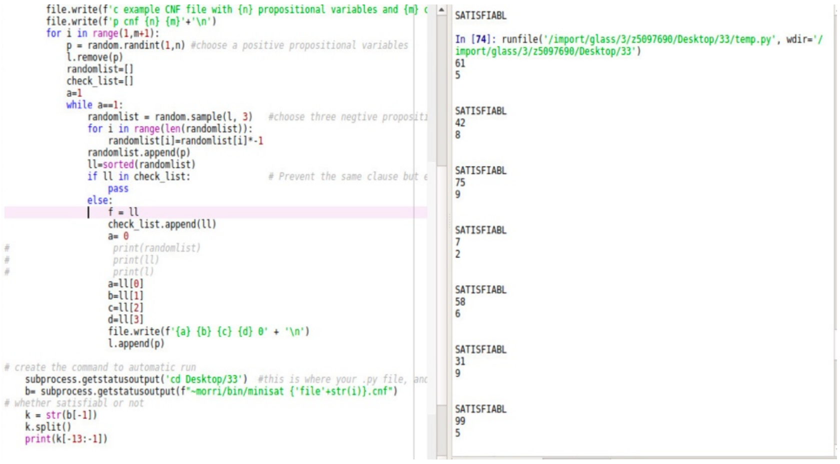
```python
import subprocess
import random

for i in range(20):            # generate 10 files
    n = random.randint(4,100)  # number of varibals
    m = random.randint(1,10)  # m clauses
    l=[]
    for j in range(n):
        l.append(j+1)          #create varibals


    print(n)                   #print varibal and clauses number
    print(m)
    with open(f"{'file'+str(i)}.cnf",'w') as file:
        file.write(f'c example CNF file with {n} propositional variables and {m} c
        file.write(f'p cnf {n} {m}'+'\n')
        for i in range(1,m+1):
            p = random.randint(1,n) #choose a positive propositional variables
            l.remove(p)
            randomlist=[]
            check_list=[]
            a=1
            while a==1:
                randomlist = random.sample(l, 3)   #choose three negtive propositi
                for i in range(len(randomlist)):
                    randomlist[i]=randomlist[i]*-1
                randomlist.append(p)
                ll=sorted(randomlist)
                if ll in check_list:               # Prevent the same clause but e
                    pass
                else:
                    f = ll
                    check_list.append(ll)
                    a= 0
#                    print(randomlist)
#                    print(ll)
#                    print(l)
                    a=ll[0]
```

```
SATISFIABL

In [72]: runfile('/import/glass/3/z5097690/Desktop/33/temp.py', wdir='/
import/glass/3/z5097690/Desktop/33')
70
10


SATISFIABL
87
2


SATISFIABL
30
1


SATISFIABL
49
8


SATISFIABL
43
6


SATISFIABL
94
6


SATISFIABL
60
5
```

IPython console | History log

```python
    with open(f"{'file'+str(i)}.cnf",'w') as file:
        file.write(f'c example CNF file with {n} propositional variables and {m} c
        file.write(f'p cnf {n} {m}'+'\n')
        for i in range(1,m+1):
            p = random.randint(1,n) #choose a positive propositional variables
            l.remove(p)
            randomlist=[]
            check_list=[]
            a=1
            while a==1:
                randomlist = random.sample(l, 3)   #choose three negtive propositi
                for i in range(len(randomlist)):
                    randomlist[i]=randomlist[i]*-1
                randomlist.append(p)
                ll=sorted(randomlist)
                if ll in check_list:               # Prevent the same clause but e
                    pass
                else:
                    f = ll
                    check_list.append(ll)
                    a= 0
#                    print(randomlist)
#                    print(ll)
#                    print(l)
                    a=ll[0]
                    b=ll[1]
                    c=ll[2]
                    d=ll[3]
                    file.write(f'{a} {b} {c} {d} 0' + '\n')
                    l.append(p)

# create the command to automatic run
    subprocess.getstatusoutput('cd Desktop/33')  #this is where your .py file, anc
    b= subprocess.getstatusoutput(f"~morri/bin/minisat {'file'+str(i)}.cnf")
# whether satisfiabl or not
    k = str(b[-1])
    k.split()
    print(k[-13:-1])
```

