

COMP4418, 2018 – Assignment 2

ZID: 5097690

Name: Baixiang Guan

Question 1:

(a).

The program of ASP that decides the k-Clique problem is shown at below:

We can identify that is there any k-Clique by giving the constant k such that $k=2$.

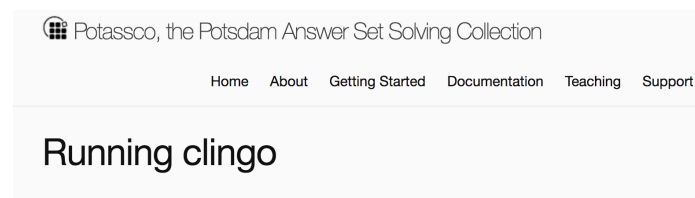
```
1 % Question 1
2 %decide if there is a k-clique. such that whether there is a 3-clique.
3 #const k=2.
4
5 % Vertices all vertices from 1 to 6
6 v(1).v(2).v(3).v(4).v(5).v(6).
7
8 % Edges all eages and two direction
9 e(1,2).e(2,1).e(1,3).e(3,1).e(1,4).e(4,1).e(2,4).e(4,2).
10 e(2,5).e(5,2).e(2,6).e(6,2).e(3,4).e(4,3).e(3,5).e(5,3).
11 e(3,6).e(6,3).e(4,5).e(5,4).e(5,6).e(6,5).
12
13 % encoding let c(X) is at least k V(X) and at most k v(X)
14 % and if there is no edges between c(X),C(Y) then C is not clique.
15 k{c(X):v(X)}k.
16 :- not e(X, Y), c(X), c(Y), X != Y.
17
18 % Display
19 #show c/1.
```

(b).

Using the running clingo at online, like the picture at below, when $k \in \{3, 4, 5, 6\}$.

The result are 6 when $k=3$ and 0 when $k=4$ and 0 when $k=5$ and 0 when $k=6$.

The picture show the result at below.



When $k=2$:

```
clingo version 5.3.0
Reading from stdin
Solving...
Answer: 1
c(5) c(6)
Answer: 2
c(5) c(4)
Answer: 3
c(1) c(4)
Answer: 4
c(5) c(2)
Answer: 5
c(6) c(2)
Answer: 6
c(1) c(2)
Answer: 7
c(2) c(4)
Answer: 8
c(5) c(3)
Answer: 9
c(6) c(3)
Answer: 10
c(1) c(3)
Answer: 11
c(3) c(4)
SATISFIABLE

Models      : 11
Calls       : 1
Time        : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

When $k=3$:

```
clingo version 5.3.0
Reading from stdin
Solving...
Answer: 1
c(5) c(2) c(4)
Answer: 2
c(5) c(3) c(4)
Answer: 3
c(5) c(6) c(2)
Answer: 4
c(5) c(6) c(3)
Answer: 5
c(1) c(2) c(4)
Answer: 6
c(1) c(3) c(4)
SATISFIABLE

Models      : 6
Calls       : 1
Time        : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

When $k=4$:

```
clingo version 5.3.0
Reading from stdin
Solving...
UNSATISFIABLE

Models      : 0
Calls       : 1
Time        : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

When $k=5$:

```
clingo version 5.3.0
Reading from stdin
Solving...
UNSATISFIABLE

Models      : 0
Calls       : 1
Time        : 0.003s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

When $k=6$:

```
clingo version 5.3.0
Reading from stdin
Solving...
UNSATISFIABLE

Models      : 0
Calls       : 1
Time        : 0.003s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

Question 2:

There are 3 outputs by using the Clingo online which is shown at below:

```

1 % Question 2
2
3 % encoding
4 a :- not b, not c.
5 b :- not a, not c.
6 c :- not b, not a.
7 d :- a.
8 d :- b.
9 d :- c.
10
11
12

```

Configuration: reasoning mode enumerate all ☐ project ☐ statistics

[Run!](#)

