

COMP4418, 2018–Assignment 1

Student ID: Z5097690 Name: Baixiang Guan

Question 1.

(i) using truth table:

(a) **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 valid.

(b) **Yes** : $\models P \rightarrow (q \rightarrow P)$

P	q	$q \rightarrow P$	$P \rightarrow (q \rightarrow P)$
T	T	T	T
T	F	T	T
F	T	F	T
F	F	T	T

We can see from the truth table which shows
 the $P \rightarrow (q \rightarrow P)$ is always true no matter
 what P or q values.
 therefore, $P \rightarrow (q \rightarrow P)$ is a tautology.

(c) **No**

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

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

(d) **Yes**

P	q	$\neg p \rightarrow \neg q$	$\neg q \rightarrow \neg p$	$p \rightarrow q$	$q \rightarrow p$	$p \leftrightarrow q$
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 truth table,
 when $\neg p \rightarrow \neg q$ and $\neg q \rightarrow \neg p$ are true,
 then $p \leftrightarrow q$ is true. Therefore, $\neg p \rightarrow \neg q, \neg q \rightarrow \neg p$
 is valid. $p \leftrightarrow q$

(e) **Yes**

P	q	r	$p \rightarrow q$	$q \rightarrow r$	$\neg r \rightarrow \neg q$
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 $p \rightarrow q$ and $q \rightarrow r$ are true,
 $\neg r \rightarrow \neg q$ is also true. Hence,
 inference is valid.

And also can proof by $q \rightarrow r$.
 is equal to $\neg r \rightarrow \neg q$.
 So, no matter $p \rightarrow q$ is true or
 false.

(ii) Resolution

1. using resolution:

(a). $P \wedge (q \vee r) \vdash (P \wedge q) \vee (P \wedge r)$

Yes

$$\text{CNF } [P \wedge (q \vee r)] \equiv P \wedge (q \vee r)$$

$$\text{CNF } [(P \wedge q) \vee (P \wedge r)] \equiv P \wedge (q \vee r)$$

$$\text{CNF } [\neg(P \wedge (q \vee r))] \equiv \neg P \vee \neg(q \vee r)$$

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

$$\text{CNF } [\neg(P \rightarrow (q \rightarrow P))] \equiv P \wedge q \wedge \neg P$$

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

$$\text{CNF } [\neg(P \rightarrow q)] \equiv P \wedge \neg q$$

$$P \rightarrow q \equiv P \vee q$$

$$\text{CNF } [\neg(P \vee q)] \equiv \neg P \wedge \neg q$$

1. $P \vee \neg q$ [premise]

2. P [conclusion]

3. $\neg q$ [conclusion]

4. $P \vee \neg q$ [1, 2 resolution]

5. $\neg q$ [3, 4 resolution]

We can not
resolve to

\square , ~~not~~
~~is true or false~~
~~a part of q are~~
~~true~~

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

Yes

$$\neg P \rightarrow \neg q \equiv P \vee \neg q$$

$$\neg q \rightarrow \neg P \equiv q \vee \neg P$$

$$\text{CNF } [\neg(P \leftrightarrow q)] \equiv (P \vee q) \wedge (\neg q \vee \neg P)$$

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). $P \rightarrow q, q \rightarrow r \vdash \neg r \rightarrow \neg P$

Yes

$$P \rightarrow q \equiv \neg P \vee q$$

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

$$\text{CNF } [\neg(\neg r \rightarrow \neg P)] \equiv r \wedge \neg P$$

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 [3, 5 resolution]

Question 2.

