Assignment-4

Yulei Sui

University of Technology Sydney, Australia

Assignment 4: Quiz + A Coding Task

- One quiz (10 points)
 - Static symbolic execution
 - Automatic translation from code to Z3 formulas/constraints

Assignment 4: Quiz + A Coding Task

- One guiz (10 points)
 - Static symbolic execution
 - Automatic translation from code to 73 formulas/constraints.
- One coding task (15 points)
 - Goal: automatically perform assertion-based verification for code using static symbolic execution.
 - Specification and code template: https://github.com/SVF-tools/ Teaching-Software-Verification/tree/main/Assignment-3
 - SVF Z3 APIs: https: //github.com/SVF-tools/Teaching-Software-Verification/wiki/Z3-API

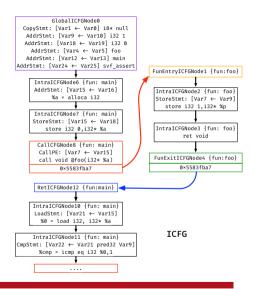
You are encouraged to finish the guizzes before starting your coding task.

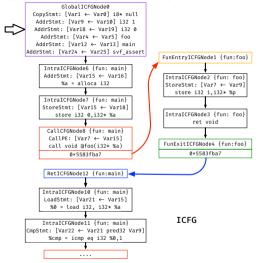
To do

You need to finish:

```
void foo(int* p) {
   *p = 1:
 int main() {
     int a = 0:
     foo(&a);
     svf_assert(a == 1);
           compile
void @foo(i32* %p) {
entry:
  store i32 1, i32* %p
  ret void
i32 @main() {
entry:
  %a = alloca i32
  store i32 0. i32* %a
  call void @foo(i32* %a)
  %0 = load i32, i32* %a
  %cmp = icmp eq i32 %0.1
  call void @svf_assert(i1 zeroext %cmp)
  ret i32 0
```

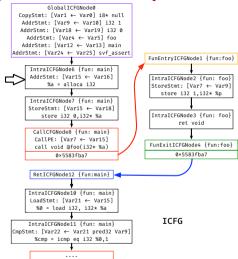
```
void foo(int* p) {
   *p = 1:
 int main() {
     int a = 0:
     foo(&a):
     svf_assert(a == 1);
            compile
void @foo(i32* %p) {
                                      SVF
entry:
  store i32 1. i32* %p
  ret void
i32 @main() {
entry:
  %a = alloca i32
  store i32 0. i32* %a
  call void @foo(i32* %a)
  %0 = load i32, i32* %a
  %cmp = icmp eq i32 %0.1
  call void @svf_assert(i1 zeroext %cmp)
  ret i32 0
```



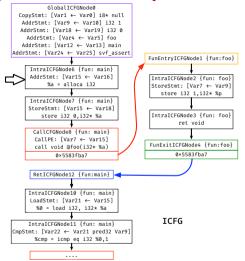


```
-----SVFVar and Value-----
ObiVar25 (0x7f000019)
                        Value: NIII.I.
ObiVar19 (0x7f000013)
                        Value: 0
ObiVar16 (0x7f000010)
                        Value: NULL
ObiVar13 (0x7f00000d)
                        Value: NULL
ObiVar10 (0x7f00000a)
                        Value: 1
ObiVar5 (0x7f000005)
                        Value: NIII.I.
ValVar24
                        Value: 0x7f000019
ObiVar2 (0x7f000002)
                        Value: NULL
ObjVar3 (0x7f000003)
                        Value: NULL
Val Var1
                        Value: 2
ValVar0
                        Value: 2
ValVar4
                        Value: 0x7f000005
ValVar9
                        Value: 1
ValVar12
                        Value: 0x7f00000d
ValVar18
                        Value: 0
```