```

clingo version 5.3.0
Reading from stdin
Solving...
Answer: 1
c d
Answer: 2
b d
Answer: 3
a d
SATISFIABLE

Models      : 3
Calls       : 1
Time        : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s

```

And the table is :

S	Reduct P^S	Stable model?
{a,b,c,d}	d←a. d←b. d←c.	✗
{a, b, c}	d←a. d←b. d←c.	✗
{a, b, d}	d←a. d←b. d←c.	✗
{a, c, d}	d←a. d←b. d←c.	✗
{b, c, d}	d←a. d←b. d←c.	✗
{a, b}	d←a. d←b. d←c.	✗
{a, c}	d←a. d←b. d←c.	✗
{a, d}	a. d←a. d←b. d←c.	✓
{b, c}	d←a. d←b. d←c.	✗
{b, d}	b. d←a. d←b. d←c.	✓
{c, d}	c. d←a. d←b. d←c.	✓
{a}	a. d←a. d←b. d←c.	✗
{b}	b. d←a. d←b. d←c.	✗
{c}	c. d←a. d←b. d←c.	✗
{d}	a. b. c. d←a. d←b. d←c.	✗
{ }	a. b. c. d←a. d←b. d←c.	✗

Question 3:

(a).

The run time for the sat-naive is more than 5 minutes.

```
xterm
z5097690@tabla01:~$ cd Desktop
z5097690@tabla01:~/Desktop$ c++ -std=c++11 -O3 -DINDEBUG -o sat-naive sat-naive.c
c++: error: unrecognized command line option '-O3'
z5097690@tabla01:~/Desktop$ c++ -std=c++11 -O3 -DINDEBUG -o sat-naive sat-naive.c
c++: error: sat-ip.cc: No such file or directory
c++: fatal error: no input files
compilation terminated.
z5097690@tabla01:~/Desktop$ c++ -std=c++11 -O3 -DINDEBUG -o sat-up sat-up.cc
z5097690@tabla01:~/Desktop$ c++ -std=c++11 -O3 -DINDEBUG -o sat-cdcl sat-cdcl.cc
z5097690@tabla01:~/Desktop$
z5097690@tabla01:~/Desktop$ ns exercise2.tcl
z5097690@tabla01:~/Desktop$ ns exercise2.tcl
z5097690@tabla01:~/Desktop$ ./sat-naive sudoku.cnf
bash: ./sat-naive: No such file or directory
z5097690@tabla01:~/Desktop$ ./sat-naive sudoku.cnf
```

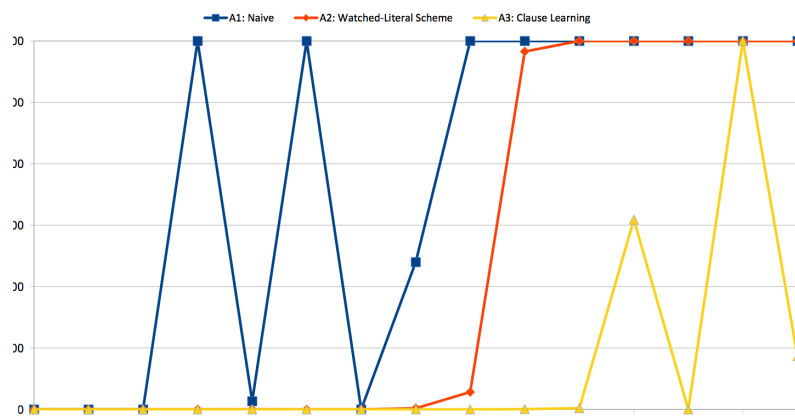
The run time for the sat-up is 0.006220s.