- 2(a).
- "I never stole the jam!" pleaded the March Hare.
 $\neg \text{Stole}(\text{marchHare}, \text{jam})$
 - "No, no" pleaded the Hatter. "One of us stole it, but it wasn't me!"
 $\exists x [\text{stole}(x, \text{jam}) \wedge \neg \text{stole}(\text{madHatter}, \text{jam})]$
 - "At least one of them did" replied the Doormouse.
 $\exists x [(x = \text{marchHare} \vee x = \text{madHatter}) \rightarrow \neg (\text{ying}(x))]$
 - the March Hare and the Doormouse were not both speaking the truth.
 $\exists x [(x = \text{marchHare} \vee x = \text{doormouse}) \rightarrow (\text{ying}(x))]$

2(b):

1b. Yes, madHare stole the jam

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

Proof Semantically:

$$S = \{ \neg \text{stole}(\text{marchHare}, \text{jam}), \exists x [\text{stole}(x, \text{jam}) \wedge \neg \text{stole}(\text{madHatter}, \text{jam})] \rightarrow [(\text{stole}(\text{marchHare}, \text{jam}) \vee \text{stole}(\text{doormouse}, \text{jam})) \wedge \neg \text{stole}(\text{madHatter}, \text{jam})], \\ \exists x [(x = \text{marchHare} \vee x = \text{madHatter}) \rightarrow \neg (\text{ying}(x))] \rightarrow [\neg (\text{ying}(\text{marchHare})) \vee \neg (\text{ying}(\text{madHatter})) \vee \neg (\text{ying}(\text{doormouse}))], \\ \exists x [(x = \text{marchHare} \vee x = \text{doormouse}) \rightarrow (\text{ying}(x))] \rightarrow [(\text{ying}(\text{marchHare}) \vee \text{ying}(\text{doormouse})) \vee (\text{ying}(\text{marchHare}) \wedge \text{ying}(\text{doormouse}))]. \}$$

$$Q = \exists x [\text{stole}(x, \text{jam})]$$

claim: $S \models Q$

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

Case 1: $I \models \neg (\text{ying}(\text{marchHare}))$
 because $(\text{ying}(\text{marchHare}) \vee \text{ying}(\text{doormouse}))$ (from 4)
 $\therefore I \models (\text{ying}(\text{doormouse}))$
 because doormouse is ying.
 $\therefore \neg \exists x [(x = \text{marchHare} \vee x = \text{madHatter}) \rightarrow \neg (\text{ying}(x))] \equiv \forall x [(x = \text{marchHare} \vee x = \text{madHatter}) \wedge (\text{ying}(x))]$
~~which means that both marchHare and madHatter are ying.~~
 $\therefore I \models (\text{ying}(\text{marchHare}))$
 Hence, it is conflict with the assumption.

Case 2: $I \models \text{Isy}(\text{marchHare}) \wedge \text{Isy}(\text{doorMouse})$.

because $\text{Isy}(\text{marchHare})$

$\therefore \neg \text{Stole}(\text{marchHare}, \text{jam}) \equiv \text{Stole}(\text{marchHare}, \text{jam})$.

$I \models \perp$

but $I \models \text{Isy}(\text{doorMouse})$

which means that:

$I \models \text{Isy}(\text{marchHare}) \wedge \text{Isy}(\text{madHatter})$

$\therefore I \models \neg \exists x [\text{Stole}(x, \text{jam}) \wedge \neg \text{Stole}(\text{madHatter}, \text{jam})] \equiv \forall x [\neg \text{Stole}(x, \text{jam}) \vee \text{Stole}(\text{madHatter}, \text{jam})]$

* This is conflict with $\text{Stole}(\text{marchHare}, \text{jam})$ because there is only one stoler.

\therefore Case 2 can not be success.

Case 3: $I \models \text{Isy}(\text{marchHare}) \wedge \neg \text{Isy}(\text{doorMouse})$.

because $\text{Isy}(\text{marchHare})$

$\therefore \neg \text{Stole}(\text{marchHare}, \text{jam}) \equiv \text{Stole}(\text{marchHare}, \text{jam})$

$\therefore I \models \perp$

And

$I \models \neg \text{Isy}(\text{doorMouse})$

because marchHare is Isy and doorMouse is not Isy.

which means that $\neg \text{Isy}(\text{madHatter})$ must be true.

due to the $\exists x [(x = \text{marchHare} \vee x = \text{madHatter}) \rightarrow \neg \text{Isy}(x)]$.

$\therefore I \models \neg \text{Isy}(\text{madHatter})$.

\therefore madHatter is not Isy. then he is not stoler. and one stoler is marchHare because he Isy.

