Code Verification Using Symbolic Execution

(Week 7)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Example 4: Branches

```
void main(int x){
    int y;
    if(x > 10) {
        y = x + 1;
    }
    else {
        y = 10;
    }
    svf_assert(y >= x + 1);
}
```

Source code

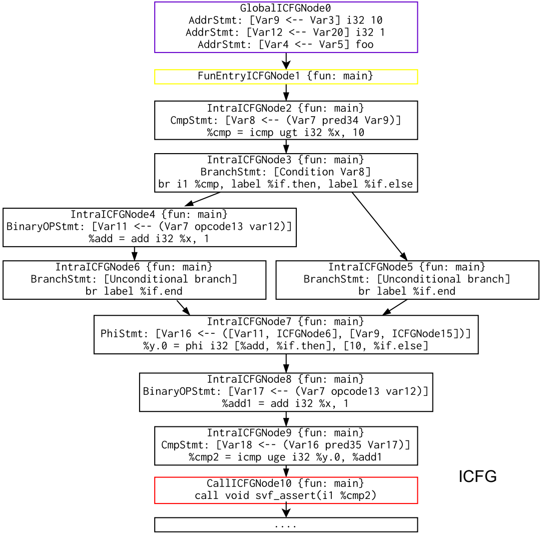
```
define void @main(i32 %x) #0 {
entry:
  %cmp = icmp ugt i32 %x, 10
  br i1 %cmp, label %if.then, label %if.else

if.then:
  %add = add i32 %x, 1
  br label %if.end

if.else:
  br label %if.end

if.end: = %if.else, %if.then
  %y.0 = phi i32 [%add, %if.then], [10, %if.else]
  %add1 = add i32 %x, 1
  %cmp2 = icmp uge i32 %y.0, %add1
  call void @svf_assert(i1 zeroext %cmp2)
  ret void
}
```

LLVM IR

Example 4: Branches

```
define void @main(i32 %x) #0 {
entry:
  %cmp = icmp ugt i32 %x, 10
  br i1 %cmp, label %if.then, label %if.else

if.then:
  %add = add i32 %x, 1
  br label %if.end

if.else:
  br label %if.end

if.end: = %if.else, %if.then
  %y.0 = phi i32 [%add, %if.then], [10, %if.else]
  %add1 = add i32 %x, 1
  %cmp2 = icmp uge i32 %y.0, %add1
  call void @svf_assert(i1 zeroext %cmp2)
  ret void
}
```

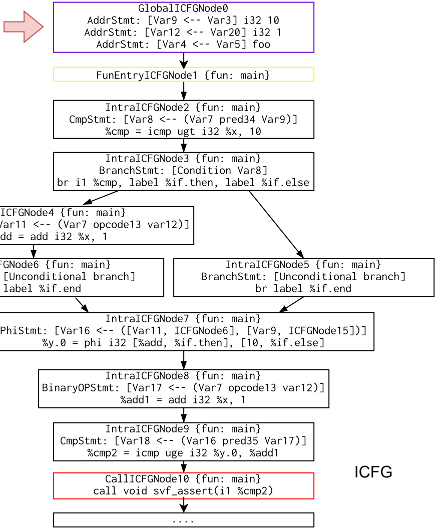
LLVM IR



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

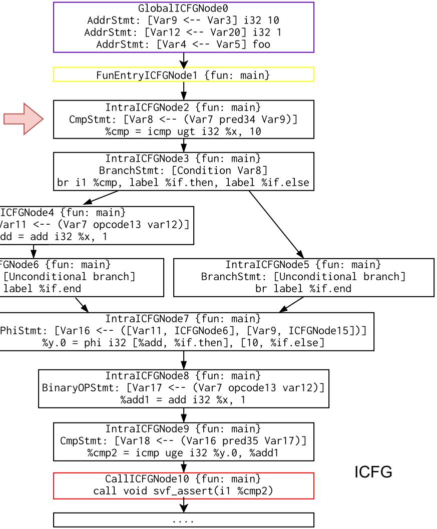
ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)    Value: NULL
ValVar4                 Value: 0x7f000005
ValVar9                 Value: 10
ValVar12                Value: 1
                   ...
------------------------------------------
```

The values of Z3 expressions for each SVFVar after analyzing `GlobalICFGNode0`

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

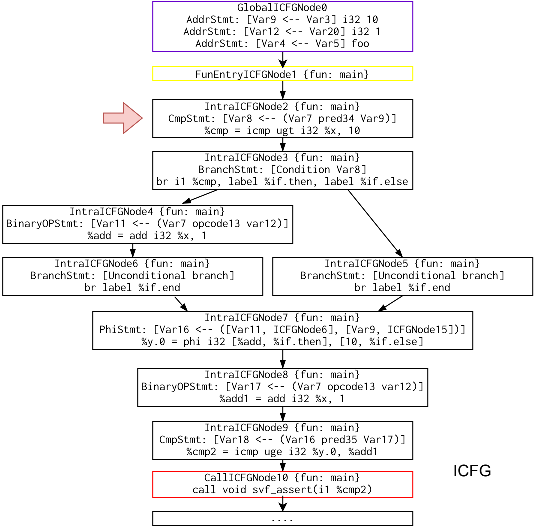
ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