```
z5097690@tabla01:~/Desktop$ ./sat-up sudoku.cnf
SATISFIABLE (in 0.006220 s, sat-up)
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 18 19 -20 -21 -22 -23
-24 -25 -26 -27 -28 29 -30 -31 -32 -33 -34 -35 -36 -37 -38 39 -40 -41 -42 -43 -4
4 -45 -46 -47 -48 -49 -50 -51 -52 53 -54 -55 -56 -57 -58 59 -60 -61 -62 -63 -64
-65 -66 -67 -68 69 -70 -71 -72 -73 -74 -75 -76 -77 -78 79 -80 -81 -82 -83 84 -85
-86 -87 -88 -89 -90 -91 92 -93 -94 -95 -96 -97 -98 -99 -100 -101 -102 -103 104
-105 -106 -107 -108 -109 -110 -111 -112 -113 -114 -115 -116 117 -118 -119 -120 -
121 -122 123 -124 -125 -126 -127 -128 -129 -130 -131 -132 133 -134 -135 -136 -13
7 -138 -139 -140 -141 -142 143 -144 -145 -146 -147 148 -149 -150 -151 -152 -153
154 -155 -156 -157 -158 -159 -160 -161 -162 -163 -164 -165 -166 -167 168 -169 -1
70 -171 -172 -173 -174 -175 -176 -177 178 -179 -180 -181 -182 -183 -184 -185 -18
6 -187 188 -189 190 -191 -192 -193 -194 -195 -196 -197 -198 -199 -200 -201 -202
-203 -204 -205 -206 -207 -208 -209 -210 -211 -212 -213 -214 -215 -216 -217 -218 -2
19 -220 -221 -222 -223 -224 -225 -226 -227 228 -229 -230 -231 -232 -233 -234 -235
-236 -237 -238 -239 -240 -241 -242 -243 244 -245 -246 -247 -248 -249 -250 -251 -
252 -253 -254 -255 -256 257 -258 -259 -260 -261 -262 -263 -264 -265 -266 267 -26
8 -269 -270 -271 -272 273 -274 -275 -276 -277 -278 -279 -280 281 -282 -283 -284
-285 -286 -287 -288 -289 -290 -291 -292 -293 -294 -295 -296 297 -298 -299 -300 -
301 -302 -303 304 -305 -306 -307 -308 -309 -310 -311 -312 -313 314 -315 -316 -31
7 -318 319 -320 -321 -322 -323 -324 -325 -326 -327 -328 -329 -330 -331 -332 -333
-334 -335 -336 337 -338 -339 -340 -341 -342 -343 -344 -345 -346 -347 -348 349 -3
50 -351 -352 -353 -354 -355 -356 357 -358 -359 -360 -361 -362 -363 -364 -365 -36
6 -367 368 -369 -370 -371 -372 -373 374 -375 -376 -377 -378 379 -380 -381 -382 -
383 -384 -385 -386 -387 -388 -389 -390 -391 -392 -393 -394 -395 396 -397 -398 39
9 -400 -401 -402 -403 -404 -405 -406 -407 -408 -409 -410 -411 -412 413 -414 -415
-416 417 -418 -419 -420 -421 -422 -423 -424 -425 -426 -427 -428 -429 -430 -431
432 -433 -434 -435 436 -437 -438 -439 -440 -441 -442 -443 -444 -445 -446 -447 44
8 -449 -450 451 -452 -453 -454 -455 -456 -457 -458 -459 460 -461 -462 -463 -464
-465 -466 -467 -468 -469 -470 -471 -472 473 -474 -475 -476 -477 -478 -479 -480 -
481 -482 483 -484 -485 -486 -487 -488 -489 -490 -491 -492 493 -494 -495 -496 -49
7 -498 -499 -500 -501 -502 -503 -504 -505 -506 -507 508 -509 -510 -511 -512 513
-514 -515 -516 -517 518 -519 -520 -521 -522 -523 -524 -525 -526 -527 -528 -529 -
530 -531 -532 533 -534 -535 -536 -537 -538 -539 -540 -541 -542 543 -544 -545 -546
-547 -548 -549 550 -551 -552 -553 -554 -555 -556 -557 -558 -559 -560 -561 -562
-563 -564 -565 566 -567 -568 -569 -570 -571 -572 -573 -574 -575 576 -577 -578 57
9 -580 -581 -582 -583 -584 -585 -586 -587 -588 -589 -590 -591 -592 -593 -594 -595
-596 -597 -598 -599 -600 -601 602 -603 -604 -605 -606 607 -608 -609 -610 -611 -
612 -613 -614 615 -616 -617 -618 -619 -620 -621 -622 -623 -624 -625 -626 627 -62
8 -629 -630 -631 -632 -633 -634 -635 -636 637 -638 -639 -640 -641 -642 -643 644
-645 -646 -647 -648 -649 -650 -651 -652 653 -654 -655 -656 -657 -658 -659 -660 -
661 -662 -663 -664 665 -666 -667 -668 669 -670 -671 -672 -673 -674 -675 -676 -677
-678 -679 -680 -681 682 -683 -684 685 -686 -687 -688 -689 -690 -691 -692 -693
-694 -695 -696 -697 -698 699 -700 -701 -702 -703 -704 -705 706 -707 -708 -709 -7
10 -711 -712 713 -714 -715 -716 -717 -718 -719 -720 -721 -722 -723 -724 -725 -72
6 -727 -728 729 0
z5097690@tabla01:~/Desktop$
```

The run time for the sat-up is 0.005160s.

