COMP4418, 2018 – Assignment 2

ZID: 5097690   Name: Baixiang Guan

Question 1:

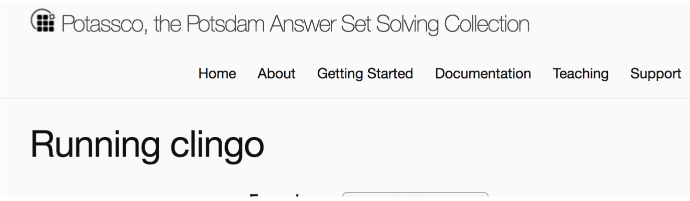
(a).

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.

```
% Question 1
%decide if there is a k-clique. such that whether there is a 3-clique.
#const k=2.

% Vertices all vertices from 1 to 6
v(1).v(2).v(3).v(4).v(5).v(6).

% Edges all eages and two direction
e(1,2).e(2,1).e(1,3).e(3,1).e(1,4).e(4,1).e(2,4).e(4,2).
e(2,5).e(5,2).e(2,6).e(6,2).e(3,4).e(4,3).e(3,5).e(5,3).
e(3,6).e(6,3).e(4,5).e(5,4).e(5,6).e(6,5).

% encoding let c(X) is at least k V(X) and at most k v(X)
% and if there is no edges betweenc(X),C(Y) then C is not clique.
k{c(X):v(X)}k.
:- not e(X, Y), c(X), c(Y), X != Y.

% Display
#show c/1.
```

(b).

Using the running clingo at online, like the picture at below, when k ∈ {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:

```
% Question 2

% encoding
a :- not b, not c.
b :- not a, not c.
c :- not b, not a.
d :- a.
d :- b.
d :- c.
```

