SoC Verification Homework 3

Arvind Singh Rathore
R10943157

#Ntk

1) Implementation

a) void V3Ntk::buildNtkBdd()
Firstly, Perform DFS traversal from DFF inputs, inout, and output gates.
Then I used getLatch(i),getInout(i), getOutput(i) for each for loop iteration to capture V3NetId, and use it to build Bdd.

b) void V3Ntk::buildBdd(const V3NetId& netId)
With the DFS list obtained , I built the network from the inputs by the order of DFS list. The sole get type I have to deal with is AIG_NODE. Checking both two of inputs of AIG_NODE and inversing them if the following conditions hold:

```
    if(getInputNetId(DepthId,0).cp) L0Node = ~L0Node;
    if(getInputNetId(DepthId,1).cp) R1Node = ~R1Node;
```

i.e the input has a complement.
Last, by using operator '&', I calculate the result of this AIG_NODE and save to bddMgr. Else, if the node isn't AIG_NODE, just don't bother & save it.

#Prove

c) void BddMgrV::buildPInitialState()
Here we want to set all the flip-flips to 0 as initial state.
In function form we can write as $\prod(S_i \equiv 0) = \prod(\sim S_i 1)$. where S ~ denotes FF state.
Therefore, I inversed all states and "and" them.

```
   const V3NetId& LntkId = NtkHand -> getLatch(i);
   BddNodeV BddL = bddMgrV -> getBddNodeV(LntkId.id);
   _initState &= ~BddL;
```

d) void BddMgrV:: buildPTransRelation()

From CH5, Transition relationship is defined as follows
TR(Y, X, I) = $\prod(y_i \equiv \mathrm{delta}_i(X, I))$,
This can be broken into each separate parts $y_i \equiv \mathrm{delta}_i(X, I)$ and "and" all of them.
For y_i, find each latch next state. For $\mathrm{delta}_i(X, I)$, get input of the latch.
With y_i and $\mathrm{delta}_i(X, I)$, I execute exclusive nor and get $\prod(y_i \equiv \mathrm{delta}_i(X, I))$. By doing "and" of all latches, _tri is finished. Since TR(Y, X) = $\exists_i$TR(Y, X, I), I called exists function. By existing all inputs for _tri, I got _tr.

e) void BddMgrV:: buildPImage(int level)

Also, from CH5, the definition of Image is $S_{n+1}(Y) = \exists X, I((TR(Y, X, I) \wedge S_n(x)))$, Not only I have to compute Y, but I also have to change it to X for next compute by S_{n+1}(X) = $S_{n+1}(Y)\ |_{Y\to X}$.
From buildPTransRelation() function, _tri and _tr are computed.
Keep in mind that S_0(X) is initial state built in buildPInitialState(). Hence, I have all inputs of the above two functions.
$I((TR(Y, X, I) \wedge S_n(x)))$ consists of all operators supported by the code.
I save each S_i(X) to _reachState after computing, if the new state is identical to old state, the flag _isFixed is set to be true.

f) void BddMgrV::runPCheckProperty(const string &name, BddNodeV monitor)

For the First part to determine whether monitor is safe , I just “and” it with the last reachable state and see whether the result is zero. If zero it means the monitor is not in reach state; therefore, it is safe.
Else, if it is not safe, I have to find the counter example, which is the second part.

In the second part, our goal is to find every I, however, we only have reach states which are X. I then calculate it by the following approach. Here R_n represents nth reach state.

$$S_n(X) = R_n$$
$$S_n(Y) = S_n(X)|_{x\to y}$$
$$S_{n-1}(Y, X, I) = S_n(Y) \wedge TR(Y, X, I) \wedge R_{n-1}$$
$$S_{n-1}(I) = \exists YX\, S_{n-1}(Y, X, I)$$
$$S_{n-1}(X) = \exists YI\, S_{n-1}(Y, X, I)$$

This approach shows that the process is recursive. When we reach the beginning state, it ends. I iteratively found Monitor ⋀ Rn from the last reach state and traced the first state that errored to discover the initial start point. The start point of the aforesaid strategy is where the first state error occurs.

2) The assertions you add and their meanings

i) **assign checkforinitialized = !initialized && ((serviceTypeOut != `SERVICE_OFF) ||(outExchange != 0) || (itemTypeOut != `ITEM_NONE));**

Assertion for checking the machine won’t run and output money or item before it is initialized.

ii) **assign p = initialized && (serviceTypeOut == `SERVICE_OFF) && (itemTypeOut == `ITEM_NONE) && (outExchange != inputValue);**

Assertion for checking the machine won’t gobble money.