```
z5097690@tabla01:~/Desktop$ ./sat-cdcl sudoku.cnf
SATISFIABLE (in 0.005160 s, sat-cdcl)
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 18 19 -20 -21 -22 -23 -24 -25 -26 -27 -28 29 -30 -
31 -32 -33 -34 -35 -36 -37 -38 39 -40 -41 -42 -43 -44 -45 -46 -47 -48 -49 -50 -51 -52 53 -54 -55 -56 -57 -58
59 -60 -61 -62 -63 -64 -65 -66 -67 -68 69 -70 -71 -72 -73 -74 -75 -76 -77 -78 79 -80 -81 -82 -83 84 -85 -86
-87 -88 -89 -90 -91 92 -93 -94 -95 -96 -97 -98 -99 -100 -101 -102 -103 104 -105 -106 -107 -108 -109 -110 -1
11 -112 -113 -114 -115 -116 117 -118 -119 -120 -121 -122 123 -124 -125 -126 -127 -128 -129 -130 -131 -132 13
3 -134 -135 -136 -137 -138 -139 -140 -141 -142 143 -144 -145 -146 -147 148 -149 -150 -151 -152 -153 154 -155
-156 -157 -158 -159 -160 -161 -162 -163 -164 -165 -166 -167 168 -169 -170 -171 -172 -173 -174 -175 -176 -17
7 -178 -179 -180 -181 -182 -183 -184 -185 -186 -187 188 -189 190 -191 -192 -193 -194 -195 -196 -197 -198 -199
-200 -201 -202 203 -204 -205 -206 -207 -208 -209 -210 -211 -212 -213 -214 -215 -216 -217 -218 -219 -220 -221
-222 -223 -224 -225 -226 -227 228 -229 -230 -231 -232 -233 -234 -235 -236 -237 -238 -239 -240 -241 -242 -243
-244 -245 -246 -247 -248 -249 -250 -251 -252 -253 -254 -255 -256 257 -258 -259 -260 -261 -262 -263 -264 -265
-266 267 -268 -269 -270 -271 -272 273 -274 -275 -276 -277 -278 -279 -280 281 -282 -283 -284 -285 -286 -287 -
288 -289 -290 -291 -292 -293 -294 -295 -296 297 -298 -299 -300 -301 -302 -303 304 -305 -306 -307 -308 -309 -
310 -311 -312 -313 314 -315 -316 -317 -318 319 -320 -321 -322 -323 -324 -325 -326 -327 -328 -329 -330 -331 -3
32 -333 -334 -335 -336 337 -338 -339 -340 -341 -342 -343 -344 -345 -346 -347 -348 349 -350 -351 -352 -353 -3
54 -355 -356 357 -358 -359 -360 -361 -362 -363 -364 -365 -366 -367 368 -369 -370 -371 -372 -373 374 -375 -37
6 -377 -378 379 -380 -381 -382 -383 -384 -385 -386 -387 -388 -389 -390 -391 -392 -393 -394 -395 396 -397 -39
8 -399 -400 -401 -402 -403 -404 -405 -406 -407 -408 -409 -410 -411 -412 413 -414 -415 -416 417 -418 -419 -420
-421 -422 -423 -424 -425 -426 -427 -428 -429 -430 -431 -432 -433 -434 -435 -436 -437 -438 -439 -440 -441 -442
-443 -444 -445 -446 -447 448 -449 -450 451 -452 -453 -454 -455 -456 -457 -458 -459 -460 461 -462 -463 -464
-465 -466 -467 -468 -469 -470 -471 -472 473 -474 -475 -476 -477 -478 -479 -480 -481 -482 483 -484 -485 -486
-487 -488 -489 -490 -491 -492 493 -494 -495 -496 -497 -498 -499 -500 501 -502 -503 -504 -505 -506 -507 508 -
509 -510 -511 -512 -513 -514 -515 -516 -517 518 -519 -520 -521 -522 -523 -524 -525 -526 -527 -528 -529 -530
531 -532 533 -534 -535 -536 -537 -538 -539 -540 -541 -542 543 -544 -545 -546 -547 -548 -549 550 -551 -552 -5
53 -554 -555 -556 -557 -558 -559 -560 -561 -562 -563 -564 -565 -566 -567 -568 -569 -570 -571 -572 -573 -574 -
575 576 577 -578 -579 -580 -581 -582 -583 -584 -585 -586 -587 -588 -589 -590 -591 -592 -593 -594 -595 -596 -5
97 -598 -599 -600 -601 602 -603 -604 -605 -606 607 -608 -609 -610 -611 -612 -613 -614 615 -616 -617 -618 -61