```
## Analyzing IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 predicate34 Var9)]
   %cmp = icmp ugt i32 %x, 10
==> (not (<= ValVar7 ValVar9))
==> (= ValVar8 1)
                      ...
```

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

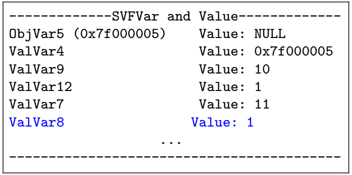
Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)



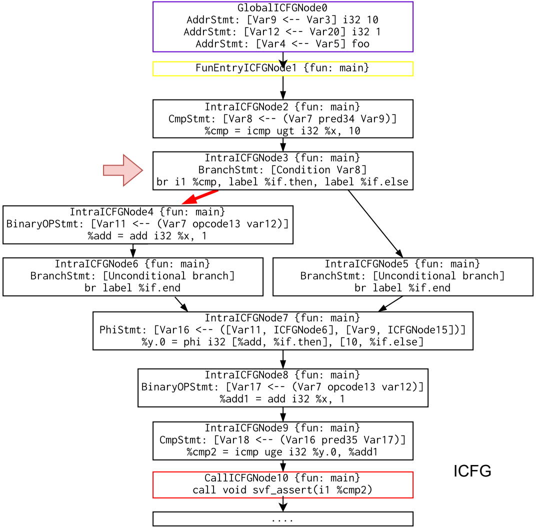
```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 11
ValVar8                  Value: 1
                   ...
------------------------------------------
```

Analyzing IntraICFGNode2 {fun: main}

CmpStmt: [Var8 ← (Var7 pred34 Var9)]

%cmp = icmp ugt i32 %x, 10

Example 4: Branches



GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....

ICFG

Algorithm 1: 3 handleIntra(intraEdge)

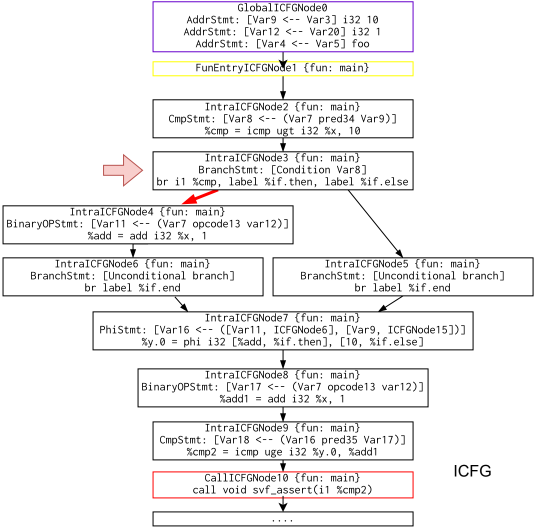
```
if intraEdge.getCondition() && !handleBranch(intraEdge) then
    return false;
else
    handleNonBranch(edge);
```

Algorithm 2: handleBranch(intraEdge)

```
cond = intraEdge.getCondition();
succ = intraEdge.getSuccessorCondValue();
getSolver().push();
addToSolver(cond == succ);
res = getSolver().check();
getSolver().pop();
if res == unsat then
    return false;
else
    addToSolver(cond == succ);
    return true;
```

Note: getSolver().push() creates a new stack frame for maintaining the newly added Z3 constraints.

Example 4: Branches



GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....

ICFG

Algorithm 3: 3 handleIntra(intraEdge)

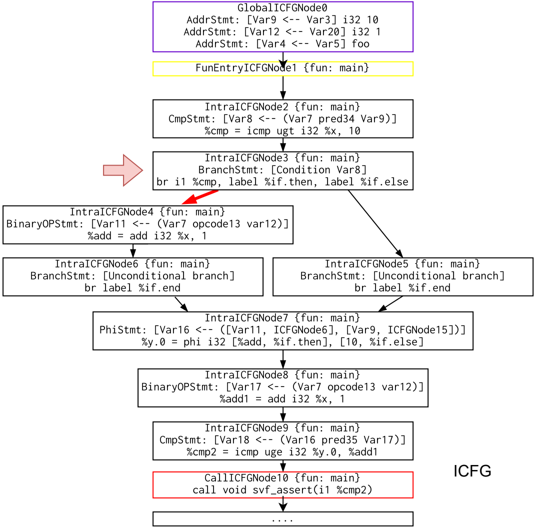
```
if intraEdge.getCondition() && !handleBranch(intraEdge) then
    return false;
else
    handleNonBranch(edge);
```

Algorithm 4: handleBranch(intraEdge)

```
cond = intraEdge.getCondition();
succ = intraEdge.getSuccessorCondValue();
getSolver().push();
addToSolver(cond == succ);
res = getSolver().check();
getSolver().pop();
if res == unsat then
    return false;
else
    addToSolver(cond == succ);
    return true;
```

Note: `getSolver().pop()` pops out the top stack frame, which contains the Z3 constraint `cond == succ`.

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

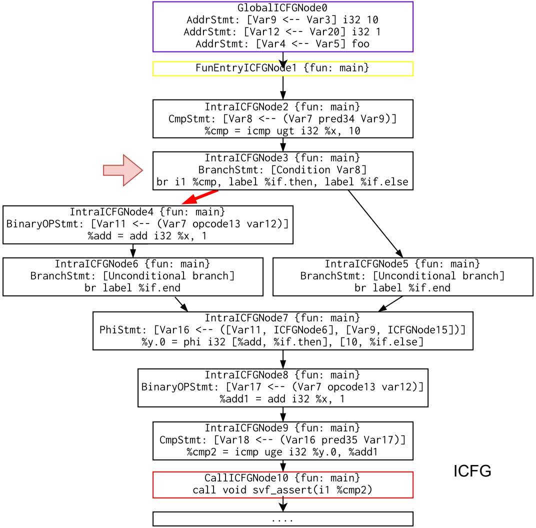
Algorithm 5: 3 `handleIntra(intraEdge)`

```
if intraEdge.getCondition() && !handleBranch(intraEdge) then
    return false;
else
    handleNonBranch(edge);
```

Algorithm 6: `handleBranch(intraEdge)`

```
cond = intraEdge.getCondition();
succ = intraEdge.getSuccessorCondValue();
getSolver().push();
addToSolver(cond == succ);
res = getSolver().check();
getSolver().pop();
if res == unsat then
    return false;
else
    addToSolver(cond == succ);
    return true;
```

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 11
ValVar8                  Value: 1
                   ...
------------------------------------------
```