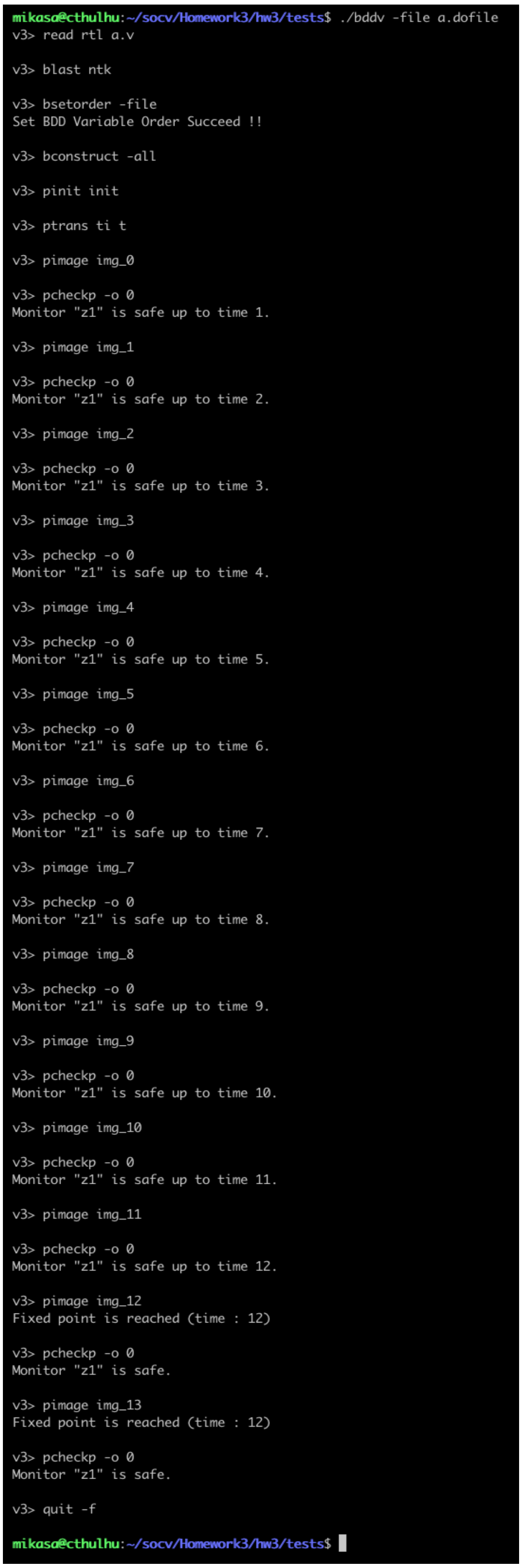
```
assign outExchange = (`VALUE_NTD_50 * {5'd0, coinOutNTD_50}) +
               (`VALUE_NTD_10 * {5'd0, coinOutNTD_10}) +
               (`VALUE_NTD_5  * {5'd0, coinOutNTD_5 }) +
               (`VALUE_NTD_1  * {5'd0, coinOutNTD_1 });

assign checkitemvalue = itemTypeOut == `ITEM_A ? `COST_A :
                 itemTypeOut == `ITEM_B ? `COST_B :
                 itemTypeOut == `ITEM_C ? `COST_C : 8'd0;
```

iii) **assign checkcorrectchange = initialized && (serviceTypeOut == `SERVICE_OFF) && (outExchange != inputValue - checkitemvalue);**

Assertion for checking the exchange from the machine is correct or not.