The values of Z3 expressions for each SVFVar after analyzing GlobalICFGNode0 (use printExprValues() to print SVFVars and their Values)



```
Algorithm 2 handleIntra(intraEdge)
1 if intraEdge.getCondition() && !handleBranch(intraEdge)
  then
      return false
3 else
      handleNonBranch(edge)
  HandleNonBranch(intraEdge)
    dst ← intraEdge.getDstNode(): src ← intraEdge.getSrcNode()
    foreach stmt ∈ dst.getSVFStmts() do
     if addr ∈ dvn_cast(AddrStmt)(stmt) then
        obi ← getMemObiAddress(addr.getRHSVarID())
        lhs ← getZ3Expr(addr.getLHSVarID())
        addToSolver(obi == lhs)
     else if copy ∈ dyn_cast(CopyStmt)(stmt) then
         lhs ← getZ3Expr(copy.getLHSVarID())
        rhs ← getZ3Expr(copv.getRHSVarID())
10
        addToSolver(rhs == 1hs)
     else if load ∈ dvn_cast(LoadStmt)(stmt) then
11
        lhs ← getZ3Expr(load.getLHSVarID())
12
        rhs ← getZ3Expr(load.getRHSVarID())
13
        addToSolver(lhs == z3Mgr.loadValue(rhs))
14
```

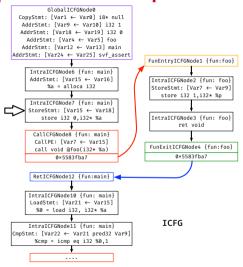


```
-----SVEVar and Value-----
ObjVar25 (0x7f000019)
                        TIIIM . arr [eV
ObiVar19 (0x7f000013)
                        Value: 0
ObiVar16 (0x7f000010)
                        Value: NIII.I.
ObjVar13 (0x7f00000d)
                        Value: NIII.I.
ObiVar10 (0x7f00000a)
                        Value: 1
ObiVar5 (0x7f000005)
                        Value: NUI.I.
ValVar24
                        Value: 0x7f000019
ObiVar2 (0x7f000002)
                        Value: NULL
ObjVar3 (0x7f000003)
                        Value: NULL
ValVar1
                        Value: 2
ValVar0
                        Value: 2
ValVar4
                        Value: 0x7f000005
ValVar9
                        Value: 1
ValVar12
                        Value: 0x7f00000d
ValVar18
                        Value: 0
+ValVar15
                       Value: 0v7f000010
```

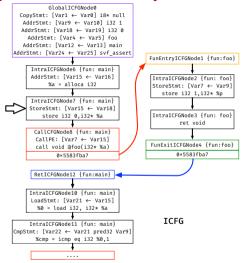
Analyzing IntralCFGNode6 {fun: main}

AddrStmt: [Var14 ← Var15]

%a = alloca i32



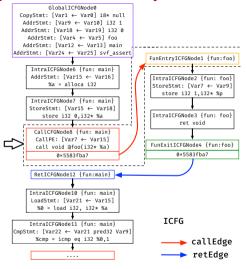
```
Algorithm 2 handleIntra(intraEdge)
1 if intraEdge.getCondition() && !handleBranch(intraEdge)
   then
      return false
3 else
      handleNonBranch(edge)
  HandleNonBranch(intraEdge)
    dst ← intraEdge.getDstNode(); src ← intraEdge.getSrcNode()
    foreach stmt ∈ dst.getSVFStmts() do
     else if load ∈ dyn_cast(LoadStmt)(stmt) then
11
         lhs ← getZ3Expr(load.getLHSVarID())
12
         rhs ← getZ3Expr(load.getRHSVarID())
13
         addToSolver(lhs == z3Mgr.loadValue(rhs))
14
15
     else if store ∈ dyn_cast(StoreStmt)(stmt) then
         lhs ← getZ3Expr(store.getLHSVarID())
16
         rhs ← getZ3Expr(store.getRHSVarID())
17
         z3Mgr.storeValue(lhs.rhs)
18
     else if gep ∈ dvn_cast(GepStmt)(stmt) then
19
         lhs ← getZ3Expr(gep.getLHSVarID())
20
         rhs ← getZ3Expr(gep.getRHSVarID())
21
         offset = z3Mgr.getGepOffset(gep)
22
         gepAddress = z3Mgr.getGepObjAddress(rhs, offset)
23
         addToSolver(lhs == gepAddress)
24
```