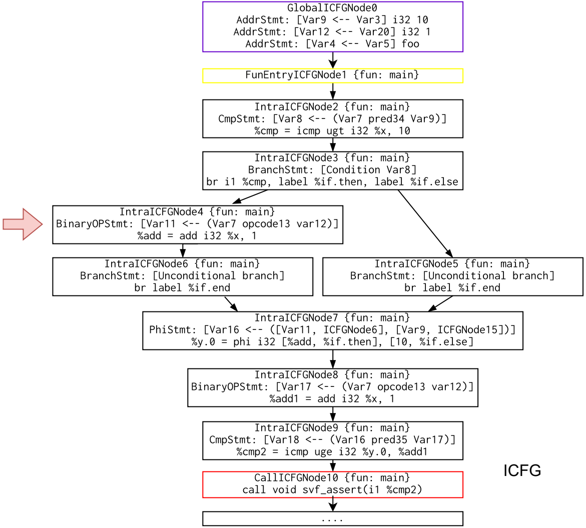
Branch IntraCFGEdge: [`ICFGNode4` ← `ICFGNode3`]

branchCondition: `%cmp` = `icmp ugt i32 %x, 10`

(= `ValVar8 1`)

This conditional ICFGEdge is **feasible**!!

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

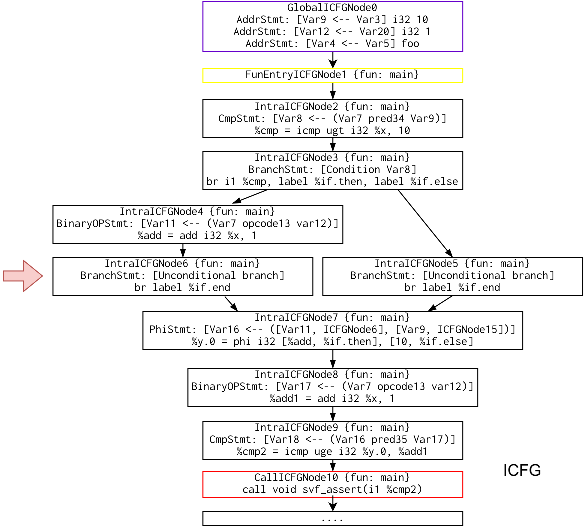
```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 11
ValVar8                  Value: 1
ValVar11                Value: 12
                    ...
------------------------------------------
```

Analyzing IntraICFGNode4 {fun: main}

BinaryOPStmt: [Var11 ← (Var7 opcode13 var12)]

`%add = add i32 %x, 1`

Example 4: Branches



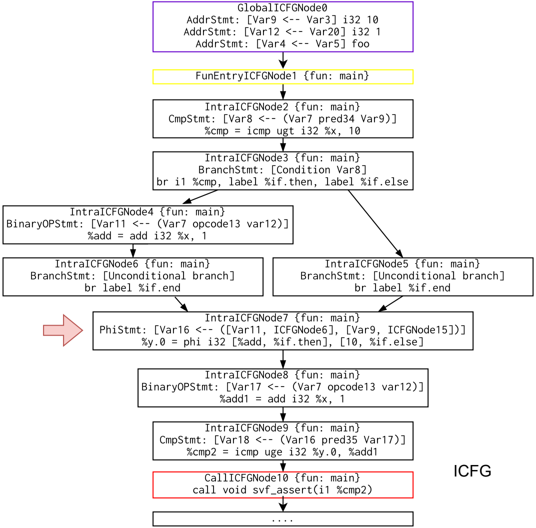
```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Algorithm 7: 3 handleIntra(intraEdge)

```
if intraEdge.getCondition() && !handleBranch(intraEdge) then
    return false;
else
    handleNonBranch(edge);
```

Example 4: Branches



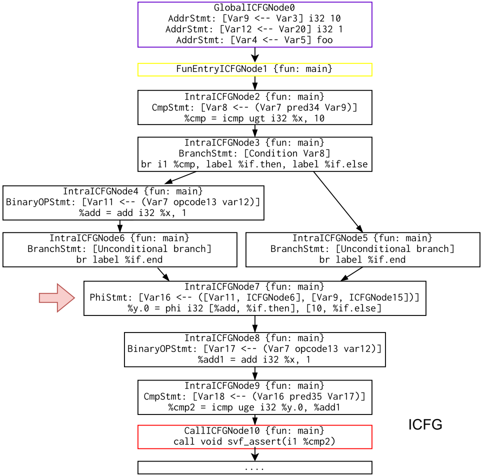
```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Algorithm 8: 3 handlePhi(intraEdge)

```
res ← getZ3Expr(phi.getResID());
opINodeFound ← false;
for i ← 0 to phi.getOpVarNum() − 1 do
    if srcNode.getFun().postDominate(srcNode.getBB(),
    phi.getOpICFGNode(i).getBB()) then
        ope ← getZ3Expr(phi.getOpVar(i).getId());
        addToSolver(res == ope);
        opINodeFound ← true;
```

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

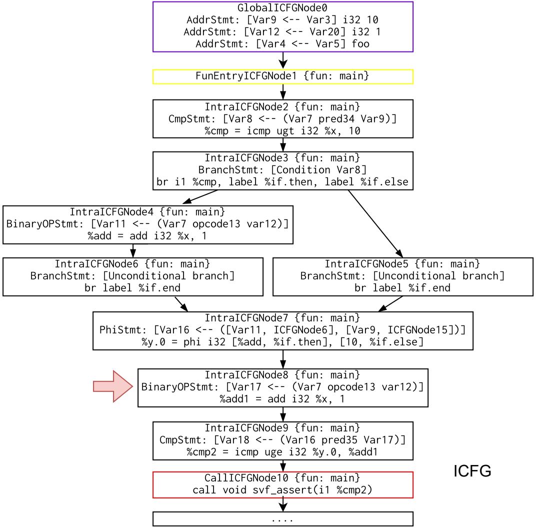
```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)   Value: NULL
ValVar4                Value: 0x7f000005
ValVar9                Value: 10
ValVar12               Value: 1
ValVar7                Value: 11
ValVar8                Value: 1
ValVar11               Value: 12
ValVar16               Value: 12
                          ...
------------------------------------------
```

Analyzing IntraICFGNode7 {fun: main}

PhiStmt: [Var16 ← ([Var11, ICFGNode6], [Var9, ICFGNode15])]