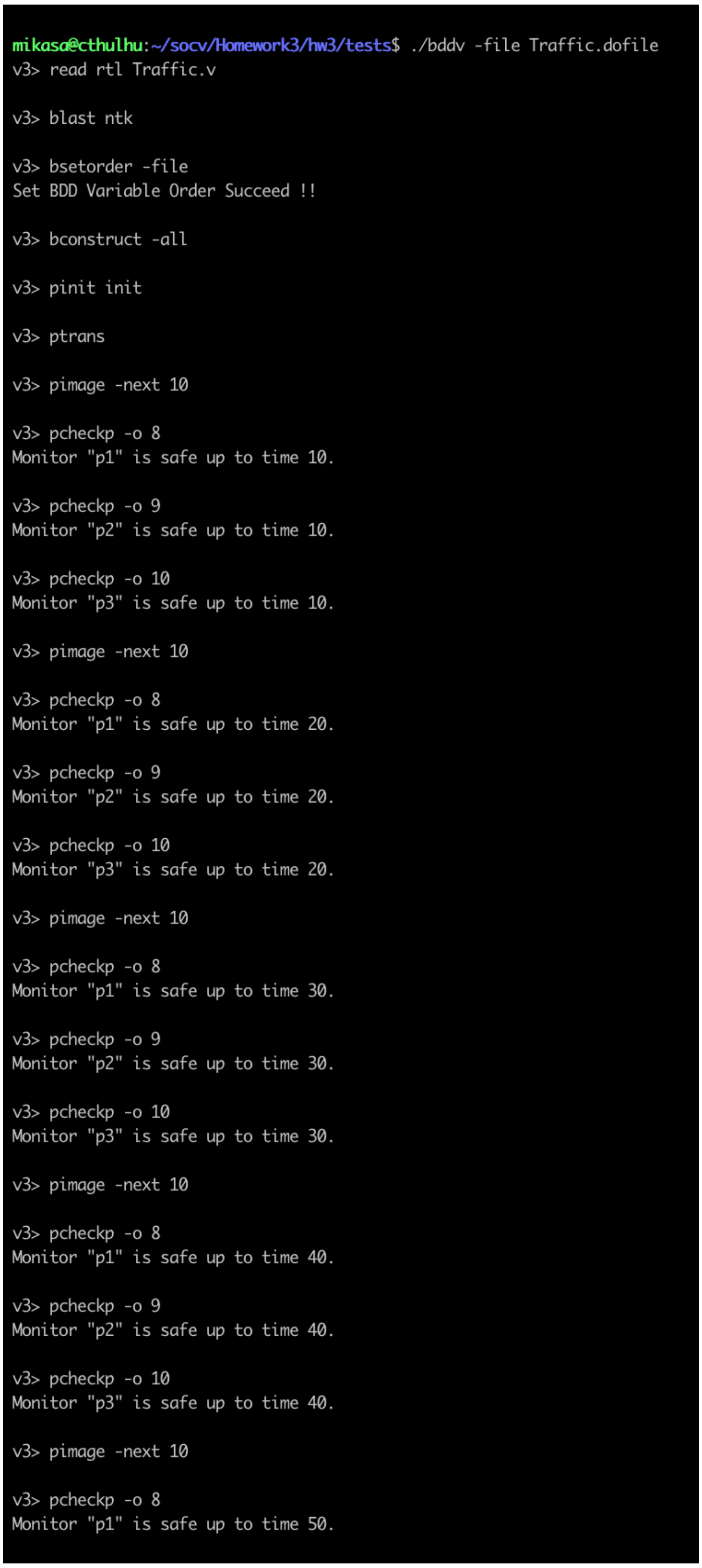
3) Verification Results

1) a.dofile

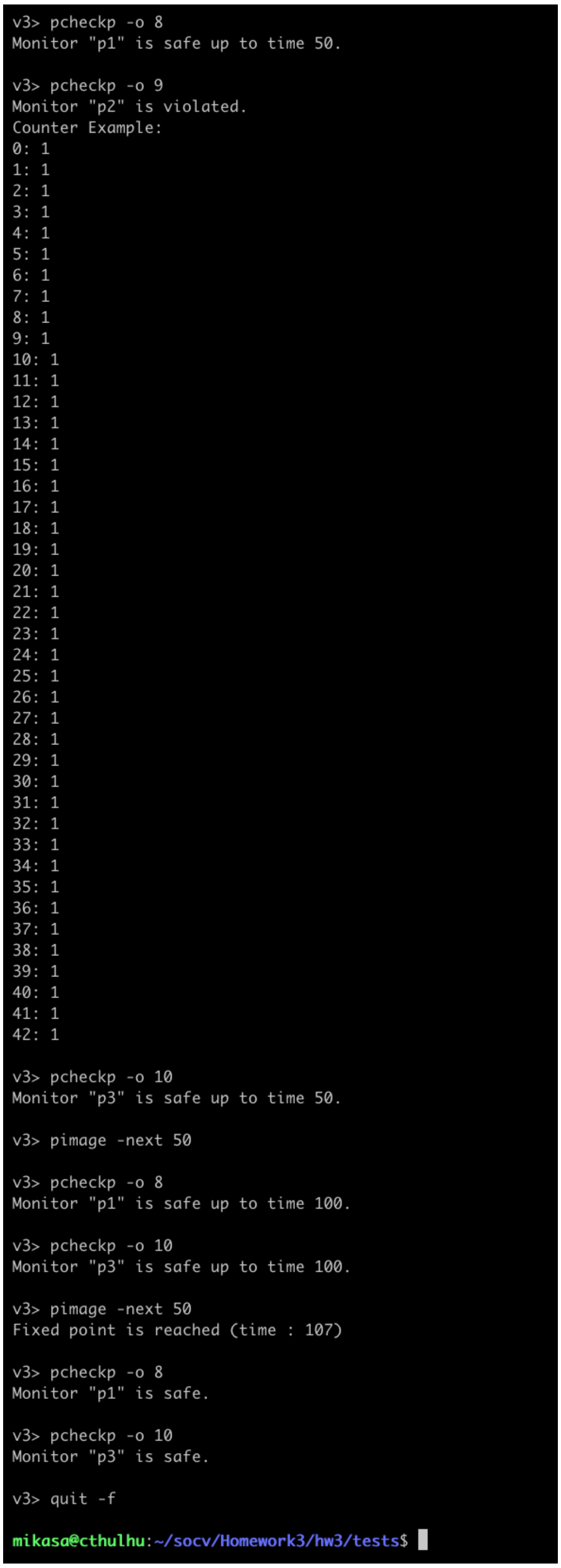


```
mikasa@cthulhu:~/socv/Homework3/hw3/tests$ ./bddv -file a.dofile
v3> read rtl a.v

v3> blast ntk

v3> bsetorder -file
Set BDD Variable Order Succeed !!

v3> bconstruct -all

v3> pinit init

v3> ptrans ti t

v3> pimage img_0

v3> pcheckp -o 0
Monitor "z1" is safe up to time 1.

v3> pimage img_1

v3> pcheckp -o 0
Monitor "z1" is safe up to time 2.

v3> pimage img_2

v3> pcheckp -o 0
Monitor "z1" is safe up to time 3.

v3> pimage img_3

v3> pcheckp -o 0
Monitor "z1" is safe up to time 4.

v3> pimage img_4

v3> pcheckp -o 0
Monitor "z1" is safe up to time 5.

v3> pimage img_5

v3> pcheckp -o 0
Monitor "z1" is safe up to time 6.

v3> pimage img_6

v3> pcheckp -o 0
Monitor "z1" is safe up to time 7.

v3> pimage img_7

v3> pcheckp -o 0
Monitor "z1" is safe up to time 8.

v3> pimage img_8

v3> pcheckp -o 0
Monitor "z1" is safe up to time 9.

v3> pimage img_9

v3> pcheckp -o 0
Monitor "z1" is safe up to time 10.

v3> pimage img_10

v3> pcheckp -o 0
Monitor "z1" is safe up to time 11.

v3> pimage img_11

v3> pcheckp -o 0
Monitor "z1" is safe up to time 12.

v3> pimage img_12
Fixed point is reached (time : 12)

v3> pcheckp -o 0
Monitor "z1" is safe.

v3> pimage img_13
Fixed point is reached (time : 12)

v3> pcheckp -o 0
Monitor "z1" is safe.

v3> quit -f

mikasa@cthulhu:~/socv/Homework3/hw3/tests$
```