9 -620 -621 -622 -623 -624 -625 -626 627 -628 -629 -630 -631 -632 -633 -634 -635 -636 637 -638 -639 -640 -64
1 -642 -643 644 -645 -646 -647 -648 -649 -650 -651 -652 653 -654 -655 -656 -657 -658 -659 -660 -661 -662 -66
3 -664 665 -666 -667 -668 669 -670 -671 -672 -673 -674 -675 -676 -677 -678 -679 -680 -681 682 -683 -684 685
-686 -687 -688 -689 -690 -691 -692 -693 -694 -695 -696 -697 -698 699 -700 -701 -702 -703 -704 -705 706 -707
-708 -709 -710 -711 -712 713 -714 -715 -716 -717 -718 -719 -720 -721 -722 -723 -724 -725 -726 -727 -728 729
0
z5097690@tabla01:~/Desktop$
```

(b).



We can see the picture at above which from the lecture that the naive is the lowest efficiency to solve the problem and the sat-cdcl is the best method in these three methods.

The main reason of different run times is that these three methods are differently.

For the sat-naive, it just tries every answer to solve the problem which means that the search space is exponential in number of variables.

And for the sat-up, it uses watched-literal scheme method to solve the problem which is a lazy data structure for fast unit propagation and very cheap backtracking. Hence, it will be more efficiency than the naive.

And the last method sat-cdcl is using Clause learning method that is based on the watched-literal scheme bur learn from the conflict to avoid similar mistakes later. This, in turn, will better than the watched-literal scheme method if there are lots of variables and clauses.

Question 4:

(a).

Yes, converting a propositional formula into an equisatisfiable CNF formula in the worst case requires exponential time (under the assumption $P \neq NP$).

We can use a example to illustrate that.

For example, $\alpha \vee \beta$ is converted to CNF as follows:

- 1.If α and β are literals which means that it is the atomic element, then $\alpha \vee \beta$ is already the CNF such that $(\alpha \vee \beta) \wedge \text{others}$.
2. If α not literals, then $\alpha = \alpha_1 \wedge \dots \wedge \alpha_k$ and $k \geq 2$, then $\alpha \vee \beta = (\alpha_1 \vee \beta) \wedge \dots \wedge (\alpha_k \vee \beta)$ and if β is literal then $(\alpha_1 \vee \beta) \wedge \dots \wedge (\alpha_k \vee \beta)$ is already the CNF, otherwise, $\beta = \beta_1 \wedge \dots \wedge \beta_k$ and $k \geq 2$. And we need to take one more distribution step to convert each $\alpha_i \wedge \beta_i$ to CNF. Hence, for each clause, if it is not the literal, then we need convert it at least 2 literals. This means that for the worst case of a propositional formula, converting it to CNF can produce a formula of size 2^n which means that it will requires exponential time under the assumption $P \neq NP$.

(b).

In my opinion, it is **not true** of that there are decision problems that cannot be reduced to SAT (if so, name a concrete problem).