`%y.0 = phi i32 [%add, %if.then], [10, %if.else]`

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

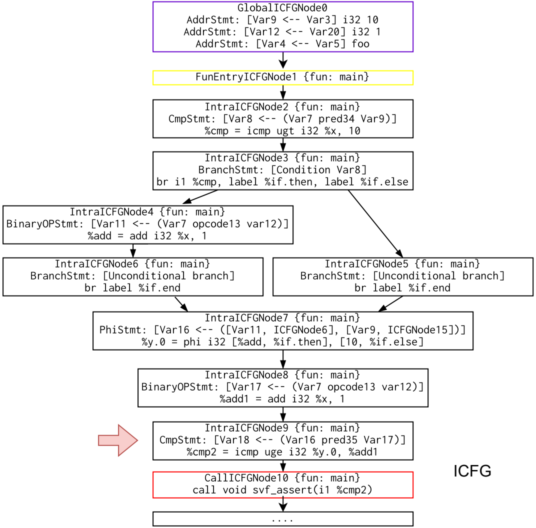
```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)    Value: NULL
ValVar4                 Value: 0x7f000005
ValVar9                 Value: 10
ValVar12                Value: 1
ValVar7                 Value: 11
ValVar8                 Value: 1
ValVar11                Value: 12
ValVar16                Value: 12
ValVar17                Value: 12
                  ...
------------------------------------------
```

Analyzing IntraICFGNode8 {fun: main}

BinaryOPStmt: [Var17 ← (Var7 opcode13 var12)]

`%add1 = add i32 %x, 1`

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

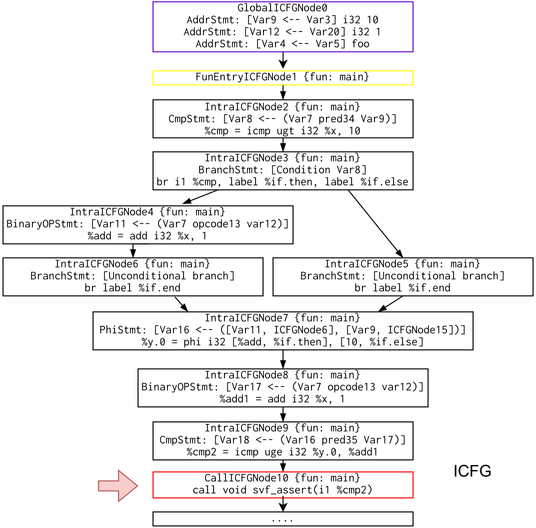
```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 11
ValVar8                  Value: 1
ValVar11                 Value: 12
ValVar16                 Value: 12
ValVar17                 Value: 12
ValVar18                 Value: 1
                   ...
------------------------------------------
```

Analyzing IntraICFGNode9 {fun: main}

CmpStmt: [Var18 ← (Var16 pred35 Var17)]

`%cmp2 = icmp uge i32 %y.0, %add1`

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

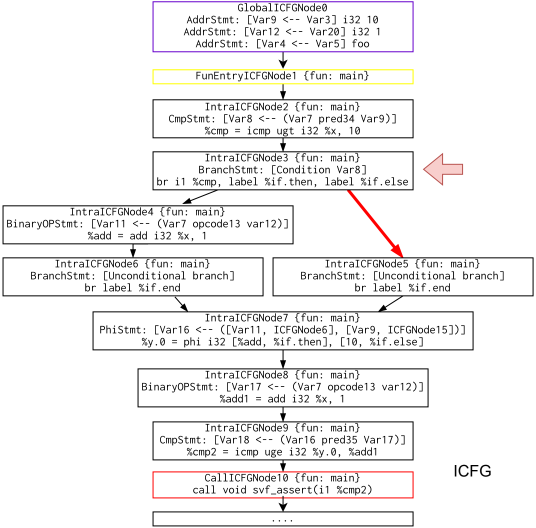
Verifying ICFG path: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 11
ValVar8                  Value: 1
ValVar11                 Value: 12
ValVar16                 Value: 12
ValVar17                 Value: 12
ValVar18                 Value: 1
                  ...
------------------------------------------
```

The assertion is successfully verified!!

START: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert*

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

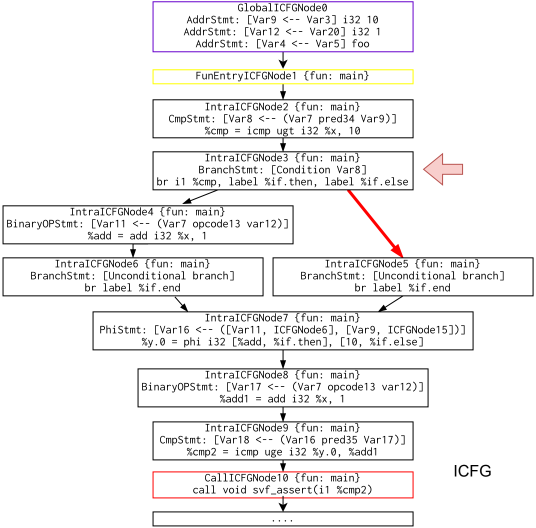
Algorithm 9: 3 handleIntra(intraEdge)

```
if intraEdge.getCondition() && !handleBranch(intraEdge) then
    return false;
else
    handleNonBranch(edge);
```

Algorithm 10: handleBranch(intraEdge)

```
cond = intraEdge.getCondition();
succ = intraEdge.getSuccessorCondValue();
getSolver().push();
addToSolver(cond == succ);
res = getSolver().check();
getSolver().pop();
if res == unsat then
    return false;
else
    addToSolver(cond == succ);
    return true;
```

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 5 → 7 → 8 → 9 → *svf_assert* (if.else branch)

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 10
ValVar8                  Value: 0
                 ...
------------------------------------------
```

Branch IntraCFGEdge: [`ICFGNode4` ← `ICFGNode3`]

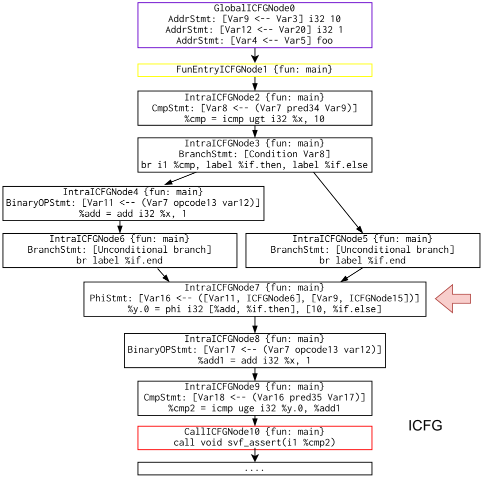
branchCondition: `%cmp = icmp ugt i32 %x, 10`

(= `ValVar8` 0)

This conditional ICFGEdge is **feasible**!!