$\therefore I \models \perp$

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

So $S \models \perp$. QED.

PS: This proof just assume only one stoler to do, If it can be more than one stoler, then we just add a condition that limit number of stoler. After that, the proof 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.

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

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

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

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

which the formula is same with 4-SAT

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

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

When random generate number of variables and clauses.

```

1 import subprocess
2 import random
3
4 for i in range(20):
5     # generate 10 files
6     n = random.randint(4,100) # number of variables
7     m = random.randint(1,10) # n clauses
8     l=[]
9     for j in range(n):
10         l.append(j+1) #create variables
11
12 #print variabl and clauses number
13 print(n)
14 print(m)
15 with open(f'file{str(i)}.cnf','w') as file:
16     file.write(f'c example CNF file with {n} propositional variables and {m} c
17     file.write(f'p cnf {n} {m} \n')
18     for i in range(1,m+1):
19         p = random.randint(1,n) #choose a positive propositional variables
20         l.remove(p)
21         randomlist=[]
22         check_list=[]
23         a=1
24         while a==1:
25             randomlist = random.sample(l, 3) #choose three negative proposi
26             for i in range(len(randomlist)):
27                 randomlist[i]=randomlist[i]*-1
28             randomlist.append(p)
29             ll=sorted(randomlist)
30             if ll in check_list:
31                 # Prevent the same clause but e
32                 pass
33             else:
34                 f = ll
35                 check_list.append(ll)
36                 a = 0
37             print(randomlist)
38             print(ll)
39             print(i)
40             a=ll[0]
41
42 with open(f'file{str(i)}.cnf','w') as file:
43     file.write(f'c example CNF file with {n} propositional variables and {m} c
44     file.write(f'p cnf {n} {m} \n')
45     for i in range(1,m+1):
46         p = random.randint(1,n) #choose a positive propositional variables
47         l.remove(p)
48         randomlist=[]
49         check_list=[]
50         a=1
51         while a==1:
52             randomlist = random.sample(l, 3) #choose three negative proposi
53             for i in range(len(randomlist)):
54                 randomlist[i]=randomlist[i]*-1
55             randomlist.append(p)
56             ll=sorted(randomlist)
57             if ll in check_list:
58                 # Prevent the same clause but e
59                 pass
60             else:
61                 f = ll
62                 check_list.append(ll)
63                 a = 0
64             print(randomlist)
65             print(ll)
66             print(i)
67             a=ll[0]
68             b=ll[1]
69             c=ll[2]
70             d=ll[3]
71             file.write(f'{a} {b} {c} {d} 0' + '\n')
72             l.append(p)
73
74 # create the command to automatic run
75 subprocess.getstatusoutput('cd Desktop/33') #this is where your .py file, and
76 b= subprocess.getstatusoutput(f'-morri/bin/minisat {file+str(i)}.cnf')
77 # whether satisfiable or not
78 k = str(b[-1])
79 k.split()
80 print(k[-13:-1])

```

```

file.write(f'c example CNF filith {n} propositional variables and {n} c')
file.write(f'p cnf {n} {n}*'\n')
for i in range(1,a+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 negative propositions
        for i in range(len(randomlist)):
            randomlist[i]=randomlist[i]*-1
        randomlist.append(p)
        ll=sorted(randomlist)
        if ll in check_list:
            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
b= subprocess.getstatusoutput(f'-morri/bin/minisat {file}'+str(i)).cnf')
# whether satisfiable or not
k = str(b[-1])
k.splitt()
print(k[-13:-1])