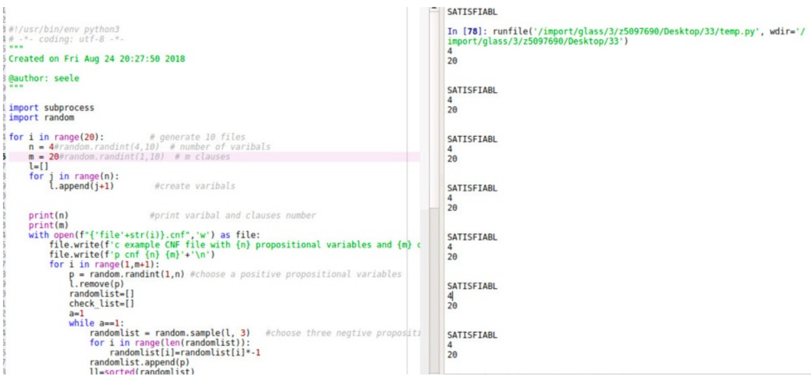
```
SATISFIABL

In [73]: runfile('/import/glass/3/z5097690/Desktop/33/temp.py', wdir='/
import/glass/3/z5097690/Desktop/33')
4
8


SATISFIABL
99
4


SATISFIABL
8
3


SATISFIABL
50
9


SATISFIABL
54
2


SATISFIABL
70
7


SATISFIABL
53
3
```



```
        file.write(f'c example CNF file with {n} propositional variables and {m} c
        file.write(f'p cnf {n} {m}'+'\n')
        for i in range(1,m+1):
            p = random.randint(1,n) #choose a positive propositional variables
            l.remove(p)
            randomlist=[]
            check_list=[]
            a=1
            while a==1:
                randomlist = random.sample(l, 3)   #choose three negtive propositi
                for i in range(len(randomlist)):
                    randomlist[i]=randomlist[i]*-1
                randomlist.append(p)
                ll=sorted(randomlist)
                if ll in check_list:                # Prevent the same clause but e
                    pass
                else:
                    f = ll
                    check_list.append(ll)
                    a= 0
#                    print(randomlist)
#                    print(ll)
#                    print(l)
                    a=ll[0]
                    b=ll[1]
                    c=ll[2]
                    d=ll[3]
                    file.write(f'{a} {b} {c} {d} 0' + '\n')
                    l.append(p)

# create the command to automatic run
    subprocess.getstatusoutput('cd Desktop/33')  #this is where your .py file, anc
    b= subprocess.getstatusoutput(f"~morri/bin/minisat {'file'+str(i)}.cnf")
# whether satisfiabl or not
    k = str(b[-1])
    k.split()
    print(k[-13:-1])
```

```
SATISFIABL

In [74]: runfile('/import/glass/3/z5097690/Desktop/33/temp.py', wdir='/
import/glass/3/z5097690/Desktop/33')
61
5


SATISFIABL
42
8


SATISFIABL
75
9


SATISFIABL
7
2


SATISFIABL
58
6


SATISFIABL
31
9


SATISFIABL
99
5
```

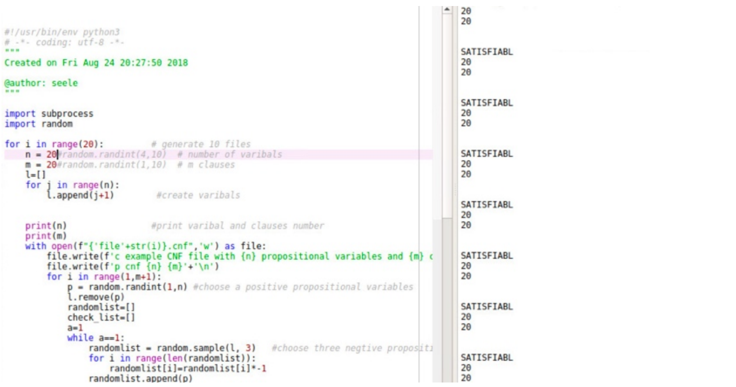
When n<C:



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 24 20:27:50 2018

@author: seele
"""

import subprocess
import random

for i in range(20):         # generate 10 files
    n = 4#random.randint(4,10)  # number of varibals
    m = 20#random.randint(1,10)  # m clauses
    l=[]
    for j in range(n):
        l.append(j+1)        #create varibals


    print(n)                #print varibal and clauses number
    print(m)
    with open(f"{'file'+str(i)}.cnf",'w') as file:
        file.write(f'c example CNF file with {n} propositional variables and {m} c
        file.write(f'p cnf {n} {m}'+'\n')
        for i in range(1,m+1):
            p = random.randint(1,n) #choose a positive propositional variables
            l.remove(p)
            randomlist=[]
            check_list=[]
            a=1
            while a==1:
                randomlist = random.sample(l, 3)   #choose three negtive propositi
                for i in range(len(randomlist)):
                    randomlist[i]=randomlist[i]*-1
                randomlist.append(p)
                ll=sorted(randomlist)
```

```
SATISFIABL

In [78]: runfile('/import/glass/3/z5097690/Desktop/33/temp.py', wdir='/
import/glass/3/z5097690/Desktop/33')
4
20

SATISFIABL
4
20

SATISFIABL
4
20

SATISFIABL
4
20

SATISFIABL
4
20

SATISFIABL
4
20

SATISFIABL
4
20
```

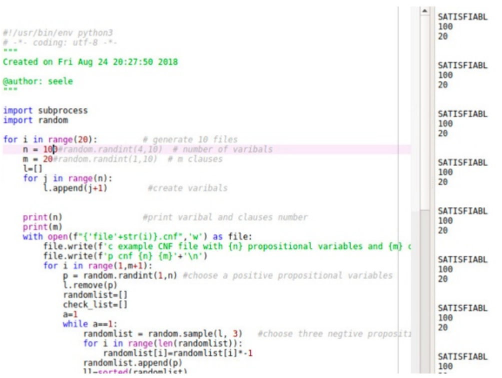
When n =c:



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 24 20:27:50 2018

@author: seele
"""

import subprocess
import random

for i in range(20):         # generate 10 files
    n = 20#random.randint(4,10)  # number of varibals
    m = 20#random.randint(1,10)  # m clauses
    l=[]
    for j in range(n):
        l.append(j+1)        #create varibals


    print(n)                #print varibal and clauses number
    print(m)
    with open(f"{'file'+str(i)}.cnf",'w') as file:
        file.write(f'c example CNF file with {n} propositional variables and {m} c
        file.write(f'p cnf {n} {m}'+'\n')
        for i in range(1,m+1):
            p = random.randint(1,n) #choose a positive propositional variables
            l.remove(p)
            randomlist=[]
            check_list=[]
            a=1
            while a==1:
                randomlist = random.sample(l, 3)   #choose three negtive propositi
                for i in range(len(randomlist)):
                    randomlist[i]=randomlist[i]*-1
                randomlist.append(p)
```

```
20
20

SATISFIABL
20
20

SATISFIABL
20
20

SATISFIABL
20
20

SATISFIABL
20
20

SATISFIABL
20
20

SATISFIABL
20
20

SATISFIABL
20
20
```

When n >c:



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 24 20:27:50 2018

@author: seele
"""

import subprocess
import random

for i in range(20):         # generate 10 files
    n = 100#random.randint(4,10)  # number of varibals
    m = 20#random.randint(1,10)  # m clauses
    l=[]
    for j in range(n):
        l.append(j+1)        #create varibals


    print(n)                #print varibal and clauses number
    print(m)
    with open(f"{'file'+str(i)}.cnf",'w') as file:
        file.write(f'c example CNF file with {n} propositional variables and {m} c
        file.write(f'p cnf {n} {m}'+'\n')
        for i in range(1,m+1):
            p = random.randint(1,n) #choose a positive propositional variables
            l.remove(p)
            randomlist=[]
            check_list=[]
            a=1
            while a==1:
                randomlist = random.sample(l, 3)   #choose three negtive propositi
                for i in range(len(randomlist)):
                    randomlist[i]=randomlist[i]*-1
                randomlist.append(p)
                ll=sorted(randomlist)
```

