(1) My implementation:

(i)
```
buildBdd(netId) {
    dfsOrder(netId, vNets);
    for each(net, vNets)
        if( net is AIG_NODE ) BAND(net, net.f0, net.f1);
}
```

(ii)
```
buildNtkBdd() {
    for each(latch, Latches) dfsOrder(latch.f0, vNets);
    for each(inout, Inouts) dfsOrder(inout, vNets);
    for each(out, Outputs) dfsOrder(out, vNets);
    for each(net, vNets)
        if( net is AIG_NODE ) BAND(net, net.f0, net.f1);
}
```

(iii)
```
buildPInitialState() {
    BddNodeV init = BddNodeV::_one;
    for each(latch, Latches) {
        BddNodeV tmp = getBddNodeV(latch);
        init &= ~tmp;
    }
}
```

(iv)
```
buildPTransRelation() {
    BddNodeV tri = BddNodeV::_one;
    for each(latch, Latches) {
        BddNodeV nState = getBddNodeV(latch.name + "_ns");
        BddNodeV nFunc = getBddNodeV(latch.f0);
        tri &= ~(nState ^ nFunc);
    }
    BddNodeV tr = tri;
    for( i = 1 to inputSize ) tr = tr.exist(i);
}
```

(v)
```
buildPImage(level) {
    for( i = 1 to level ) {
        BddNodeV rStates = getPReachState();
        BddNodeV nStates = rStates & _tr;
        for( j = cState_level to nState_level ) nState = nStates.exit(j);
```

```
            nStates.nodeMove(nState_level, cState_level);
            if( nState == cState ) setIsFixed();
            else pushBackPReachState(nState);
        }
    }
```

(vi)

```
    runPCheckProperty(name, monitor) {
        BddNodeV rStates = getPReachState();
        BddNodeV intersect = rStates & monitor;
        if( intersect == BddNodeV::_zero ) return;          // safe
        else {
            BddNodeV cube = getCube();
            BddNodeV pi = getPi(cube);
            BddNodeV cState = getCState(cube);
            vector<BddNodeV> vPi;
            for( i = 1 to _reachStates.size() ) {
                BddNodeV nState = cState.nodeMove(cState_level, nState_level);
                nState &= _tri;
                ith = 0;
                while( 1 ) {
                    cube = nState.getCube(ith++);
                    cState = getCState(cube);
                    intersect = cState & previosPReachState(i);
                    if( intersect == BddNodeV::_zero ) continue;
                    pi = getPi(cube);
                    vPi.push_back(pi);
                }
            }
            reportPi(vPi);
        }
    }
```

(2) Assertions for vending.v:

```
(1)

p = initialized && (serviceTypeOut == `SERVICE_OFF) && (outExchange + cost != inputValue),
where cost is the corressponding cost of the item we want.

This is more general than the original assertion p.
If no item comes out, the cost is 0, then the assertion is the same as the original one.
If we get an item, the outExchange should be inputValue - cost.
```

```
(2)

q = initialized && (serviceTypeOut == `SERVICE_BUSY) && exchangeReady && (outExchange + serviceValue != inputValue - cost)

This assertion tells that when the vending machine is in the exchanging process, the total value to be exchanged is inputValue - cost,
which should be exactly the same as the current exchanged value + the remaining should-be-exchanged value.
```

```
(3)

r = initialized && (serviceTypeOut == `SERVICE_ON) && (coinOutNTD_50 || coinOutNTD_10 || coinOutNTD_5 || coinOutNTD_1 || serviceValue)

This assertion tells that when the vending machine is at stand-by-state, there should be no coins coming out, and the serviceValue should also be 0.
```

(3) Verification results:

The proof complexity is too big to solve.

(4) Comparison with ref:

ref can’t deal with this problem either.

(5) Abstraction of vending.v:

I try to reduce the original design by:

(i) Reducing coin types to NTD_5 and NTD_1.

(ii) Reducing coin value of NTD_5 to 3.

(iii) Reducing item types to only ITEM_A.

(iv) Reducing item cost of ITEM_A to 4.

The vending mechanism remains unchanged.

By the reduction, the proof complexity narrows and the problem becomes solvable.

The figure below shows that my implementation is slower than the ref program.

But it is fine to solve the problem, and all three assertions are safe!

```
r04943179@valkyrie:~/SoCV_HWs/hw3/tests$ ../bddv -f vending.dofile
v3> read rtl vending_abs.v

v3> blast ntk

v3> bseto -f
Set BDD Variable Order Succeed !!

v3> bconst -all

v3> pinit init

v3> ptran tri tr

v3> usage
Period time used : 10.76 seconds
Total time used  : 10.76 seconds
Peak memory used   : 133.6 M Bytes
Total memory used  : 128.1 M Bytes
Current memory used: 136.1 M Bytes

v3> pimage -n 30
Fixed point is reached (time : 27)

v3> usage
Period time used : 24.9 seconds
Total time used  : 35.66 seconds
Peak memory used   : 134.4 M Bytes
Total memory used  : 128.9 M Bytes
Current memory used: 136.9 M Bytes

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

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

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

v3> q -f
```

```
r04943179@valkyrie:~/SoCV_HWs/hw3/tests$ ../ref/bddv-ref -f vending.dofile
v3> read rtl vending_abs.v

v3> blast ntk

v3> bseto -f
Set BDD Variable Order Succeed !!

v3> bconst -all

v3> pinit init

v3> ptran tri tr

v3> usage
Period time used : 3.78 seconds
Total time used  : 3.78 seconds
Peak memory used   : 133 M Bytes
Total memory used  : 126.4 M Bytes
Current memory used: 135.1 M Bytes

v3> pimage -n 30
Fixed point is reached (time : 27)

v3> usage
Period time used : 7.7 seconds
Total time used  : 11.48 seconds
Peak memory used   : 134 M Bytes
Total memory used  : 127.5 M Bytes
Current memory used: 136.2 M Bytes

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

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

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

v3> q -f
```