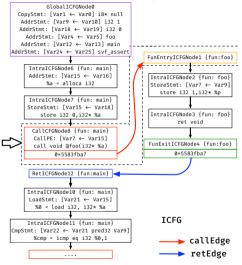
```
-----SVEVar and Value-----
ObjVar25 (0x7f000019)
                        TIIIM . arr [eV
ObiVar19 (0x7f000013)
                        Value: 0
ObiVar16 (0x7f000010)
                        Value: 0
ObjVar13 (0x7f00000d)
                        Value: NIII.I.
ObiVar10 (0x7f00000a)
                        Value: 1
ObiVar5 (0x7f000005)
                        Value: NUI.I.
ValVar24
                        Value: 0x7f000019
ObiVar2 (0x7f000002)
                        Value: NULL
ValVar15
                       Value: 0x7f000010
ObiVar3 (0x7f000003)
                        Value: NIII.I.
Val Var1
                        Value: 2
ValVar0
                        Value: 2
ValVar4
                        Value: 0x7f000005
ValVar9
                        Value: 1
ValVar12
                        Value: 0v7f00000d
+ValVar18
                       Value: 0
```

```
## Analyzing IntralCFGNode6 {fun: main}
```

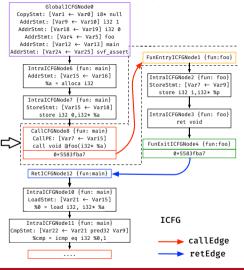
StoreStmt: [Var15
$$\leftarrow$$
 Var18] store i32 0, i32 * %a



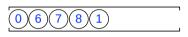
```
Algorithm 1 Context sensitive control-flow reachability
  Input: src: ICFGNode dst: ICFGNode
         path: vector(ICFGNode) visited: set(ICFGNode):
1 dfs(path, src, dst)
    visited.insert(src)
    path.push_back(src)
    if erc -- det then
     print path
    foreach edge ∈ src.getOutEdges() do
     if edge.dst ∉ visited then
         if edge.isIntraCFGEdge() then
            if handleIntra(edge) then
                dfs(path, edge.dst, dst)
         else if edge.isCallCFGEdge() then
11
             if handleCall(edge) then
12
                dfs(path, edge.dst, dst)
         else if edge.isRetCFGEdge() then
14
             if handleRet(edge) then
                dfs(path.edge.dst.dst)
    visited.erase(src)
    path.pop_back(src)
```



```
Algorithm 3 handleCall(callEdge)
   callNode ← callEdge.getSrcNode();
   FunEntryNode ←callEdge.getDstNode();
   callstack.push back(callNode):
   getSolver().push():
5 foreach callPE ∈ calledge.getCallPEs() do
     lhs ← getZ3Expr(callPE.getLHSVarID()):
     rhs ← getZ3Expr(callPE.getRHSVarID());
     addToSolver(lhs == rhs):
9 return true:
```





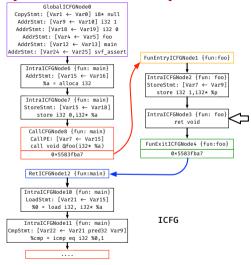


path (a sequence of ICFGNodes represented by their IDs)

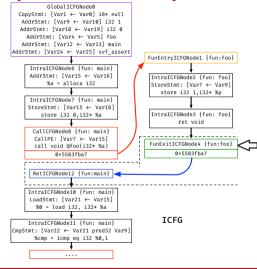
State of callstack after processing call edge between CallCFGNode8 and FunEntryICFGNode1

```
Global TCEGNodos
  ConvStmt · [Var1 ← Var0] i8* null
   AddrStmt: [Var9 ← Var10] i32 1
  AddrStmt · [Var18 ← Var19] i32 0
    AddrStmt: [Var4 ← Var5] foo
   AddrStmt: [Var12 ← Var13] main
AddrStmt: [Var24 ← Var25] syf assert
                                          FunEntrvICFGNode1 {fun:foo}
     IntraICEGNode6 (fun: main)
     AddrStmt: [Var15 ← Var16]
                                           IntraICFGNode2 {fun: foo}
           %a = alloca i32
                                           StoreStmt: [Var7 ← Var9] <
                                              store i32 1.i32* %p
     IntraICFGNode7 {fun: main}
     StoreStmt: [Var15 ← Var18]
         store i32 0.i32* %a
                                           IntraICFGNode3 {fun: foo}
                                                    ret void
       CallCEGNodes (fun: main)
       CallPF: [Var7 ← Var15]
                                           FunExitICFGNode4 {fun:foo}
       call void @foo(i32* %a)
             0×5583fha7
                                                   0×5583fba7
      RetICFGNode12 {fun:main}
     IntraICEGNode10 (fun: main)
     LoadStmt: [Var21 ← Var15]
       %0 = load i32. i32* %a
                                              ICFG
     IntraICEGNodell (fun: main)
CmpStmt: [Var22 ← Var21 pred32 Var9]
       %cmp = icmp eq i32 %0.1
```