b) Traffic.dofile

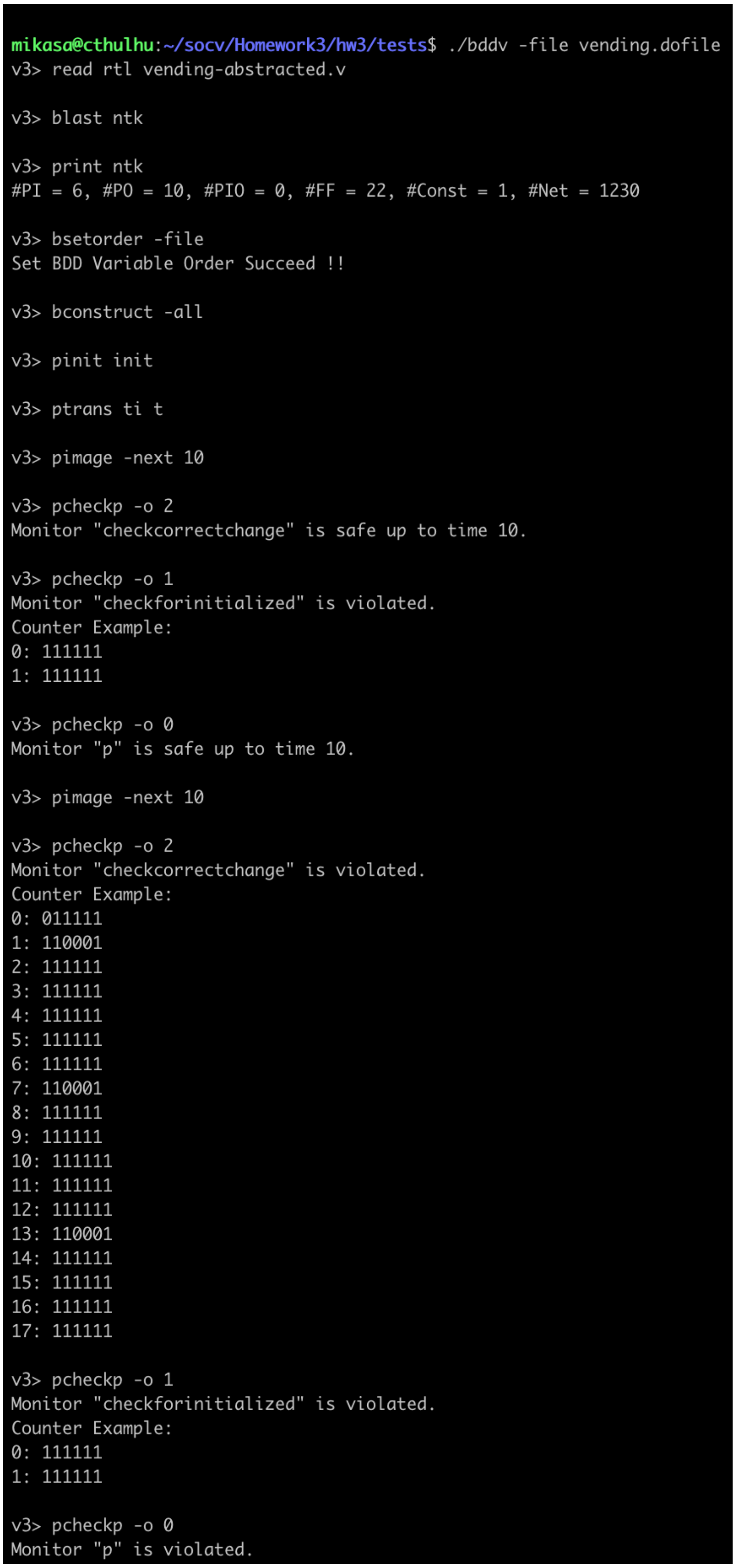


```
mikasa@cthulhu:~/socv/Homework3/hw3/tests$ ./bddv -file Traffic.dofile
v3> read rtl Traffic.v

v3> blast ntk

v3> bsetorder -file
Set BDD Variable Order Succeed !!

v3> bconstruct -all

v3> pinit init

v3> ptrans

v3> pimage -next 10

v3> pcheckp -o 8
Monitor "p1" is safe up to time 10.

v3> pcheckp -o 9
Monitor "p2" is safe up to time 10.

v3> pcheckp -o 10
Monitor "p3" is safe up to time 10.

v3> pimage -next 10

v3> pcheckp -o 8
Monitor "p1" is safe up to time 20.

v3> pcheckp -o 9
Monitor "p2" is safe up to time 20.

v3> pcheckp -o 10
Monitor "p3" is safe up to time 20.

v3> pimage -next 10

v3> pcheckp -o 8
Monitor "p1" is safe up to time 30.

v3> pcheckp -o 9
Monitor "p2" is safe up to time 30.

v3> pcheckp -o 10
Monitor "p3" is safe up to time 30.

v3> pimage -next 10

v3> pcheckp -o 8
Monitor "p1" is safe up to time 40.

v3> pcheckp -o 9
Monitor "p2" is safe up to time 40.

v3> pcheckp -o 10
Monitor "p3" is safe up to time 40.

v3> pimage -next 10

v3> pcheckp -o 8
Monitor "p1" is safe up to time 50.
```