```

When $n < C$:

```

1  #!/usr/bin/env python3
2  #- coding: utf-8 -*-
3
4  Created on Fri Aug 24 20:27:57 2018
5
6  @author: seele
7  """
8
9  import subprocess
10 import random
11
12 for i in range(20):
13     # generate 10 files
14     n = 4*random.randint(4,10) # number of variables
15     m = 20*random.randint(1,10) # # clauses
16     l=[]
17     for j in range(n):
18         l.append(j+1)
19         #create variables
20
21     #print variabel and clauses number
22     print(n)
23     with open(f'file'+str(l).conf+'.w') as file:
24         file.write(f'c example CNF file with {n} propositional variables and {m} c\n')
25         file.write(f'p cnf {n} {m} 1\n')
26         for i in range(1,m+1):
27             p = random.randint(1,n) #choose a positive propositional variables
28             l.remove(p)
29             randomlist=[]
30             check_list=[]
31             a=1
32             while a==1:
33                 randomlist = random.sample(l, 3)
34                 for i in range(len(randomlist)):
35                     randomlist[i]=randomlist[i]-1
36                 randomlist.append(p)
37                 l=sorted(randomlist)
38
39 In [78]: runfile('/import/glass/3/5097690/Desktop/33/temp.py', wdir='/
40 /import/glass/3/5097690/Desktop/33')
41 4
42 20
43
44 SATISFIABL
45 4
46 20
47
48 SATISFIABL
49 4
50 20
51
52 SATISFIABL
53 4
54 20
55
56 SATISFIABL
57 4
58 20
59
60 SATISFIABL
61 4
62 20
63
64 SATISFIABL
65 4
66 20
67
68 SATISFIABL
69 4
70 20
71
72 SATISFIABL
73 4
74 20
75
76 SATISFIABL
77 4
78 20
79
80 SATISFIABL
81 4
82 20
83
84 SATISFIABL
85 4
86 20
87
88 SATISFIABL
89 4
90 20
91
92 SATISFIABL
93 4
94 20
95
96 SATISFIABL
97 4
98 20
99
100 SATISFIABL
101 4
102 20
103
104 SATISFIABL
105 4
106 20
107
108 SATISFIABL
109 4
110 20
111
112 SATISFIABL
113 4
114 20
115
116 SATISFIABL
117 4
118 20
119
120 SATISFIABL
121 4
122 20
123
124 SATISFIABL
125 4
126 20
127
128 SATISFIABL
129 4
130 20
131
132 SATISFIABL
133 4
134 20
135
136 SATISFIABL
137 4
138 20
139
140 SATISFIABL
141 4
142 20
143
144 SATISFIABL
145 4
146 20
147
148 SATISFIABL
149 4
150 20
151
152 SATISFIABL
153 4
154 20
155
156 SATISFIABL
157 4
158 20
159
160 SATISFIABL
161 4
162 20
163
164 SATISFIABL
165 4
166 20
167
168 SATISFIABL
169 4
170 20
171
172 SATISFIABL
173 4
174 20
175
176 SATISFIABL
177 4
178 20
179
180 SATISFIABL
181 4
182 20
183
184 SATISFIABL
185 4
186 20
187
188 SATISFIABL
189 4
190 20
191
192 SATISFIABL
193 4
194 20
195
196 SATISFIABL
197 4
198 20
199
200 SATISFIABL
201 4
202 20
203
204 SATISFIABL
205 4
206 20
207
208 SATISFIABL
209 4
210 20
211
212 SATISFIABL
213 4
214 20
215
216 SATISFIABL
217 4
218 20
219
220 SATISFIABL
221 4
222 20
223
224 SATISFIABL
225 4
226 20
227
228 SATISFIABL
229 4
230 20
231
232 SATISFIABL
233 4
234 20
235
236 SATISFIABL
237 4
238 20
239
240 SATISFIABL
241 4
242 20
243
244 SATISFIABL
245 4
246 20
247
248 SATISFIABL
249 4
250 20
251
252 SATISFIABL
253 4
254 20
255
256 SATISFIABL
257 4
258 20
259
260 SATISFIABL
261 4