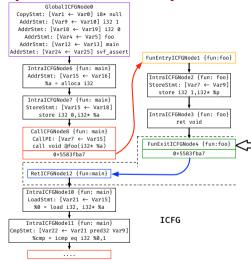
```
Algorithm 2 handleIntra(intraEdge)
1 if intraEdge.getCondition() && !handleBranch(intraEdge)
   then
      return false
3 else
      handleNonBranch(edge)
  HandleNonBranch(intraEdge)
    dst ← intraEdge.getDstNode(); src ← intraEdge.getSrcNode()
    foreach stmt ∈ dst.getSVFStmts() do
     else if load ∈ dyn_cast(LoadStmt)(stmt) then
11
         lhs ← getZ3Expr(load.getLHSVarID())
12
         rhs ← getZ3Expr(load.getRHSVarID())
13
         addToSolver(lhs == z3Mgr.loadValue(rhs))
14
15
     else if store ∈ dyn_cast(StoreStmt)(stmt) then
         lhs ← getZ3Expr(store.getLHSVarID())
16
         rhs ← getZ3Expr(store.getRHSVarID())
17
         z3Mgr.storeValue(lhs.rhs)
18
     else if gep ∈ dvn_cast(GepStmt)(stmt) then
19
         lhs ← getZ3Expr(gep.getLHSVarID())
20
         rhs ← getZ3Expr(gep.getRHSVarID())
21
         offset = z3Mgr.getGepOffset(gep)
22
         gepAddress = z3Mgr.getGepObjAddress(rhs, offset)
23
         addToSolver(lhs == gepAddress)
```



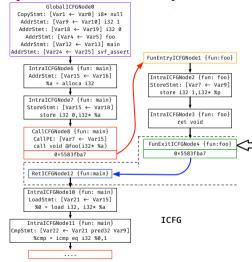
ret void instruction. Nothing needs to be done. Continue



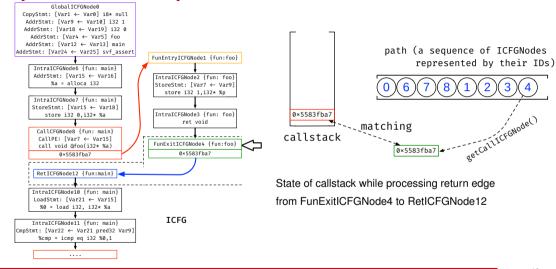
```
Algorithm 1 Context sensitive control-flow reachability
  Input: src: ICFGNode dst: ICFGNode
         path: vector(ICFGNode) visited: set(ICFGNode):
1 dfs(path.src.dst)
    visited insert(src)
    path.push_back(src)
    if erc -- det then
     print path
    foreach edge ∈ src.getOutEdges() do
     if edge.dst ∉ visited then
         if edge.isIntraCFGEdge() then
             if handleIntra(edge) then
10
                dfs(path,edge.dst,dst)
         else if edge.isCallCFGEdge() then
11
             if handleCall(edge) then
                dfs(path,edge.dst,dst)
         else if edge.isRetCFGEdge() then
14
             if handleRet(edge) then
                dfs(path, edge.dst, dst)
17
    visited.erase(src)
    path.pop_back(src)
```

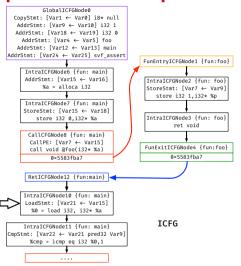


```
Algorithm 4 handleRet(retEdge)
    retNode ← retEdge.getDstNode():
    rhs(getCtx()):
    lhs(getCtx()):
    if retPE = retEdge.getRetPE() then
     rhs ← getEvalExpr(getZ3Expr(retPE.getRHSVarID()));
     lhs ← getZ3Expr(retPE.getLHSVarID());
    if callstack \neq \emptyset then
      if callstack.back() == getCallICFGNode(retNode) then
         callstack.pop_back():
         getSolver().pop():
     else
12
         return false:
    if retEdge.getRetPE() then
      addToSolver(lhs == rhs):
    return true:
```