Configuration: resoning 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
```

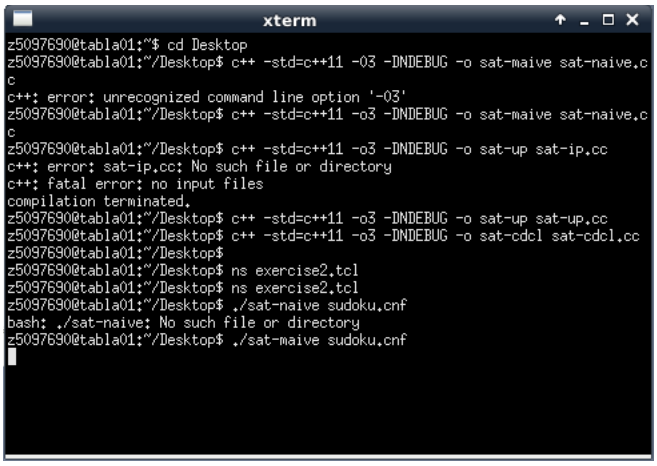
And the table is :

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.	✗

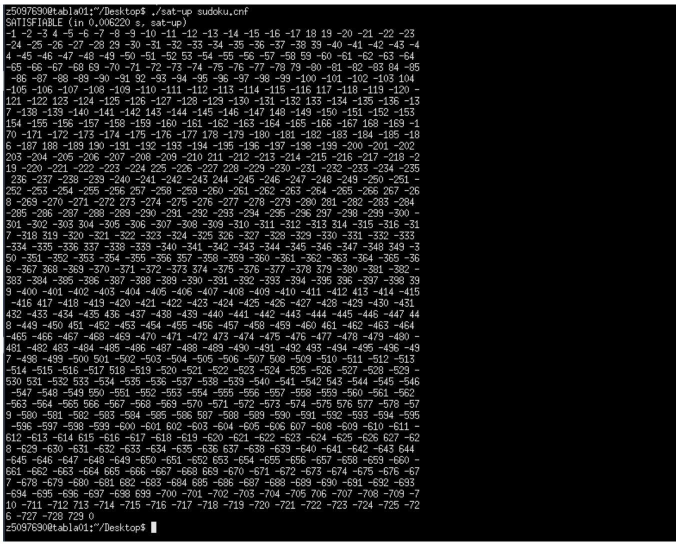
Question 3:

(a).

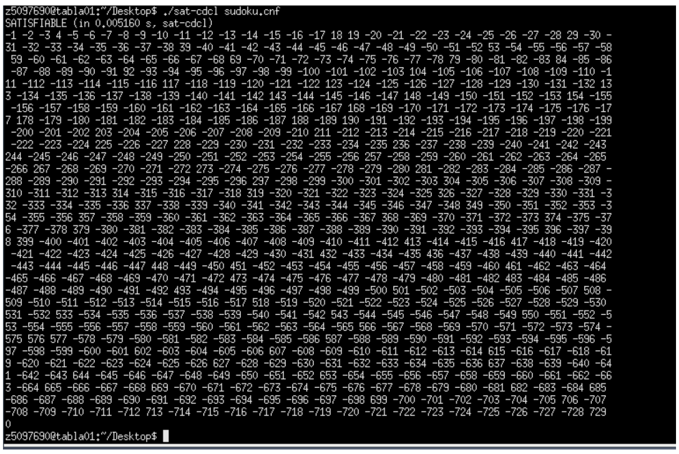
The run time for the sat-naive is more than 5 minutes.



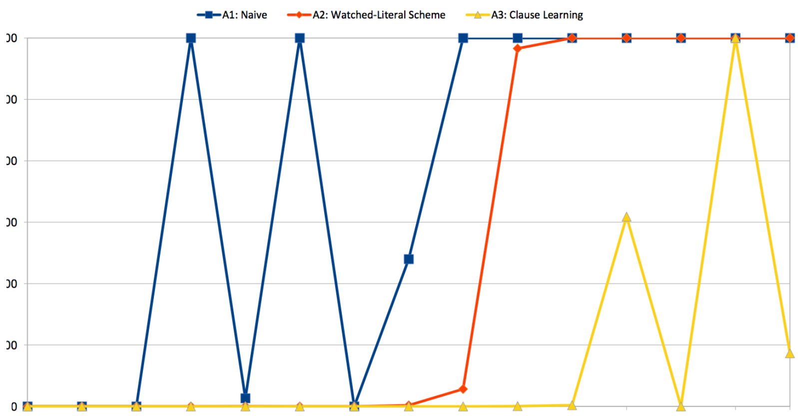
The run time for the sat-up is 0.006220s.



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



(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 naïve.
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 /= NP).
We can use a example to illustrate that.
For example, α ∨ β is converted to CNF as follows:
1.If α and β are literals which means that it is the atomic element, then α ∨ β is already the CNF such that (α ∨ β) ∧ others.
2. If α not literals, then α= α1 ∧ . . . ∧ αk and k >=2, then α ∨ β = (α1 ∨ β) ∧ . . . ∧ (αk ∨ β) and if β is literal then (α1 ∨ β) ∧ . . . ∧ (αk ∨ β) is already the CNF, otherwise, β = β1 ∧ . . . ∧ βk and k >=2. And we need to take one more distribution step to convert each αi ∧ βi 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 /= 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 ∪ {x} under unit propagation relative to ϕ it suffices to inspect the clauses c ∈ ϕ 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 $(x_1 \vee \ldots \vee x_k) \in \phi$ with $\bar{x}_1, \ldots, \bar{x}_k \in I^j$: Return conflict $(x_1 \vee \ldots \vee x_k)$
  - If there is a $(x_1 \vee \ldots \vee x_{k+1}) \in \phi$ with $\bar{x}_1, \ldots, \bar{x}_k \in I^j$: Let $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return I^j

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

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

We can see from the picture at above, the I^2 is closed under unit propagation relative to ϕ because $\mathrm{I}^2 = \mathrm{I}^3$. And if we add the $\{\neg s\}$ in the I, which can be describe by I = I ∪ $\{\neg s\} = \{\neg p, q, r, \neg s\}$, then we just inspect the clauses c ∈ ϕ 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 (x1 ∨ . . . ∨ xk+1) ∈ ϕ with x1, . . . , xk ∈ 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 ∈ ϕ 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:

I	Clauses and Watched Literals		
	p ∨ q ∨ r ∨ s	p ∨¬q∨¬t	p∨t
	p,q	p, ¬q	p,t
$\neg p$	q,r	¬q,¬t	p,t
t		¬q,¬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). **K**Happy∧**K**¬Happy **is unsatisfiable.**

Let e, w be an arbitrary interpretation. Suppose e, w |= **K**Happy∧**K**¬Happy, which means that e, w |= **K**Happy and e, w |= **K**¬Happy. Then for the e, w |= **K**Happy which implies that for all w′ ∈ e, e,w′ |= Happy; and e, w |= **K**¬Happy is that for all w′ ∈ e, e,w′ |= ¬Happy and this is means that e,w′ does not satisfy Happy. So, it conflicts with the e,w′ |= Happy. Therefore, **K**Happy∧**K**¬Happy is unsatisfiable because there is no interpretation e, w satisfies **K**Happy and also **K**¬Happy.

(b). **K**(Happy∨Sad)→¬**K**Happy **is satisfiable.**

Let e, w be an arbitrary interpretation. Suppose e, w |= **K**(Happy∨Sad) , which means that for all w′ ∈ e, e,w′ |= Happy∨Sad, then, for all w′ ∈ e, e,w′ |= Happy or e,w′ |= Sad.

While ¬**K**Happy is meaning there is some possible world satisfies ¬Happy. Such as the lecture notes:

- ¬**K**p: some possible world satisfies ¬p.
- ¬**K**q: some possible world satisfies ¬q.

