# SoCV HW3 Report

電機丙研二 R08921053 梁峻瑋

**(i) My Implementation**

1. V3NtkBdd.cpp
   (a) void V3Ntk::buildNtkBdd()
   針對"DFF input", "inout", "output", 分別跑 for-loop, 再用 getLatch(i), getInout(i), getOutput(i)來抓出 V3NetId, 使用來建立 Bdd.

   (b) void V3Ntk::buildBdd()
   用 for-loop traversal sorted nets (gates). 只需要對於 AIG gate, 進行 ANG BddNodeV, 以及 addBddNodeV()的操作.

2. proveBdd.cpp
   (a) void BddMgrV::buildPInitialState()
   精神上, 要設定_initState := (~x0)(~x1)...(~xn)! 因為_initState=1 是我們要解的對象, 會得到 x0,…,xn=0 的解, 也就是初始化的意義. 實作上, 就是" _initState &= ~getSupport(ntk->getLatch(i).id)" with for-loop visiting all DFF-input xi.

   (b) Void BddMgrV::buildPTransRelation()
   精神上, 要做 TR(Y,X,I) = ~(Y^delta(X,I)), TR(Y,X) = exist I TR(Y,X,I)這兩步, 也就是算出 TR, TRI. 實際上, 就比照(a)部分, 跑 for-loop 就好.

   (c) Void BddMgrV::buildPImage( int level )
   這邊的寫法會特別一點:

```
void
BddMgrV::buildPImage( int level )
{
  // TODO : remember to add _reachStates and set _isFixed
  // Note:: _reachStates record the set of reachable states
  for(int i=0; i<level; ++i){
    if(!_isFixed)
      BddMgrV::buildPImageStep();
  }
  if(_isFixed)
    cout << "Fixed point is reached (time : " << _reachStates.size()-1 << ") " << endl;
}
```

   針對給定的 level(步數), 去呼叫 buildImageStep(), 直到_isFixed TRUE 為止. 最後再 cout information if _isFixed TRUE. 至於 buildImageStep()內, 則做了四個動作—S(Y,X) = TR(Y,X) ^ Rn(X), S(Y) = existX S(Y,X), S(Y) -> S(X), Rn+1(X)

= Rn(X)|S(X), set _isFixed. 細節部分就不一一解說.

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

這邊, 先附上程式碼:

```
  // TODO : prove the correctness of AG(~monitor)
  V3Ntk* ntk = v3Handler.getCurHandler()->getNtk();
  BddNodeV target = monitor & getPReachState();

  if(target.countCube()!=0){
    //counter-example
    cout << "Monitor \"" << name << "\" is violated." << endl;
    cout << "Counter Example:" << endl;
    int i = _reachStates.size()-1;
    while((monitor & _reachStates[i]).countCube()!=0)
      i--;
    i++;
    BddNodeV S = target.getCube(), V;
    bool isMoved;
    vector<string> output;
    for(i; i>=0; --i){
      //Step1: Vn(I) = exist Y,X "TRI & Sn+1(X->Y)"
      S = S.nodeMove(ntk->getLatch(0).id, ntk->getLatch(0).id+ntk->getLatchSize(), isMoved);
      V = getPTri() & S;
      for(unsigned i = 0, n = ntk->getLatchSize(); i < n; ++i) {
        V = V.exist(ntk->getLatch(i).id);
      }
      for(unsigned i = 0, n = ntk->getLatchSize(); i < n; ++i) {
        V = V.exist(ntk->getLatch(i).id+ntk->getLatchSize());
      }

      //Step2: Sn(X) = exist Y,I "TRI & Sn+1(X->Y) & Vn"
      S = (getPTri() & S & V);
      for(unsigned i = 0, n = ntk->getInputSize(); i < n; ++i) {
        S  = S.exist(ntk->getInput(i).id);
      }
      for(unsigned i = 0, n = ntk->getLatchSize(); i < n; ++i) {
        S = S.exist(ntk->getLatch(i).id+ntk->getLatchSize());
      }

      //Step3: cout
      std::stringstream ss;
      ss << i << ": " << V.getCube().toString()[1];
      output.push_back(ss.str());
    }
    reverse(output.begin(), output.end());
    for(int j=0; j<output.size(); ++j)
      cout << output[j] << endl;

  }
  else if(_isFixed)
    cout << "Monitor \"" << name << "\" is safe." << endl;
  else
    cout << "Monitor \"" << name << "\" is safe up to time " << _reachStates.size()-1 << "." << endl;
}
```

原則上, 就是算(monitor &Rn).countCube(). 如果是 0, 那很容易解決, 就直接 cout 做結尾. 否則, 則需要給出 counter-example, 也就是算 sn(X), Vn(I). 但就是按照推導公式算. 沒有困難!