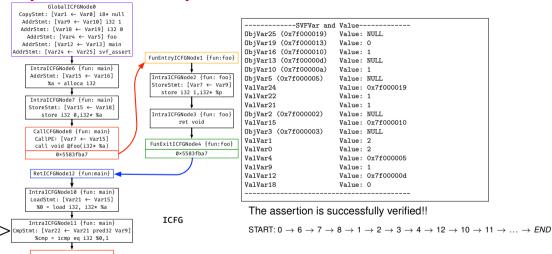


```
Algorithm 4 handleRet(retEdge)
    retNode ← retEdge.getDstNode():
    rhs(getCtx()):
    lhs(getCtx()):
    if retPE = retEdge.getRetPE() then
     rhs ← getEvalExpr(getZ3Expr(retPE.getRHSVarID()));
     lhs ← getZ3Expr(retPE.getLHSVarID());
    if callstack \neq \emptyset then
      if callstack.back() == getCallICFGNode(retNode) then
         callstack.pop_back():
         getSolver().pop();
     else
12
         return false:
    if retEdge.getRetPE() then
      addToSolver(lhs == rhs):
    return true:
```



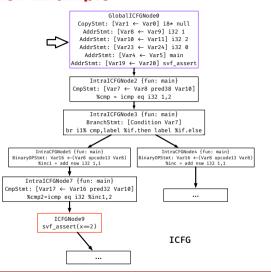


```
Algorithm 2 handleIntra(intraEdge)
1 if intraEdge.getCondition() && !handleBranch(intraEdge)
   then
      return false
3 else
      handleNonBranch(edge)
  HandleNonBranch(intraEdge)
    dst ← intraEdge.getDstNode(); src ← intraEdge.getSrcNode()
    foreach stmt ∈ dst.getSVFStmts() do
     else if load ∈ dyn cast(LoadStmt)(stmt) then
11
         lhs ← getZ3Expr(load.getLHSVarID())
12
         rhs ← getZ3Expr(load.getRHSVarID())
13
         addToSolver(lhs == z3Mgr.loadValue(rhs))
14
15
     else if store ∈ dyn_cast(StoreStmt)(stmt) then
         lhs ← getZ3Expr(store.getLHSVarID())
16
         rhs ← getZ3Expr(store.getRHSVarID())
17
         z3Mgr.storeValue(lhs.rhs)
18
     else if gep ∈ dvn_cast(GepStmt)(stmt) then
19
         lhs ← getZ3Expr(gep.getLHSVarID())
20
         rhs ← getZ3Expr(gep.getRHSVarID())
21
         offset = z3Mgr.getGepOffset(gep)
22
         gepAddress = z3Mgr.getGepObjAddress(rhs, offset)
23
         addToSolver(lhs == gepAddress)
```



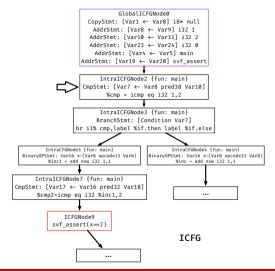
```
int main(){
       int x = 1, y = 1;
      int a = 1, b = 2;
       if (a > b) {
          y++;
       } else {
          X++:
          svf_assert (x == 2);
       return 0:
                 compile
i32 @main() {
entry:
 %cmp = icmp sat i32 1. 2
 br i1 %cmp, label %if.then, label %if.else
if then:
 %inc = add nsw i32 1, 1
 br label %if.end
if else:
 \%inc1 = add nsw i32 1. 1
 %cmp2 = icmp eq i32 %inc1, 2
 call void @svf_assert(i1 zeroext %cmp2)
 br label %if end
if end:
ret i32 0
```

```
Global TCEGNode0
    int main(){
                                                                                         CopyStmt: [Var1 ← Var0] i8* null
        int x = 1, y = 1;
                                                                                          AddrStmt: [Var8 ← Var9] i32 1
        int a = 1. b = 2:
                                                                                         AddrStmt: [Var10 ← Var11] i32 2
        if (a > b) {
                                                                                         AddrStmt: [Var23 ← Var24] i32 0
            V++:
                                                                                           AddrStmt: [Var4 ← Var5] main
        } else {
                                                                                       AddrStmt: [Var19 ← Var20] syf assert
            X++:
            syf assert (x == 2):
                                                                                             IntraICFGNode2 {fun: main}
                                                                                        CmpStmt: [Var7 ← Var8 pred38 Var10]
        return 0:
                                                                                              %cmp = icmp eq i32 1.2
                    compile
                                                                                             IntraICFGNode3 {fun: main}
                                                                                           BranchStmt: [Condition Var7]
i32 @main() -
                                                       SVF
                                                                                      br i1% cmp.label %if.then label %if.else
entry:
  %cmp = icmp sat i32 1. 2
                                                                              IntraCEGNode5 (fun: main)
                                                                                                                IntraCEGNode4 (fun: main)
  br i1 %cmp, label %if.then, label %if.else
                                                                                                         BinaryOPStmt: Var16 ←(Var8 opcode13 Var8)
                                                                        BinaryOPStmt: Var16 ←(Var8 opcode13 Var8)
                                                                              %inc1 = add nsw i32 1,1
                                                                                                                 %inc = add nsw i32 1.1
if then:
  %inc = add nsw i32 1, 1
                                                                           IntraICEGNode7 {fun: main}
  br label %if end
                                                                     CmpStmt: [Var17 ← Var16 pred32 Var10]
                                                                           %cmp2=icmp eq i32 %inc1.2
if else:
  \%inc1 = add nsw i32 1. 1
                                                                                   TCEGNode9
  %cmp2 = icmp eq i32 %inc1, 2
                                                                                svf assert(x=2)
  call void @svf_assert(i1 zeroext %cmp2)
                                                                                                                  TCFG
  br label %if end
if end:
ret i32 0
```