This implies that for some w′ ∈ e, e,w′ |= ¬Happy.
Because the **K**(Happy∨Sad) means for all w′ ∈ e, e,w′ |= Happy or e,w′ |= Sad.
Hence, when some w′ ∈ 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′ ∈ e, e,w′ satisfy ¬Happy.
Therefore, **K**(Happy∨Sad)→¬**K**Happy 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 Pickup(y) and the negative effect to Holding(x) is puton(y). or put on Table. So, this means that the successor-state axiom for the predicate Holding(x) is :

$$\Box \forall a \forall x \forall y \Big([a]\,Holding(x) \leftrightarrow \big(\underbrace{on(x,y)}_{\text{(x on y or table)}} \wedge (a = pickUp(x) \vee (y = T \wedge a = pickUp(x))) \big)$$

(x on y or table) Pickup x from y or y is table. Pick up from Table.

$$\vee \big(\underbrace{Holding(x)}_{} \wedge (a \neq putOn(y) \vee (y = T \wedge a \neq putOnTable)) \big) \Big).$$

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

(b).

(b). According to the lecture notes:

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

$\phi \wedge \Sigma \models [pu(B)][po(c)]\ on(B,C)$.

iff $\phi \models \mathcal{R}([pu(B)][po(c)]\ on(B,C))$. Base on $\Box \forall a \forall x \forall y ([a]\,On(x,y) \leftrightarrow (H(x) \wedge (a = Po(y) \vee (y = T \wedge a = Pot))) \vee (on(x,y) \wedge a \neq pu(x)))$.

iff. $\phi \models \mathcal{R}([pu(B)]\ \gamma_{on}{}^{a\ \ x\ \ y}_{Po(c)\ B\ C})$

iff $\phi \models \mathcal{R}([pu(B)]\ (H(B) \wedge (\underbrace{Po(c) = Po(c)}_{true} \vee (C = T \wedge Po(c) = Pot))) \vee (on(B,C) \wedge \underbrace{Po(c) \neq Pu(x)}_{true.}))$

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

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

iff $\phi \models \mathcal{R}([pu(B)]\ H(B) \vee \mathcal{R}(\gamma_{on}{}^{a\ \ \ x\ \ y}_{pu(B)\ B\ C})$

iff $\phi \models \mathcal{R}([pu(B)]\ H(B) \vee (H(B) \wedge (\underbrace{Pu(B) = Po(y)}_{false} \vee (y = T \wedge \underbrace{Pu(B) = Pot}_{false}))) \vee (on(x,y) \wedge \underbrace{Pu(B) \neq Pu(B)}_{false.}))$

iff $\phi \models \mathcal{R}([pu(B)]\ H(B))$

Base on (a). question.

iff $\phi \models \mathcal{R}(\gamma_{H}{}^{a\ \ \ x}_{Pu(B)\ B})$

iff $\phi \models \mathcal{R}(on(B,y) \wedge (\underbrace{Pu(B) = Pu(B)}_{true} \vee (y = T \wedge Pu(B) = Pu(B)))) \vee (H(B) \wedge (\underbrace{Pu(B) \neq Po(y) \vee \cdots}_{true.})))$

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

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

$y = C$, then

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

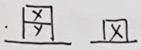
Because the action sequence is $[pu(B)][po(c)]$, so, after $[pu(B)]$. we do $po(c)$ which means that $\mathcal{R}(H(B))$ effect. can be igored. Hence. $\phi \models \mathcal{R}(on(B,C))$. it is valid <u>when y is C</u>

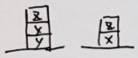
Therefore, the whole formula is valid.

(c).

(c). The proposition schema Clear(x) is redundant when using the Logic of Action to Model the blocks world.

The main reason is that we can use predicte 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.  : $\Box \forall a \forall x \forall y ([a]\, clear(x) \leftrightarrow ((on(x,y) \vee (y=T \wedge on(x,y))) \wedge a \neq Pickup(x) \wedge a \neq puton(x))$
$\vee (Holding(x) \wedge (a = puton(y) \vee (y=T \wedge a = putonTable(e)))$.

2.  ; $\Box \forall a \forall x \forall y \forall z ([a]\, clear(x) \leftrightarrow (on(z,x) \wedge (On(x,y) \vee (y=T \wedge on(x,y))) \wedge a = Pickup(z))$.