**(ii) The assertions you add and their meanings**

在 hw1 的 vending-machine.v 當中，我設定了三個 monitors:

```
assign p = initialized && (serviceTypeOut == `SERVICE_OFF) && (itemTypeOut == `ITEM_NONE) && (outExchange != inputValue); // catch the bug
assign q = initialized && (serviceTypeOut == `SERVICE_OFF) && (inputValue != 8'd0);
assign r = initialized && (serviceTypeOut == `SERVICE_OFF) &&
          (inputValue != (outExchange + ((itemTypeOut==`ITEM_A)*`COST_A+(itemTypeOut==`ITEM_B)*`COST_B+(itemTypeOut==`ITEM_C)*`COST_C)));
```

第一個 p，是沿用原先的 setting，用來檢查機器運算結束時，如果沒有給出東西，也就是交易失敗的狀況，那退還的零錢金額是否正確．這也是作業一修正目標.

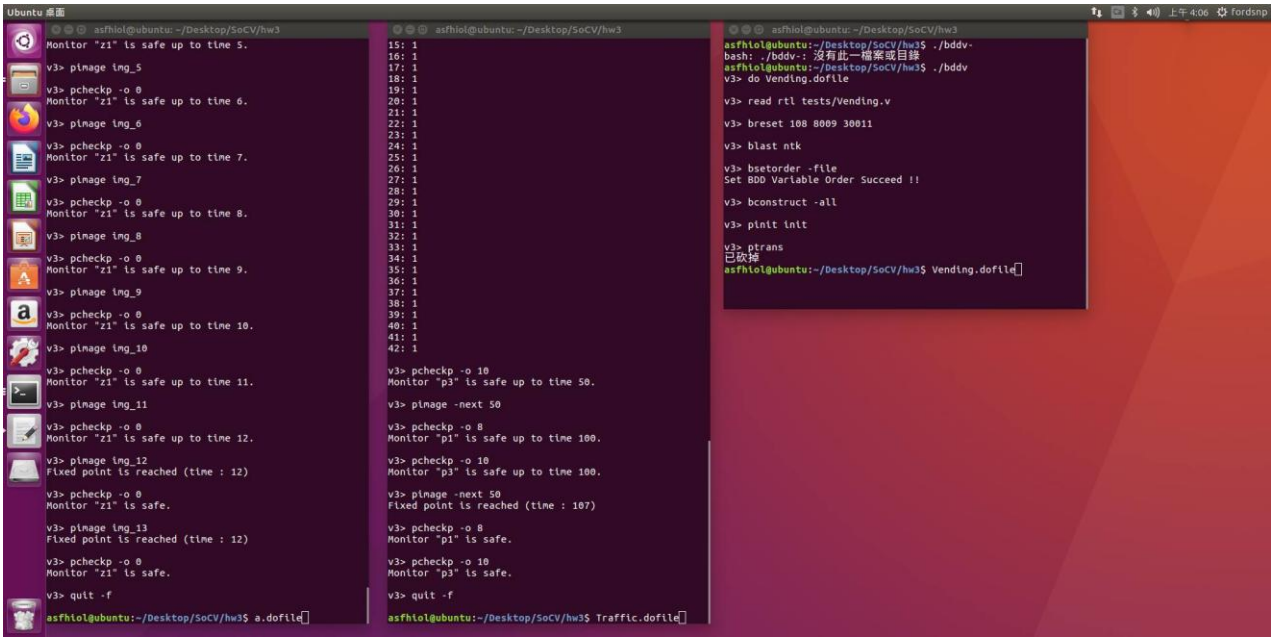
第二個 q，則是單純的檢查，當機器運算結束時，輸入金額一定不能是 0 元．算是一個初步且基礎的檢查.

第三個 r，則是更詳盡的檢查，每次機器運算結束時，計算輸出的商品價值+退換零錢金額，是否等同於輸入的零錢金額.