In this path, `ValVar8`'s value is chaged to 0, therefore, `ValVar7`'s value is chaged to 10.

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 5 → 7 → 8 → 9 → *svf_assert* (if.else branch)

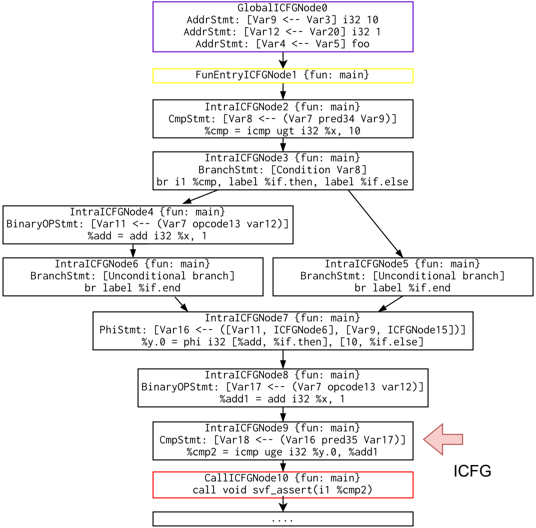
```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)   Value: NULL
ValVar4                Value: 0x7f000005
ValVar9                Value: 10
ValVar12               Value: 1
ValVar7                Value: 10
ValVar8                Value: 1
ValVar16               Value: 10
                   ...
------------------------------------------
```

Analyzing IntraICFGNode7 {fun: main}

PhiStmt: [Var16 ← ([Var11, ICFGNode6], [Var9, ICFGNode5])]

`%y.0 = phi i32 [%add, %if.then], [10, %if.else]`

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 → 5 → 7 → 8 → 9 → *svf_assert* (if.else branch)

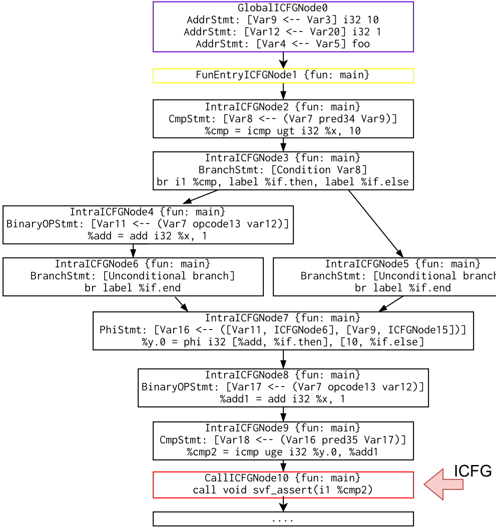
```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)    Value: NULL
ValVar4                 Value: 0x7f000005
ValVar9                 Value: 10
ValVar12                Value: 1
ValVar7                 Value: 11
ValVar8                 Value: 1
ValVar16                Value: 10
ValVar17                Value: 11
ValVar18                Value: 0
                   ...
------------------------------------------
```

Analyzing IntraICFGNode9 {fun: main}

CmpStmt: [Var18 ← (Var16 pred35 Var17)]

`%cmp2 = icmp uge i32 %y.0, %add1`

Example 4: Branches



```
GlobalICFGNode0
AddrStmt: [Var9 <-- Var3] i32 10
AddrStmt: [Var12 <-- Var20] i32 1
AddrStmt: [Var4 <-- Var5] foo

FunEntryICFGNode1 {fun: main}

IntraICFGNode2 {fun: main}
CmpStmt: [Var8 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

IntraICFGNode3 {fun: main}
BranchStmt: [Condition Var8]
br i1 %cmp, label %if.then, label %if.else

IntraICFGNode4 {fun: main}
BinaryOPStmt: [Var11 <-- (Var7 opcode13 var12)]
%add = add i32 %x, 1

IntraICFGNode6 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode5 {fun: main}
BranchStmt: [Unconditional branch]
br label %if.end

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode15])]
%y.0 = phi i32 [%add, %if.then], [10, %if.else]

IntraICFGNode8 {fun: main}
BinaryOPStmt: [Var17 <-- (Var7 opcode13 var12)]
%add1 = add i32 %x, 1

IntraICFGNode9 {fun: main}
CmpStmt: [Var18 <-- (Var16 pred35 Var17)]
%cmp2 = icmp uge i32 %y.0, %add1

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp2)

....
```

Verifying ICFG path: 0 → 1 → 2 → 3 → 5 → 7 → 8 → 9 → *svf_assert* (if.else branch)

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 11
ValVar8                  Value: 1
ValVar16                 Value: 10
ValVar17                 Value: 11
ValVar18                 Value: 0
                    ...
------------------------------------------
```

The assertion fails!!

START: 0 → 1 → 2 → 3 → 5 → 7 → 8 → 9 → *svf_assert*

Example 5: Interprocedural

```
 1 int foo(int p) {
 2     return p;
 3 }
 4 int main(int argc) {
 5   int x;
 6   int y;
 7   x = foo(3); // ctx_7
 8   y = foo(argc); // ctx_8
 9   assert(y == argc);
10 }
```

`ctx_7`: context calling `foo` at ℓ_7

`ctx_8`: context calling `foo` at ℓ_8

Example 5: Interprocedural