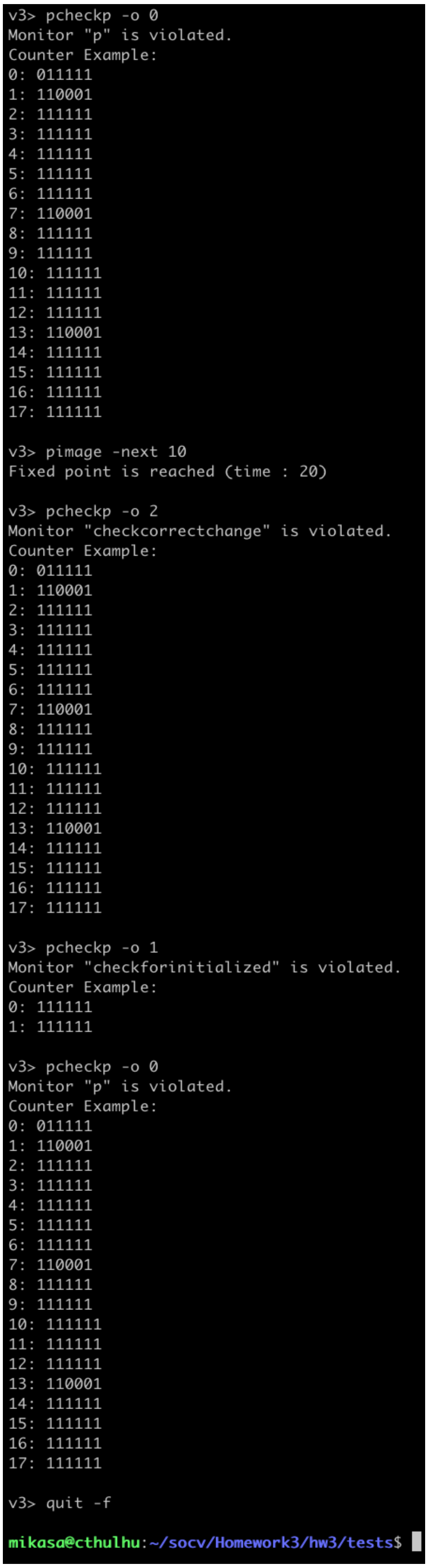


```
v3> pcheckp -o 8
Monitor "p1" is safe up to time 50.

v3> pcheckp -o 9
Monitor "p2" is violated.
Counter Example:
0: 1
1: 1
2: 1
3: 1
4: 1
5: 1
6: 1
7: 1
8: 1
9: 1
10: 1
11: 1
12: 1
13: 1
14: 1
15: 1
16: 1
17: 1
18: 1
19: 1
20: 1
21: 1
22: 1
23: 1
24: 1
25: 1
26: 1
27: 1
28: 1
29: 1
30: 1
31: 1
32: 1
33: 1
34: 1
35: 1
36: 1
37: 1
38: 1
39: 1
40: 1
41: 1
42: 1

v3> pcheckp -o 10
Monitor "p3" is safe up to time 50.

v3> pimage -next 50

v3> pcheckp -o 8
Monitor "p1" is safe up to time 100.

v3> pcheckp -o 10
Monitor "p3" is safe up to time 100.

v3> pimage -next 50
Fixed point is reached (time : 107)

v3> pcheckp -o 8
Monitor "p1" is safe.

v3> pcheckp -o 10
Monitor "p3" is safe.

v3> quit -f

mikasa@cthulhu:~/socv/Homework3/hw3/tests$
```