The main reason is that the decision problems always have the answer of yes or no. This implies that is a propositional formula satisfiable? So, for each decision problems, we can decompose that to the clauses and the clauses can also be decomposed and converting the whole propositional to the CNF formula. So, based on the sat definition, the decision problems can be reduced to SAT problems. And lots of NP problems can reduced to the SAT problems at the moment which means if we solve the all SAT problems we may solve the NP problems.

(c).

If I is closed under unit propagation relative to ϕ , then in order to close $I \cup \{x\}$ under unit propagation relative to ϕ it suffices to inspect the clauses $c \in \phi$ that watch $\neg x$.

Yes. This can be explained from the lecture example:

Definition: closure of I under unit propagation relative to ϕ

- Let $I^0 = I$
- Repeat for $j > 0$ until $I^j = I^{j+1}$:
 - ▶ If there is a $(x_1 \vee \dots \vee x_k) \in \phi$ with $\bar{x}_1, \dots, \bar{x}_k \in I^j$:
Return **conflict** $(x_1 \vee \dots \vee x_k)$
 - ▶ If there is a $(x_1 \vee \dots \vee x_{k+1}) \in \phi$ with $\bar{x}_1, \dots, \bar{x}_k \in I^j$:
Let $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return I^j

Ex. 1: $I = \{\neg p\}$ $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q \vee r), (p \vee q)\}$

- $I^0 = \{\neg p\}$
- $I^1 = \{\neg p, q\}$
- $I^2 = \{\neg p, q, r\}$

We can see from the picture at above, the I^2 is closed under unit propagation relative to ϕ because $I^2 = I^3$. And if we add the $\{\neg s\}$ in the I , which can be describe by $I = I \cup \{\neg s\} = \{\neg p, q, r, \neg s\}$, then we just inspect the clauses $c \in \phi$ which include the s . Because $I = \{\neg p, q, r\}$ is already closed under unit propagation relative to ϕ . So, we just determine the new element that does it conflict the clause which include the s or if there is a $(x_1 \vee \dots \vee x_{k+1}) \in \phi$ with $x_1, \dots, x_k \in I^j$ and repeat this process.

Hence, for every time of we add a new element x in the I , we just inspect the clauses $c \in \phi$ which contain $\neg x$. And in the picture, the different colors are also showing that evert time we just find the $\neg x$ in the clauses and repeating process.

Question 5:

Clauses and Watched Literals			
I	$p \vee q \vee r \vee s$	$p \vee \neg q \vee \neg t$	$p \vee t$
	p, q	$p, \neg q$	p, t
$\neg p$	q, r	$\neg q, \neg t$	p, t
t		$\neg q, \neg t$	
$\neg q$	r, s		

Hence, the closure of $\{\neg p\}$ under unit propagation relative to φ is $\{\neg p, t, \neg q\}$.

Question 6:

(a). **$\mathbf{KHappy} \wedge \mathbf{K}\neg\text{Happy}$ is unsatisfiable.**

Let e, w be an arbitrary interpretation. Suppose $e, w \models \mathbf{KHappy} \wedge \mathbf{K}\neg\text{Happy}$, which means that $e, w \models \mathbf{KHappy}$ and $e, w \models \mathbf{K}\neg\text{Happy}$. Then for the $e, w \models \mathbf{KHappy}$ which implies that for all $w' \in e, e, w' \models \text{Happy}$; and $e, w \models \mathbf{K}\neg\text{Happy}$ is that for all $w' \in e, e, w' \models \neg\text{Happy}$ and this is means that e, w' does not satisfy Happy. So, it conflicts with the $e, w' \models \text{Happy}$. Therefore, $\mathbf{KHappy} \wedge \mathbf{K}\neg\text{Happy}$ is unsatisfiable because there is no interpretation e, w satisfies \mathbf{KHappy} and also $\mathbf{K}\neg\text{Happy}$.