262 20
263
264 SATISFIABL
265 4
266 20
267
268 SATISFIABL
269 4
270 20
271
272 SATISFIABL
273 4
274 20
275
276 SATISFIABL
277 4
278 20
279
280 SATISFIABL
281 4
282 20
283
284 SATISFIABL
285 4
286 20
287
288 SATISFIABL
289 4
290 20
291
292 SATISFIABL
293 4
294 20
295
296 SATISFIABL
297 4
298 20
299
300 SATISFIABL
301 4
302 20
303
304 SATISFIABL
305 4
306 20
307
308 SATISFIABL
309 4
310 20
311
312 SATISFIABL
313 4
314 20
315
316 SATISFIABL
317 4
318 20
319
320 SATISFIABL
321 4
322 20
323
324 SATISFIABL
325 4
326 20
327
328 SATISFIABL
329 4
330 20
331
332 SATISFIABL
333 4
334 20
335
336 SATISFIABL
337 4
338 20
339
340 SATISFIABL
341 4
342 20
343
344 SATISFIABL
345 4
346 20
347
348 SATISFIABL
349 4
350 20
351
352 SATISFIABL
353 4
354 20
355
356 SATISFIABL
357 4
358 20
359
360 SATISFIABL
361 4
362 20
363
364 SATISFIABL
365 4
366 20
367
368 SATISFIABL
369 4
370 20
371
372 SATISFIABL
373 4
374 20
375
376 SATISFIABL
377 4
378 20
379
380 SATISFIABL
381 4
382 20
383
384 SATISFIABL
385 4
386 20
387
388 SATISFIABL
389 4
390 20
391
392 SATISFIABL
393 4
394 20
395
396 SATISFIABL
397 4
398 20
399
400 SATISFIABL
401 4
402 20
403
404 SATISFIABL
405 4
406 20
407
408 SATISFIABL
409 4
410 20
411
412 SATISFIABL
413 4
414 20
415
416 SATISFIABL
417 4
418 20
419
420 SATISFIABL
421 4
422 20
423
424 SATISFIABL
425 4
426 20
427
428 SATISFIABL
429 4
430 20
431
432 SATISFIABL
433 4
434 20
435
436 SATISFIABL
437 4
438 20
439
440 SATISFIABL
441 4
442 20
443
444 SATISFIABL
445 4
446 20
447
448 SATISFIABL
449 4
450 20
451
452 SATISFIABL
453 4
454 20
455
456 SATISFIABL
457 4
458 20
459
460 SATISFIABL
461 4
462 20
463
464 SATISFIABL
465 4
466 20
467
468 SATISFIABL
469 4
470 20
471
472 SATISFIABL
473 4
474 20
475
476 SATISFIABL
477 4
478 20
479
480 SATISFIABL
481 4
482 20
483
484 SATISFIABL
485 4
486 20
487
488 SATISFIABL
489 4
490 20
491
492 SATISFIABL
493 4
494 20
495
496 SATISFIABL
497 4
498 20
499
500 SATISFIABL
501 4
502 20
503
504 SATISFIABL
505 4
506 20
507
508 SATISFIABL
509 4
510 20
511
512 SATISFIABL
513 4
514 20
515
516 SATISFIABL
517 4
518 20
519
520 SATISFIABL
521 4
522 20
523
524 SATISFIABL
525 4
526 20
527
528 SATISFIABL
529 4
530 20
531
532 SATISFIABL
533 4
534 20
535
536 SATISFIABL
537 4
538 20
539
540 SATISFIABL
541 4
542 20
543
544 SATISFIABL
545 4
546 20
547
548 SATISFIABL
549 4
550 20
551
552 SATISFIABL
553 4
554 20
555
556 SATISFIABL
557 4
558 20
559
560 SATISFIABL
561 4
562 20
563
564 SATISFIABL
565 4
566 20
567
568 SATISFIABL
569 4
570 20
571
572 SATISFIABL
573 4
574 20
575
576 SATISFIABL
577 4
578 20
579
580 SATISFIABL
581 4
582 20
583
584 SATISFIABL
585 4
586 20
587
588 SATISFIABL
589 4
590 20
591
592 SATISFIABL
593 4
594 20
595
596 SATISFIABL
597 4
598 20
599
600 SATISFIABL
601 4
602 20
603
604 SATISFIABL
605 4
606 20
607
608 SATISFIABL
609 4
610 20
611
612 SATISFIABL
613 4
614 20
615
616 SATISFIABL
617 4
618 20
619
620 SATISFIABL
621 4
622 20
623
624 SATISFIABL
625 4
626 20
627
628 SATISFIABL
629 4
630 20
631
632 SATISFIABL
633 4
```