```
1  int foo(int p) {
2      return p;
3  }
4  int main(int argc) {
5    int x;
6    int y;
7    x = foo(3); // ctx_7
8    y = foo(argc); // ctx_8
9    assert(y == argc);
10 }
```

ctx_7: context calling `foo` at ℓ_7

ctx_8: context calling `foo` at ℓ_8

One Concrete Execution
(Concrete states)

One execution:

`argc` : 0

push calling context (calling foo at ℓ_7)

`p` : 3

calling context pop (returning from foo at ℓ_2)

`x` : 3

push calling context (calling foo at ℓ_8)

`p` : 0

pop calling context (returning from foo ℓ_2)

`y` : 0

Example 5: Interprocedural

```
1  int foo(int p) {
2      return p;
3  }
4  int main(int argc) {
5    int x;
6    int y;
7    x = foo(3); // ctx_7
8    y = foo(argc); // ctx_8
9    assert(y == argc);
10 }
```

`ctx_7`: context calling `foo` at ℓ_7

`ctx_8`: context calling `foo` at ℓ_8

One Concrete Execution (Concrete states)

One execution:

`argc` : 0

push calling context (calling foo at ℓ_7)

`p` : 3

calling context pop (returning from foo at ℓ_2)

`x` : 3

push calling context (calling foo at ℓ_8)

`p` : 0

pop calling context (returning from foo ℓ_2)

`y` : 0

Symbolic Execution (Symbolic states)

One execution:

`argc` : `getZ3Expr(argc)`

push calling context (calling foo at ℓ_7)

`ctx_7_p` : 3

pop calling context (returning from foo at ℓ_2)

`x` : `getZ3Expr(p, ctx_7)`

push calling context (calling foo at ℓ_8)

`ctx_8_p` : `getZ3Expr(argc)`

pop calling context (returning from foo ℓ_2)

`y` : `getZ3Expr(p, ctx_8)`

Example 5: Interprocedural

```
1  int foo(int p) {
2      return p;
3  }
4  int main(int argc) {
5    int x;
6    int y;
7    x = foo(3); // ctx_7
8    y = foo(argc); // ctx_8
9    assert(y == argc);
10 }
```

ctx_7: context calling foo at ℓ_7

ctx_8: context calling foo at ℓ_8

One Concrete Execution (Concrete states)

One execution:

argc : 0

push calling context (calling foo at ℓ_7)

p : 3

calling context pop (returning from foo at ℓ_2)

x : 3

push calling context (calling foo at ℓ_8)

p : 0

pop calling context (returning from foo ℓ_2)

y : 0

Symbolic Execution (Symbolic states)

One execution:

argc : getZ3Expr(argc)

push calling context (calling foo at ℓ_7)

ctx_7_p : 3

pop calling context (returning from foo at ℓ_2)

x : getZ3Expr(p, ctx_7)

push calling context (calling foo at ℓ_8)

ctx_8_p : getZ3Expr(argc)

pop calling context (returning from foo ℓ_2)

y : getZ3Expr(p, ctx_8)

Checking non-existence of counterexamples:

$\psi(N_1) \wedge \psi(N_2) \wedge \ldots \psi(N_i) \wedge \neg\psi(Q)$	Satisfiability	Counterexample
ctx_7_p $\equiv$ 3 $\wedge$ x $\equiv$ ctx_7_p $\wedge$ ctx_8_p $\equiv$ argc $\wedge$ y $\equiv$ ctx_7_p $\wedge$ y $\neq$ argc	unsat	$\varnothing$

foo's argument p needs to be **differentiated and renamed** as ctx_7_p and ctx_8_p due to two calling contexts, ctx_7 and ctx_8 to mimic the runtime call stack which holds the local variable p.

Example 5: Interprocedural

```c
int foo(int p) {
    return p;
}
int main(int argc) {
  int x;
  int y;
  x = foo(3);
  y = foo(argc);
  svf_assert(y == argc);
}
```

Source code

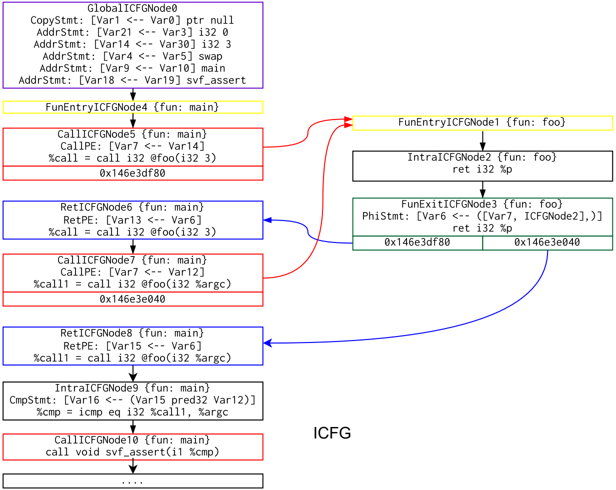
```llvm
define i32 @foo(i32 %p) #0 {
entry:
  ret i32 %p
}

define i32 @main(i32 %argc) #0 {
entry:
  %call = call i32 @foo(i32 3)
  %call1 = call i32 @foo(i32 %argc)
  %cmp = icmp eq i32 %call1, %argc
  call void @svf_assert(i1 zeroext %cmp)
  ret i32 0
}
```

LLVM IR

Example 5: Interprocedural