(b). **$\mathbf{K}(\text{Happy} \vee \text{Sad}) \rightarrow \neg\mathbf{KHappy}$ is satisfiable.**

Let e, w be an arbitrary interpretation. Suppose $e, w \models \mathbf{K}(\text{Happy} \vee \text{Sad})$, which means that for all $w' \in e, e, w' \models \text{Happy} \vee \text{Sad}$, then, for all $w' \in e, e, w' \models \text{Happy}$ or $e, w' \models \text{Sad}$.

While $\neg\mathbf{KHappy}$ is meaning there is some possible world satisfies $\neg\text{Happy}$. Such as the lecture notes:

- ▶ $\neg\mathbf{K}p$: some possible world satisfies $\neg p$.
- ▶ $\neg\mathbf{K}q$: some possible world satisfies $\neg q$.

This implies that for some $w' \in e, e, w' \models \neg\text{Happy}$.

Because the $\mathbf{K}(\text{Happy} \vee \text{Sad})$ means for all $w' \in e, e, w' \models \text{Happy}$ or $e, w' \models \text{Sad}$.

Hence, when some $w' \in e, e, w'$ cannot satisfy Happy which means that it satisfy the Sad in this case. This, in turn, lead to that for some $w' \in e, e, w'$ satisfy $\neg\text{Happy}$.

Therefore, $\mathbf{K}(\text{Happy} \vee \text{Sad}) \rightarrow \neg\mathbf{KHappy}$ is satisfiable, because there are some interpretations e, w satisfies that.

Another possible method to prove is that we can assume that we know all people are Happy or Sad. So, it means that there are some people are Happy and some people are Sad, but we do not know who is Happy or sad. Hence, $\mathbf{K}(\text{Happy} \vee \text{Sad}) \rightarrow \neg \mathbf{K}\text{Happy}$.

Question 7:

(a).

(a). Because in the example given above, we do not consider the predicate, clear(), Block() and HandEmpty which are not shown in the example. And Base on the STRIPS model which from the lecture notes (46/48), we can know that the positive effect to the Holding(x) is pick up(y) and the negative effect to Holding(x) is put on(y) or put on Table. So, this means that the successor-state axiom for the predicate Holding(x) is:

$$\Box \forall x \forall y \forall z (\text{Holding}(x) \leftrightarrow (\text{on}(x, y) \wedge (\neg \text{pick up}(y) \vee (y = \text{table} \wedge \neg \text{pick up}(\text{table}))))$$

or on y or table. Pick up from y or y is table. Pick up from table.

$$\vee (\text{Holding}(x) \wedge (\neg \text{put on}(y) \vee (y = \text{table} \wedge \neg \text{put on}(\text{table}))))$$

we already Holding(y) and we do not put y on y or put x on table.

(b).

(b). According to the lecture notes:

Let Σ contain the successor-state axioms for Holding(x), On(x, y)

$\phi \wedge \Sigma \models [\text{pu}(B)] [\text{po}(C)] \text{on}(B, C)$

iff $\phi \models R([\text{pu}(B)] [\text{po}(C)] \text{on}(B, C))$. Base on $\Box \forall x \forall y (\text{Holding}(x) \leftrightarrow (\text{H}(x) \wedge (\neg \text{pick up}(y) \vee (y = \text{table} \wedge \neg \text{pick up}(\text{table})))) \vee (\text{on}(x, y) \wedge \neg \text{put on}(y)))$