c) vending-abstracted.v



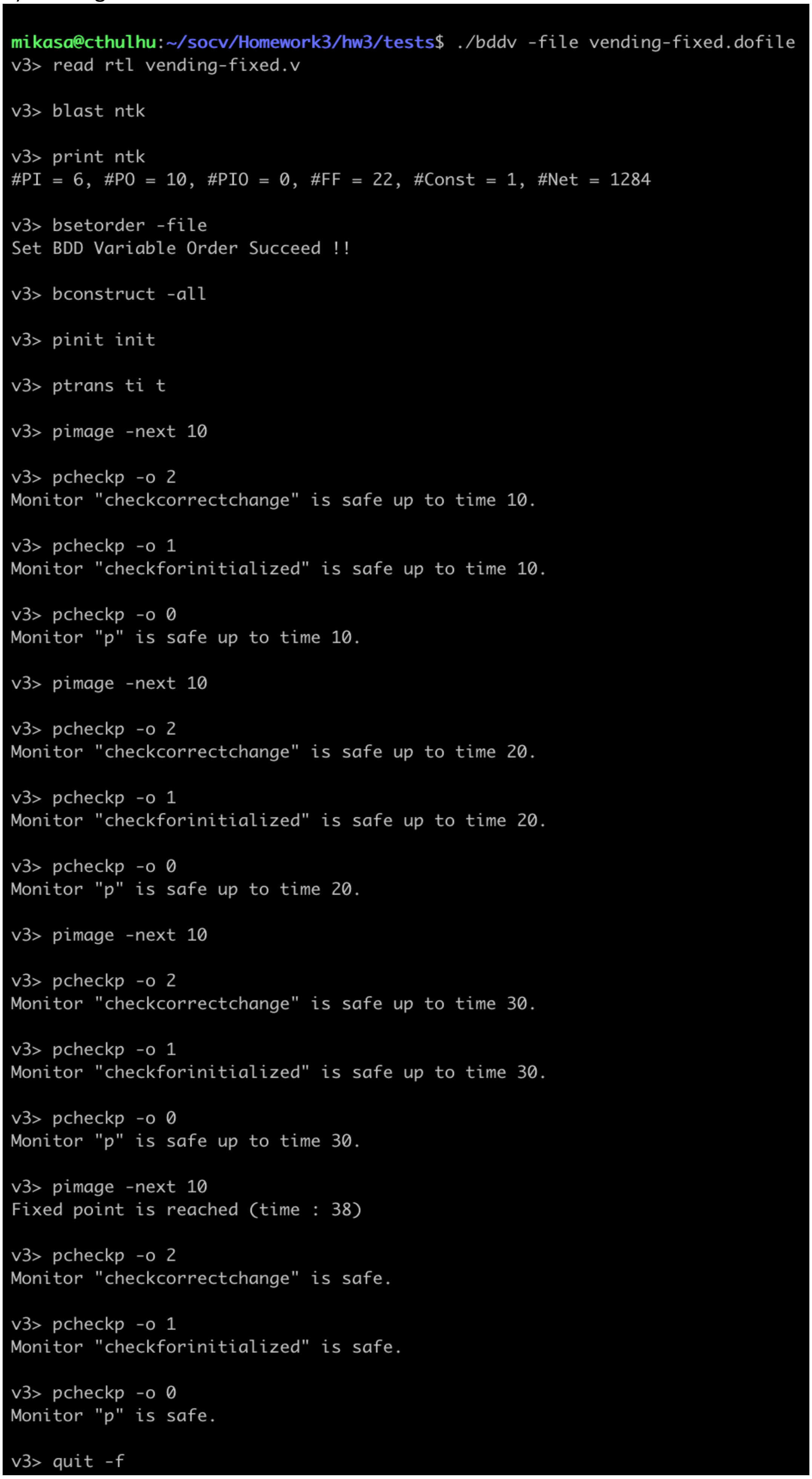
```
mikasa@cthulhu:~/socv/Homework3/hw3/tests$ ./bddv -file vending.dofile
v3> read rtl vending-abstracted.v

v3> blast ntk

v3> print ntk
#PI = 6, #PO = 10, #PIO = 0, #FF = 22, #Const = 1, #Net = 1230

v3> bsetorder -file
Set BDD Variable Order Succeed !!

v3> bconstruct -all

v3> pinit init

v3> ptrans ti t

v3> pimage -next 10

v3> pcheckp -o 2
Monitor "checkcorrectchange" is safe up to time 10.

v3> pcheckp -o 1
Monitor "checkforinitialized" is violated.
Counter Example:
0: 111111
1: 111111

v3> pcheckp -o 0
Monitor "p" is safe up to time 10.

v3> pimage -next 10

v3> pcheckp -o 2
Monitor "checkcorrectchange" is violated.
Counter Example:
0: 011111
1: 110001
2: 111111
3: 111111
4: 111111
5: 111111
6: 111111
7: 110001
8: 111111
9: 111111
10: 111111
11: 111111
12: 111111
13: 110001
14: 111111
15: 111111
16: 111111
17: 111111

v3> pcheckp -o 1
Monitor "checkforinitialized" is violated.
Counter Example:
0: 111111
1: 111111

v3> pcheckp -o 0
Monitor "p" is violated.
```



```
v3> pcheckp -o 0
Monitor "p" is violated.
Counter Example:
0: 011111
1: 110001
2: 111111
3: 111111
4: 111111
5: 111111
6: 111111
7: 110001
8: 111111
9: 111111
10: 111111
11: 111111
12: 111111
13: 110001
14: 111111
15: 111111
16: 111111
17: 111111

v3> pimage -next 10
Fixed point is reached (time : 20)

v3> pcheckp -o 2
Monitor "checkcorrectchange" is violated.
Counter Example:
0: 011111
1: 110001
2: 111111
3: 111111
4: 111111
5: 111111
6: 111111
7: 110001
8: 111111
9: 111111
10: 111111
11: 111111
12: 111111
13: 110001
14: 111111
15: 111111
16: 111111
17: 111111

v3> pcheckp -o 1
Monitor "checkforinitialized" is violated.
Counter Example:
0: 111111
1: 111111

v3> pcheckp -o 0
Monitor "p" is violated.
Counter Example:
0: 011111
1: 110001
2: 111111
3: 111111
4: 111111
5: 111111
6: 111111
7: 110001
8: 111111
9: 111111
10: 111111
11: 111111
12: 111111
13: 110001
14: 111111
15: 111111
16: 111111
17: 111111

v3> quit -f

mikasa@cthulhu:~/socv/Homework3/hw3/tests$
```

All my 3 monitors have a counter example which is certainly buggy!