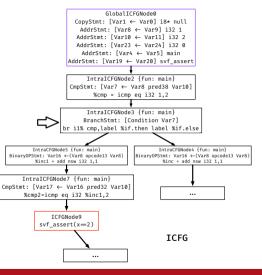
```
-----SVEVar and Value-----
ObiVar20 (0x7f000014)
                        Value: NIII.I.
ObiVar24 (0x7f000018)
                       Value: 0
ObjVar11 (0x7f00000b)
                       Value: 2
ObiVar9 (0x7f000009)
                        Value: 1
ObjVar5 (0x7f000005)
                       Value: NULL
ValVar19
                       Value: 0x7f000014
ValVar23
                       Value: 0
ObjVar2 (0x7f000002)
                       Value: NULL
ObiVar3 (0x7f000003)
                       Value: NIII.I.
ValVar1
                       Value: 3
ValVar0
                       Value: 3
ValVar4
                       Value: 0x7f000005
ValVar8
                       Value: 1
ValVar10
                        Value: 2
```

The values of Z3 expressions for each SVFVar after analyzing GlobalICFGNodeO



```
## Analyzing IntraICFGNode2 {fun: main}
CmpStmt: [Var7 <-- (Var8 predicate38 Var10)]
  %cmp = icmp sgt i32 1. 2
==> (not (<= ValVar8 ValVar10))
==> (= ValVar7 0)
```

Code for handling CmpStmt has been implemented in the HandleNonBranch() function.



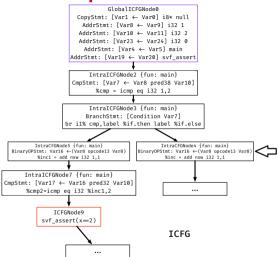
```
Algorithm 2 handleIntra(intraEdge)
```

- if intraEdge.getCondition() && !handleBranch(intraEdge) then return false 3 else handleNonBranch(edge)

return true

handleBranch(intraEdge)

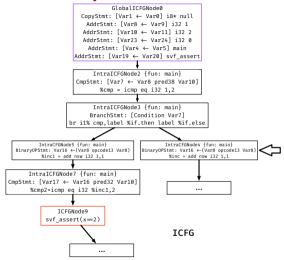
```
cond = intraEdge.getCondition()
    successorVal = intraEdge.getSuccessorCondValue()
    res = getEvalExpr(cond == suc)
  if res.is_false() then
      addToSolver(cond! = suc)
     return false
7 else if res.is_true() then
      addToSolver(cond == suc)
      return true
10 else
```