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 variables
    p = 20*random.randint(1,10) # n clauses
    l=[]
    for j in range(n):
        l.append(j+1)

    #create variables

    print(n)
    print(l)
    with open(f'file{str(i)}.cnf','w') as file:
        file.write(f'c example CNF file with {n} propositional variables and {m} c\n')
        file.write(f'p cnf {n} {m} {n} \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 negative propositions
                for i in range(len(randomlist)):
                    randomlist[i]=random.randint(1,i+1)
                randomlist.append(p)
                a=0
```

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
    s = 10*random.randint(1,10) # number of variables
    m = 20*random.randint(1,10) # m clauses
    l=[]
    for j in range(n):
        l.append(j+1)
        #create variables

print(n)
#Print variabel and clauses number
print(m)
with open('file'+str(i).conf+'.w') as file:
    file.write('c example CNF file with (n) propositional variables and (m) c\n')
    for i in range(1,m+1):
        p = random.randint(1,n) #choose a positive propositional variables
        l.remove(p)
        randomlist=[]
        check_list=[]
        s+=1
        while s==1:
            randomlist = random.sample(l, 3)
            for i in range(1 len(randomlist)):
                randomlist[i]=randomlist[i]-1
            randomlist.append(p)
            s=1
```


Explanation: We can see that whatever the number of propositional variables n is smaller or equal or larger than the number of clauses C , the outcome is always satisfiable. 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 satisfy the clauses.

Because of that the Horn clauses have two form, positive clause. like $[\neg p_1, \neg p_2, \dots, q]$ negative clause: $[\neg p_1, \dots, \neg p_n]$. And in the programming it just generate positive clause, because the negative clause can easily be repaired. So, we 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 variable P_i , we can find the $\neg p_i$ in other clauses and use the resolution to resolve these two clauses, to generate a new clause. for the new clauses, we also do this work repeatedly until the last clause is left or there is no clause to resolution, the process is like:

because of that we can not resolve the left clauses which means there is no conflict between the clauses. So, just assign each positive variable is true and negative variable is false then it is always satisfiable.

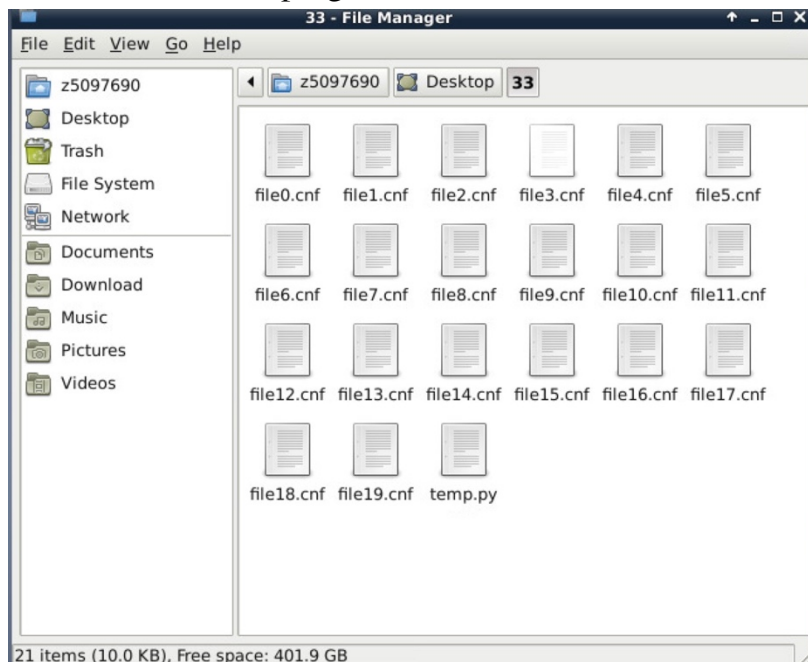


2. If all clauses is negative clauses, which means in each clause, there is no positive variable. Then we just ^{assign} all variable to be False. then the whole clause is true, because there is no conflict between clauses. Hence, it is always satisfiable.
3. If the clauses contain positive and negative clause, it is same with 1 or 2. at the end of resolution. Therefore, it is satisfiable.