d) vending-fixed.v



```
mikasa@cthulhu:~/socv/Homework3/hw3/tests$ ./bddv -file vending-fixed.dofile
v3> read rtl vending-fixed.v

v3> blast ntk

v3> print ntk
#PI = 6, #PO = 10, #PIO = 0, #FF = 22, #Const = 1, #Net = 1284

v3> bsetorder -file
Set BDD Variable Order Succeed !!

v3> bconstruct -all

v3> pinit init

v3> ptrans ti t

v3> pimage -next 10

v3> pcheckp -o 2
Monitor "checkcorrectchange" is safe up to time 10.

v3> pcheckp -o 1
Monitor "checkforinitialized" is safe up to time 10.

v3> pcheckp -o 0
Monitor "p" is safe up to time 10.

v3> pimage -next 10

v3> pcheckp -o 2
Monitor "checkcorrectchange" is safe up to time 20.

v3> pcheckp -o 1
Monitor "checkforinitialized" is safe up to time 20.

v3> pcheckp -o 0
Monitor "p" is safe up to time 20.

v3> pimage -next 10

v3> pcheckp -o 2
Monitor "checkcorrectchange" is safe up to time 30.

v3> pcheckp -o 1
Monitor "checkforinitialized" is safe up to time 30.

v3> pcheckp -o 0
Monitor "p" is safe up to time 30.

v3> pimage -next 10
Fixed point is reached (time : 38)

v3> pcheckp -o 2
Monitor "checkcorrectchange" is safe.

v3> pcheckp -o 1
Monitor "checkforinitialized" is safe.

v3> pcheckp -o 0
Monitor "p" is safe.

v3> quit -f
```

Here when I fixed the bug, I see all my monitors are safe.

4) Comparision with bddv-ref

1) Time

Time taken	a.dofile	Traffic.dofile	vending.dofile	Vending-fixed.dofile
Bddv	Real: 0m2.008s User: 0m1.940s Sys: 0m0.028s	Real: 0m0.696s User: 0m0.644s Sys: 0m0.008s	Real: 4m14.939s User: 4m14.228s Sys: 0m0.688s	Real: 7m11.445s User: 7m10.696s Sys: 0m0.752s
Bddv-ref	Real: 0m0.659s User: 0m0.636s Sys: 0m0.020s	Real: 0m0.163s User: 0m0.152s Sys: 0m0.004s	Real: 1m9.686s User: 1m9.180s Sys: 0m0.504s	Real: 2m3.994s User: 2m3.388s Sys: 0m0.604s

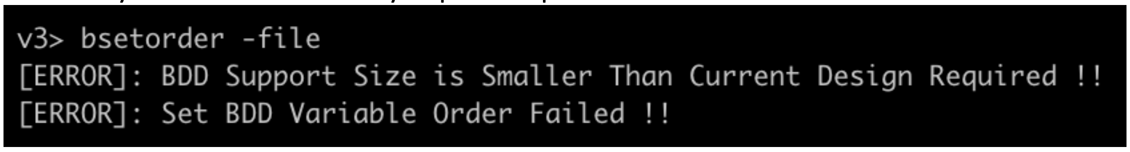
My Bddv is about three times slower than Bddv-ref

2) Accuracy
For the 4 do-files (a.dofile, Traffic.dofile, vending.dofile, vending-fixed.dofile),
I get the same results as the reference. I was able to replicate the results with my Bddv as shown in the screenshots above.

5) The advanced techniques and/or abstraction of the design

The original vending machine design is really big, with #PI = 11, #PO = 19, #PIO = 0, #FF = 48, #Const = 1, #Net = 6154. This enormous design can't be sent into Bdd.

We always encounter memory explosion problem.



```
v3> bsetorder -file
[ERROR]: BDD Support Size is Smaller Than Current Design Required !!
[ERROR]: Set BDD Variable Order Failed !!
```

So, I reduced the size of the design and tried different combinations to reduce the parameters. As a result, I've abstracted the design. To begin, I minimize the number of objects to just one. Second, I reduce the amount of money input to two dollars, one dollar, and three dollars, accordingly. The reason for picking 3 instead of 5 is that 5 would require at least four 1 dollar to save in order to generate all dollars of change, allowing the output number of coin 1 register to be 3 digits since 0-4 requires 5 (3 bits), and 3*5=15 if the value was larger than 16, requiring additional space to save.

By reducing the size to #PI = 6, #PO = 10, #PIO = 0, #FF = 22, #Const = 1, #Net = 1230, I am able to reduce the size to #PI = 6, #PO = 10, #PIO = 0, #FF = 22, #Const = 1, #Net = 1230, which is 5 times smaller than the original design. #PI = 6, #PO = 10, #PIO = 0, #FF = 22, #Const = 1, #Net = 1284 is the size of the fixed one.

However, this abstracted design is quite basic model of vending machine. BDD isn't the best way to run such a complicated design. Perhaps we should use SAT solvers to verify this vending machine since BDD cannot even verify a simple vending machine without using advanced techniques to overcome memory explosion problem.