```
SATISFIABL
100
20

SATISFIABL
100
20

SATISFIABL
100
20

SATISFIABL
100
20

SATISFIABL
100
20

SATISFIABL
100
20

SATISFIABL
100
20

SATISFIABL
100
```

Explaination: We can see that whatever the number of propositional varibles n is smaller or equal or larger than the number of clauses C, the outcome is always satisifiable. Therefore, there is no easy-hard-easy pattern by using Horn clauses in 4-SAT.

The main reason is that when we use the resolution to resolve the clauses, we can not resolve the $\square$ at the end, which means there always exist a way to satisify the clauses.

Because of that the Horn clauses have two form, positive clause like $[\neg p_1, \neg p_2 \cdots, q]$ negtive clause: $[\neg p_1 \cdots \neg p_n]$. And in the programming it just generate posive clause, because the negtive clause can easily to be proved. So I just code the positive clauses in the programme.

Proof:

1. If all clauses in 4-SAT are Horn clauses with the positive clauses, which means in each clause, there is a positive propositional varible P_i, we can find the $\neg p_i$ in other clauses and use the resolution to resolve there two clauses to generate a new clauses. for the new clauses, we also do this work repeatedly until the last clauses is left or there is no clauses to resolution. the processing like:

because of that we can not resolve the left clauses which means there is no conflict beteen the clauses. So, just assign each positive varible is True and negative varible is False then it is always satisifable.

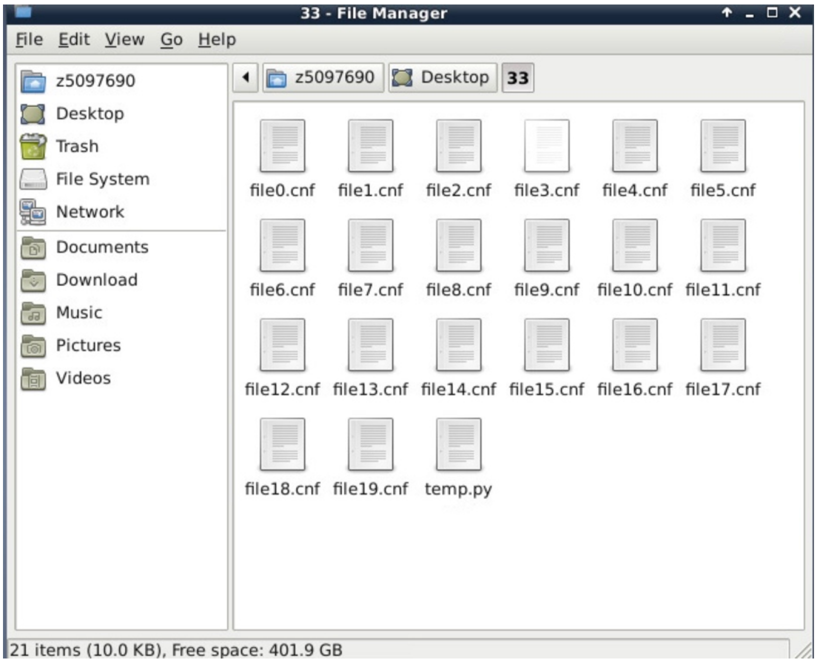
2. If all caluses is negative clauses, which means in each clause there is no positive varible. Then we just assign all varible to be False. then the whole clause is True, because there is no conflict between clauses. Hence, it is always satisifiable

3. If the clauses conclude positive and negative clause, it is same with 1 or 2 at the end of resolution. Therefore it is satisifiable.

PS: the only way can resolve $\square$ is like $\{[\neg p_1, \neg p_2, \neg p_3, q], [p_1, p_2, p_3, \neg q]$ in the 4-SAT or just have conflict between two clauses which are left. However, for Horn clauses, ~~there is no way~~ and in 4-SAT there is no way to achieve two clause conflict at the end. the main reason is each clauses must have 4 varibles and at most one positive varible in the clause. So, the base case is after resolution there are still 2 varibles left. just like: $\{[\neg p, \neg q, \neg r, s], [p, \neg q, \neg r, \neg s]\} \xrightarrow{\text{resolve}} [\neg q \neg r]$

Hence, It is can not be get $\square$ at the end which means that there is always a method to satisify

This is test file and program screenshot:



The code is:

```
ASS1-Q3.py

#code by python 3.6

import subprocess
import random

for i in range(20):            # generate 10 files
    n = random.randint(4,10)   # number of varibals
    m = random.randint(1,10)   # m clauses
    l=[]
    for j in range(n):
        l.append(j+1)          #create varibals


    print(n)                   #print varibal and clauses number
    print(m)
    with open(f"{'file'+str(i)}.cnf",'w') as file:
        file.write(f'c example CNF file with {n} propositional variables and {m} clauses'+'\n')
        file.write(f'p cnf {n} {m}'+'\n')
        for i in range(1,m+1):
            p = random.randint(1,n) #choose a positive propositional variables
            l.remove(p)
            randomlist=[]
            check_list=[]
            a=1
            while a==1:
                randomlist = random.sample(l, 3)   #choose three negtive propositional variables
                for i in range(len(randomlist)):
                    randomlist[i]=randomlist[i]*-1
                randomlist.append(p)
                ll=sorted(randomlist)
                if ll in check_list:               # Prevent the same clause but even same is ok
                    pass
                else:
                    f = ll
                    check_list.append(ll)
                    a= 0
#                    print(randomlist)
#                    print(ll)
#                    print(l)
                    a=ll[0]
                    b=ll[1]
                    c=ll[2]
                    d=ll[3]
                    file.write(f'{a} {b} {c} {d} 0' + '\n')
                    l.append(p)

# create the command to automatic run
    subprocess.getstatusoutput('cd Desktop/33')  #this is where your .py file, and can be change to test
```


```
#                    print(randomlist)
#                    print(ll)
#                    print(l)
                    a=ll[0]
                    b=ll[1]
                    c=ll[2]
                    d=ll[3]
                    file.write(f'{a} {b} {c} {d} 0' + '\n')
                    l.append(p)

# create the command to automatic run
    subprocess.getstatusoutput('cd Desktop/33')  #this is where your .py file, and can be change to test
```


```
    b= subprocess.getstatusoutput(f"~morri/bin/minisat {'file'+str(i)}.cnf")
# whether satisfiabl or not
    k = str(b[-1])
    k.split()
    print(k[-13:-1])
```

Question 4.

In this question, I would like to introduce a method for knowledge representation and reasoning which the Second-Order Logic .

(a).

Second-Order Logic

Firstly, the definition of Second-order logic is:

Second-order logic is base on the first-order logic and expand the syntax which means that it is extend the first-order logic by introducing quantification of predicate and functions variables of arity n (n>0).[1]

For example:

At first, suppose that x is a Human and x is a Man.

We can use First-order logic to express the knowledge that $\forall x[M(x) \to H(x)]$.

However, we just can express and study the individual variable x and the predict/function H and M, we cannot limit that or study.

Hence, to some extent, the Second-order Logic can solve this problem.

A simple example, x is a good Man and x is Responsible father which means that there are some properties such as a good man in the Man function/predict.

We can use Second-order Logic to express the knowledge:

$\forall M \exists g \forall x[M(g(x)) \to R(x)]$.

This is different to First-order logic because the First-order logic just can determine the man or woman, and the sub-properties or limit of Man predict cannot be expressed.

(b).

A simple knowledge base and sample: [2]

S = {Man(good), Man(rude), Woman(beautiful), Man(bad), good(Bob), rude(Jam), beautiful(Alice), Responsible father(Bob), Responsible mother(Alice)}

a = ∀M∃g∀x[M(g(x))->RF(x)].

Claim: S=>a,

Proof: let I be any interpretation and I |= S

Because of ∀M we have two situations in S.

Case 1: I |= Man(rude).

Due to the rude(Jam).
∴ I |= Man(rude(Jam)).

Because there is no RF(Jam).

∴ I |≠ RF(Jam).

∴ I |≠ a

Case 2: I |= Man(good).

Due to the good(Bob).
∴ I |= Man(good(Bob)).

And because Responsible father(Bob)

∴ Man(good(Bob)) -> RF(Bob).

∴ I |= a.

Either way, for any I, if I |= S then I |= α.
So S |= α. QED

(c).

The importance issues of Second-order Logic are that this method just like the First-order Logic. And if the sentences are really complex which means that the it is limited by the complex knowledge expressing. Hence, we need Higher-order Logic to solve that. In addition, there are also some methods to express knowledge and reasoning. Such as Frame representation or Semantic network representation。

There are two references:

1. Ketland, Jeffrey. "Second-Order Logic." Macmillan Reference USA, 2005.
2. Van Harmelen, Frank, Vladimir Lifschitz, and Bruce Porter, eds. *Handbook of knowledge representation*. P16-18, Vol. 1. Elsevier, 2008.