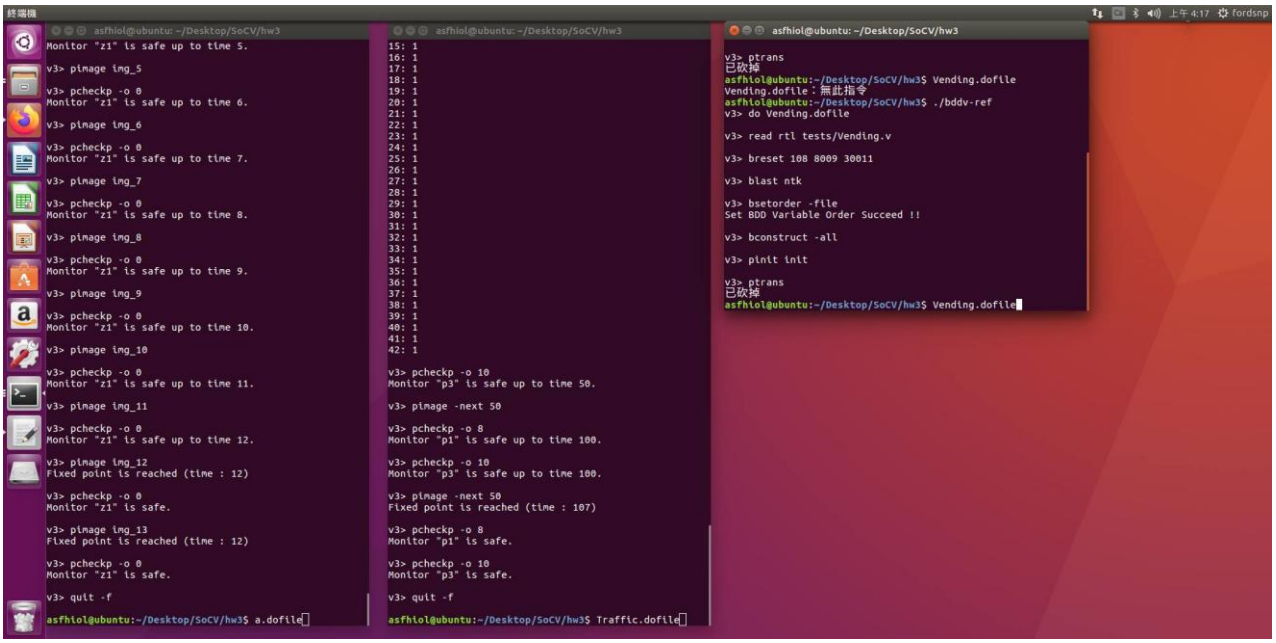
**(iii) Your verification results**

**(iv) The comparison with the ref program**

我的結果:




Ref 的結果:




兩邊對於 a.v, Traffic.v, Vending.v 的驗證結果，是一模一樣的. 對於 a.dofile, monitor z1 在第 12 步時，達到 Reachibility 的極限，也驗證了沒有反例存在. 對於 Traffic.dofile, p1 及 p3 最後都是沒有反例的，而 p2 則有一個反例在第 42 步時. 對於 Vending.v，由於無法建立 TR 或 TRI，所以也無從驗證 monitor p,q,r 的正確性與否.

**(v) The advanced techniques and/or abstraction of the design**

```
void
BddMgrV::buildPImageStep()
{
  //Step1: S(Y,X) = TR(Y,X) ^ Rn(X)
  V3Ntk* ntk = v3Handler.getCurHandler()->getNtk();
  BddNodeV S = getPTr() & getPReachState();
#ifdef FrontierSimplification
  if(_reachStates.size()>=2)
    S = getPTr() & Restrict(getPReachState(), ~(_reachStates[_reachStates.size()-2]));
#endif

  //step2: S(Y) = existX S(Y,X)
  for(unsigned i = 0, n = ntk->getLatchSize(); i < n; ++i) {
    S = S.exist(ntk->getLatch(i).id);
  }

  //step3: S(Y) -> S(X)
  bool isMoved;
  V3NetId id = ntk->getLatch(0);
  S = S.nodeMove(id.id+ntk->getLatchSize(), id.id, isMoved);

  //step4: Rn+1(X) = Rn(X)|S(X), set _isFixed
  BddNodeV oldR = getPReachState();
  if((oldR|S)==oldR){
    _isFixed = true;
  }
  else
    _reachStates.push_back((oldR|S));
}
```

首先，我第一個想到的優化部分，是 Step1 的部分．由於 Rn-1(X)對於 Sn(X)是一個 don't care region．因此，我們可以用 restrict(Sn, ~Rn-1)來取代 Sn，加速 BddNode 的計算流程．這部分參考自 Ric 老師去年在社團內的講解.

實驗結果:

| | Experiment #1 | Experiment #2 | Experiment #3 |
|---|---|---|---|
| a.dofile<br>without accelerate | user 0m2.000s<br>sys 0m0.009s | user 0m2.017s<br>sys 0m0.028s | user 0m2.010s<br>sys 0m0.012s |
| a.dofile<br>with accelerate | user 0m2.589s<br>sys 0m0.048s | user 0m2.620s<br>sys 0m0.020s | user 0m2.655s<br>sys 0m0.031s |
| Traffic.dofile<br>without accelerate | user 0m0.462s<br>sys 0m0.000s | user 0m0.464s<br>sys 0m0.012s | user 0m0.479s<br>sys 0m0.004s |
| Traffic.dofile<br>with accelerate | user 0m0.616s<br>sys 0m0.006s | user 0m0.573s<br>sys 0m0.004s | user 0m0.584s<br>sys 0m0.016s |

然而，令人驚訝的，使用 restrict-加速方式的 runtime，反而增加了約 29~33%不等. 考慮到 restrict 函數的實作層面，相當的單一且簡單，應該不會有寫壞的可能性，因此我們做出這個推論:

" restrict-加速方式可能會花費不少時間在實作遞迴函數呼叫，計算 restrict；然而 restrict(Sn, ~Rn-1)又沒佔到太多便宜，因此反而拖垮整體效率. "

此外，就算 restrict-加速方式真的能更快算出 BddNode，但這也是在建立完 TR 以後的事情．而我們的 Vending.v 卻是連 TR,TRI 都無法建立．因此無濟於事.

在其他方面，我們也試著做一些 minor-level 的優化，像是合併 X,Y 的 for-loop，或者是在 for-loop 外宣告重複使用的 variable，來減少 function call 等等，然而這些的成效都不彰. 因此，最終我們仍然無法驗證 Vending.v 的正確性!