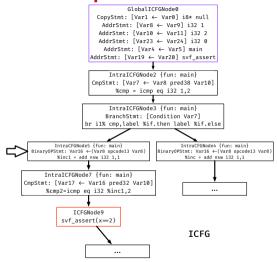
```
-----SVFVar and Value-----
ObiVar20 (0x7f000014)
                        Value: NIII.I.
ObjVar24 (0x7f000018)
                        Value: 0
ObiVar11 (0x7f00000b)
                        Value: 2
ObiVar9 (0x7f000009)
                        Value: 1
ObiVar5 (0x7f000005)
                        Value: NULL
ValVar19
                        Value: 0x7f000014
ValVar23
                        Value: 0
ObiVar2 (0x7f000002)
                        Value: NIII.I.
ObiVar3 (0x7f000003)
                        Value: NIII.I.
ValVar1
                        Value: 3
ValVar0
                        Value: 3
ValVar4
                        Value: 0x7f000005
ValVar8
                        Value: 1
ValVar10
                        Value: 2
ValVar7
                        Value: 0
```

```
Branch IntraCFGEdge: [ICFGNode4 ← ICFGNode3] branchCondition: %cmp = icmp sgt i32 1, 2
```

This conditional ICFGEdge is **infeasible**!!

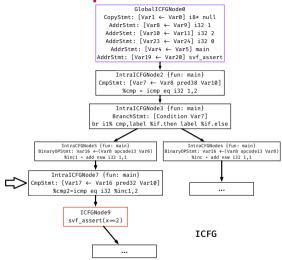


```
Algorithm 1 Context sensitive control-flow reachability
  Input: src: ICFGNode dst: ICFGNode
         path: vector(ICFGNode) visited: set(ICFGNode):
 dfs(path.src.dst)
    visited.insert(src)
    path.push_back(src)
    if src == dst then
     print path
    foreach edge ∈ src.getOutEdges() do
     if edge.dst ∉ visited then
         if edge.isIntraCFGEdge() then
             if handleIntra(edge) then
10
                dfs(path, edge.dst, dst)
         else if edge.isCallCFGEdge() then
11
             if handleCall(edge) then
13
                dfs(path, edge.dst, dst)
         else if edge.isRetCFGEdge() then
14
             if handleRet(edge) then
15
                dfs(path, edge.dst, dst)
16
    visited.erase(src)
    path.pop_back(src)
```



```
-----SVFVar and Value-----
ObiVar20 (0x7f000014)
                        Value: NIII.I.
ObjVar24 (0x7f000018)
                        Value: 0
ObiVar11 (0x7f00000b)
                        Value: 2
ObiVar9 (0x7f000009)
                        Value: 1
ObiVar5 (0x7f000005)
                        Value: NULL
ValVar19
                        Value: 0x7f000014
ValVar23
                        Value: 0
ObiVar2 (0x7f000002)
                        Value: NIII.I.
ObiVar3 (0x7f000003)
                        Value: NIII.I.
ValVar1
                        Value: 3
OreViev
                        Value: 3
ValVar4
                        Value: 0x7f000005
ValVar8
                        Value: 1
ValVar10
                        Value: 2
ValVar7
                        Value: 0
```

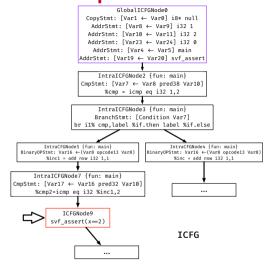
```
Branch IntraCFGEdge: [ICFGNode5 ← ICFGNode3]
branchCondition: %cmp = icmp sgt i32 1, 2
(= ValVar7 0)
This conditional ICFGEdge is feasible!!
```



```
-----SVFVar and Value-----
                        Value: NULL
ObiVar20 (0x7f000014)
ObiVar24 (0x7f000018)
                         Value: 0
ObiVar11 (0x7f00000b)
                        Value: 2
ObiVar9 (0x7f000009)
                        Value: 1
ObiVar5 (0x7f000005)
                        Value: NIII.I.
ValVar19
                        Value: 0x7f000014
ValVar23
                        Value: 0
ValVar17
                        Value: 1
ObiVar2 (0x7f000002)
                        Value: NIII.I.
ObjVar3 (0x7f000003)
                        Value: NIII.I.
ValVar16
                        Value: 2
ValVar1
                        Value: 3
ValVar0
                        Value: 3
ValVar4
                        Value: 0x7f000005
ValVar8
                        Value: 1
ValVar10
                        Value: 2
ValVar7
                        Value: 0
```

Analyzing IntralCFGNode7 fun: main

CmpStmt: [Var17 ← (Var16 predicate32 Var10)]



The assertion is successfully verified!!

START: 0
$$\rightarrow$$
 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow \textit{END}