iff $\phi \models R([\text{pu}(B)] \vee_{\text{po}(C)}^{\alpha} \vee_{B \ C}^{\alpha} \vee_{B \ C}^{\alpha})$

iff $\phi \models R([\text{pu}(B)] (\text{H}(B) \wedge (\frac{\text{po}(C)=\text{po}(C)}{\text{true}} \vee (C = \text{table} \wedge \text{po}(C) = \text{put on}(\text{table}))) \vee (\text{on}(B, C) \wedge \frac{\text{po}(C) \neq \text{put on}(y)}{\text{true}})))$

iff $\phi \models R([\text{pu}(B)] (\text{H}(B) \vee \text{on}(B, C)))$

iff $\phi \models R([\text{pu}(B)] \text{H}(B) \vee R([\text{pu}(B)] \text{on}(B, C)))$

iff $\phi \models R([\text{pu}(B)] \text{H}(B) \vee R(\vee_{\text{on po}(B)}^{\alpha} \vee_{B \ C}^{\alpha} \vee_{B \ C}^{\alpha}))$

iff $\phi \models R([\text{pu}(B)] \text{H}(B) \vee (\text{H}(B) \wedge (\frac{\text{pu}(B)=\text{po}(y)}{\text{false}} \vee (y = \text{table} \wedge \frac{\text{pu}(B)=\text{put on}(\text{table})}{\text{false}})) \vee (\text{on}(x, y) \wedge \frac{\text{pu}(B) \neq \text{put on}(y)}{\text{false}})))$

iff $\phi \models R([\text{pu}(B)] \text{H}(B))$ Base on (a) question.

iff $\phi \models R(\vee_{\text{H po}(B)}^{\alpha} \vee_{B}^{\alpha})$

iff $\phi \models R(\text{on}(B, y) \wedge (\frac{\text{pu}(B)=\text{pu}(B)}{\text{true}} \vee (y = \text{table} \wedge \text{pu}(B) = \text{put on}(\text{table})))) \vee (\text{H}(B) \wedge (\frac{\text{pu}(B) \neq \text{po}(y)}{\text{true}} \vee \dots))$

iff $\phi \models R(\text{on}(B, y) \vee \text{H}(B))$

iff $\phi \models R(\text{on}(B, y)) \vee R(\text{H}(B))$ ✓

$y = C$, then

$\phi \models R(\text{on}(B, C)) \vee R(\text{H}(B))$

Because the action sequence is $[\text{pu}(B)] [\text{po}(C)]$, so, after $[\text{pu}(B)]$, we do $\text{po}(C)$ which means that $R(\text{H}(B))$ effect can be ignored. Hence, $\phi \models R(\text{on}(B, C))$, it is valid when y is C . Therefore, the whole formula is valid.

(c).

1. The Proposition Schema $clear(x)$ is redundant when using the Logic of Action & Model the blocks world.

The main reason is that we can use predicate $on(x,y)$ or $on(z,x)$ to determine whether x is clear or not. And also can use $holding(x)$ to express the following effects that x is clear or not. For example, the below can be expressed by:

$$1. \begin{array}{|c|} \hline x \\ \hline y \\ \hline \end{array} \quad \begin{array}{|c|} \hline x \\ \hline \end{array} : \quad \square \forall a \forall x \forall y ((\neg clear(x)) \leftrightarrow ((on(x,y) \vee (y=T \wedge on(x,y))) \wedge a \neq pickup(x) \wedge a \neq puton(x) \\ \vee (Holding(x) \wedge (a = puton(y) \vee (y=T \wedge a = putonTable)))).$$

$$2. \begin{array}{|c|} \hline z \\ \hline y \\ \hline \end{array} \quad \begin{array}{|c|} \hline z \\ \hline x \\ \hline \end{array} : \quad \square \forall a \forall x \forall y \forall z ((\neg clear(x)) \leftrightarrow (on(z,x) \wedge (on(x,y) \vee (y=T \wedge on(x,y))) \wedge a = pickup(z)).$$