```
define i32 @foo(i32 %p) #0 {
entry:
  ret i32 %p
}

define i32 @main(i32 %argc) #0 {
entry:
  %call = call i32 @foo(i32 3)
  %call1 = call i32 @foo(i32 %argc)
  %cmp = icmp eq i32 %call1, %argc
  call void @svf_assert(i1 zeroext %cmp)
  ret i32 0
}
```

LLVM IR



GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

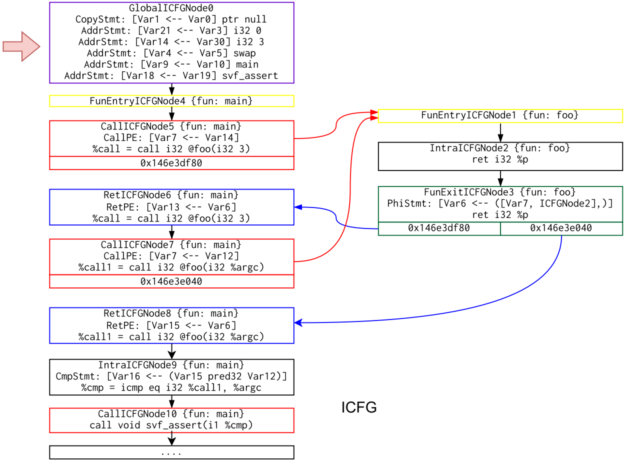
FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040

ICFG

Example 5: Interprocedural



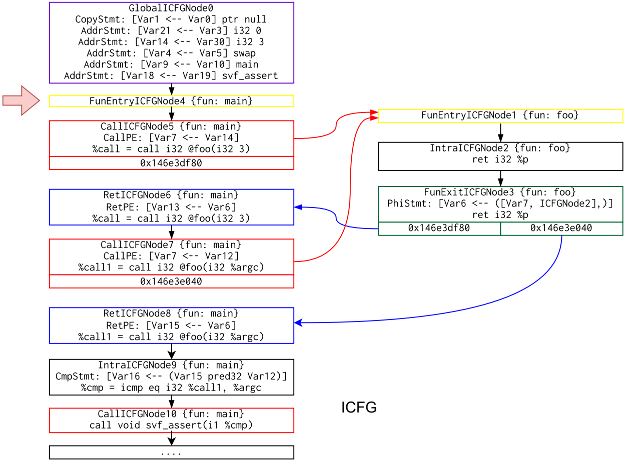
```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040
```

ICFG

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
                    ...
------------------------------------------
```

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

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert
```

```
FunEntryICFGNode4 {fun: main}
```

```
CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80
```

```
RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)
```

```
CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040
```

```
RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)
```

```
IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc
```

```
CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)
```

```
....
```

```
FunEntryICFGNode1 {fun: foo}
```

```
IntraICFGNode2 {fun: foo}
ret i32 %p
```

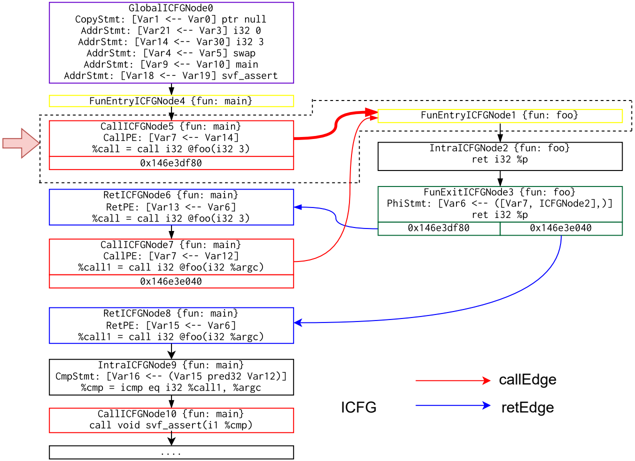
```
FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

Algorithm 11: 2 translatePath(path)

```
foreach edge ∈ path do
    if IntraEdge ← dyn_cast⟨IntraCFGEdge⟩(edge) then
        if handleIntra(IntraEdge) == false then
            return false;
    else if CallEdge ← dyn_cast⟨CallCFGEdge⟩(edge) then
        handleCall(CallEdge);
    else if RetEdge ← dyn_cast⟨RetCFGEdge⟩(edge) then
        handleRet(RetEdge);
    else
        assert(false &&"what other edges we have?");
    return true;
```

Example 5: Interprocedural

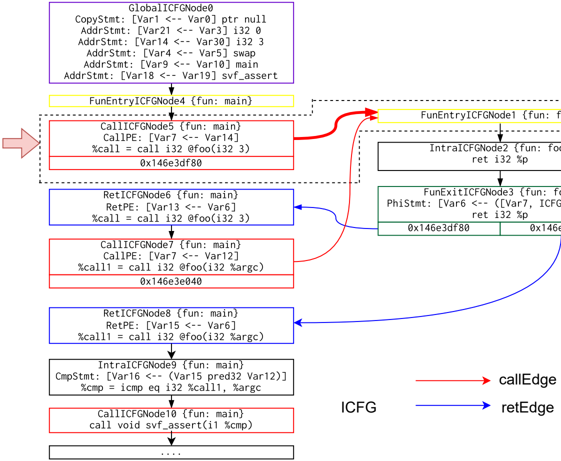


```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

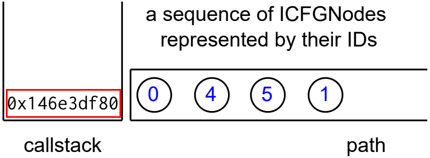
Algorithm 12: handleCall(callEdge)