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: ~~one~~ conflict at the end, the main reason is each clause must have 4 variables and at most one positive variable in the clause. So, the base case is after resolution. There are still ~~left~~ 2 variables 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 got \square at the end which means that ^{there} is always a method to satisfy.

This is test file and program screenshot:



The code is:

```
ASST-Q3.py
1 #code by python 3.6
2
3 import subprocess
4 import random
5
6 for i in range(20): # generate 10 files
7     n = random.randint(4,10) # number of varibals
8     m = random.randint(1,10) # m clauses
9     l=[]
10    for j in range(n):
11        l.append(j+1) #create varibals
12
13
14    print(n) #print varibal and clauses number
15    print(m)
16    with open(f'file{str(i)}.cnf','w') as file:
17        file.write(f'c example CNF file with {n} propositional variables and {m} clauses'+'\n')
18        file.write(f'p cnf {n} {m}'+'\n')
19        for i in range(1,m+1):
20            p = random.randint(1,n) #choose a positive propositional variables
21            l.remove(p)
22            randomlist=[]
23            check_list=[]
24            a=1
25            while a==1:
26                randomlist = random.sample(l, 3) #choose three negative propositional variables
27                for i in range(len(randomlist)):
28                    randomlist[i]=randomlist[i]*-1
29                randomlist.append(p)
30                ll=sorted(randomlist)
31                if ll in check_list: # Prevent the same clause but even same is ok
32                    pass
33                else:
34                    f = ll
35                    check_list.append(ll)
36                    a= 0
37                    print(randomlist)
38                    print(ll)
39                    print(l)
40                    a=ll[0]
41                    b=ll[1]
42                    c=ll[2]
43                    d=ll[3]
44                    file.write(f'{a} {b} {c} {d} 0'+ '\n')
45                    l.append(p)
46
47 # create the command to automatic run
48 subprocess.getstatusoutput('cd Desktop/33') #this is where your .py file, and can be change to test
49 b=subprocess.getstatusoutput(f'~morri/bin/minisat {f'file{str(i)}.cnf'})
50
51
52
53
54
55
56
57 # print(randomlist)
58 # print(ll)
59 # print(l)
60 # a=ll[0]
61 # b=ll[1]
62 # c=ll[2]
63 # d=ll[3]
64 # file.write(f'{a} {b} {c} {d} 0'+ '\n')
65 # l.append(p)
66
67 # create the command to automatic run
68 subprocess.getstatusoutput('cd Desktop/33') #this is where your .py file, and can be change to test
69 b=subprocess.getstatusoutput(f'~morri/bin/minisat {f'file{str(i)}.cnf'})
70 # whether satisfiabl or not
71 k = str(b[-1])
72 k.split()
73 print(k[-13:-1])
74
```


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 $\forall x[M(x) \rightarrow H(x)]$.

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:

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

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]

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

$a = \forall M \exists g \forall x [M(g(x)) \rightarrow \text{RF}(x)]$.

Claim: $S \Rightarrow a$,

Proof: let I be any interpretation and $I \models S$

Because of $\forall M$ we have two situations in S .

Case 1: $I \models \text{Man}(\text{rude})$.

Due to the $\text{rude}(\text{Jam})$.

$\therefore I \models \text{Man}(\text{rude}(\text{Jam}))$.

Because there is no $\text{RF}(\text{Jam})$.

$\therefore I \not\models \text{RF}(\text{Jam})$.

$\therefore I \not\models a$

Case 2: $I \models \text{Man}(\text{good})$.

Due to the $\text{good}(\text{Bob})$.

$\therefore I \models \text{Man}(\text{good}(\text{Bob}))$.

And because $\text{Responsible father}(\text{Bob})$

$\therefore \text{Man}(\text{good}(\text{Bob})) \rightarrow \text{RF}(\text{Bob})$.

$\therefore I \models a$.

Either way, for any I , if $I \models S$ then $I \models a$.

So $S \models a$. 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.