```
expr_vector preCtxExprs(getCtx());
callPEs $\leftarrow$ calledge $\rightarrow$ getCallPEs();
foreach callPE $\in$ callPEs do
    preCtxExprs.push_back(rhs);
pushCallingCtx(calledge $\rightarrow$ getCallSite());
for i = 0; i < callPEs.size(); ++i do
    lhs $\leftarrow$ getZ3Expr(callPEs[i] $\rightarrow$ getLHSVarID());
    addToSolver(lhs == preCtxExprs[i]);
return true;
```

Example 5: Interprocedural



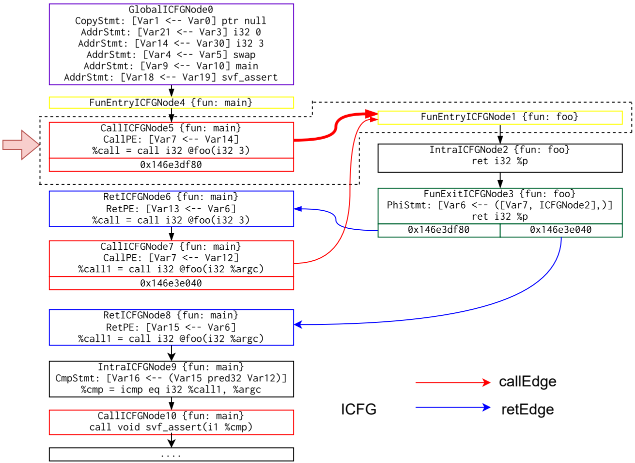
```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040
```



State of callstack after processing call edge

between CallCFGNode5 and FunEntryICFGNode1

Example 5: Interprocedural



GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

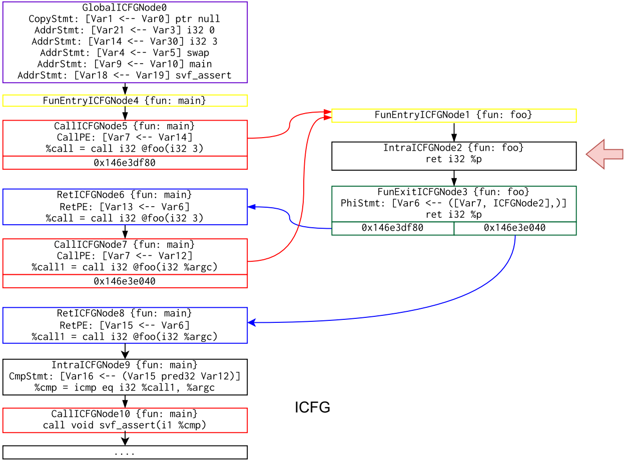
FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
+ValVar7 (ctx:[5])       Value: 3
                       ...
------------------------------------------
```

The values of Z3 expressions for each SVFVar after analyzing `CallICFGNode5`

`ValVar7 (ctx:[5])` has value 3

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert
```

```
FunEntryICFGNode4 {fun: main}
```

```
CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80
```

```
RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)
```

```
CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040
```

```
RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)
```

```
IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc
```

```
CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)
```

```
....
```

```
FunEntryICFGNode1 {fun: foo}
```

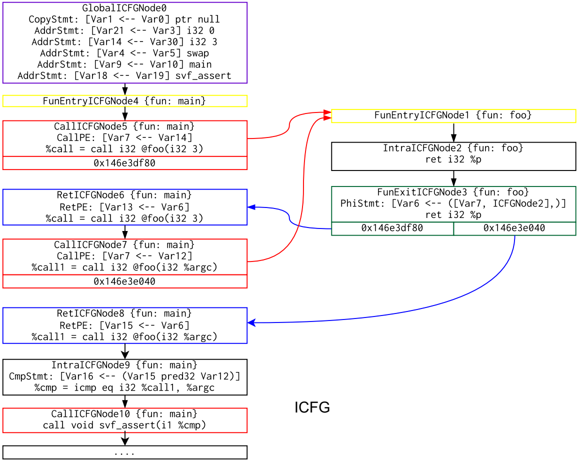
```
IntraICFGNode2 {fun: foo}
ret i32 %p
```

```
FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

`ret i32 %p` instruction.
Nothing needs to be done.
Continue.

Example 5: Interprocedural



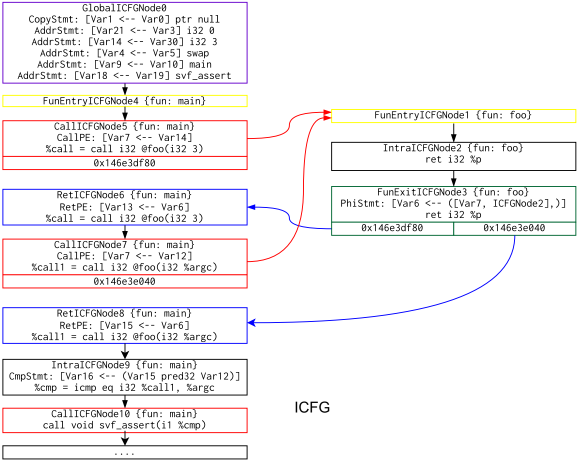
```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

Algorithm 13: 3 handlePhi(intraEdge)

```
res $\leftarrow$ getZ3Expr(phi.getResID());
opINodeFound $\leftarrow$ false;
for i $\leftarrow$ 0 to phi.getOpVarNum() $-$ 1 do
    if srcNode.getFun().postDominate(srcNode.getBB(),
    phi.getOpICFGNode(i).getBB()) then
        ope $\leftarrow$ getZ3Expr(phi.getOpVar(i).getId());
        addToSolver(res == ope);
        opINodeFound $\leftarrow$ true;
```

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

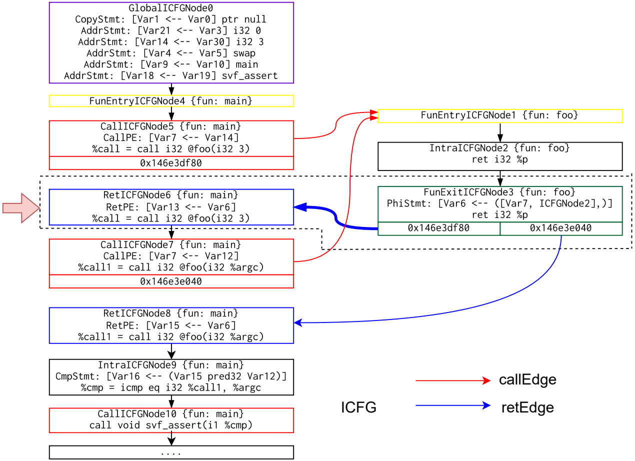
ICFG

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
ValVar7 (ctx:[5])        Value: 3
+ValVar6 (ctx:[5])       Value: 3
                   ...
------------------------------------------
```

The values of Z3 expressions for each SVFVar after analyzing `FuncExitICFGNode3`

`ValVar6 (ctx:[5])` has value 3

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

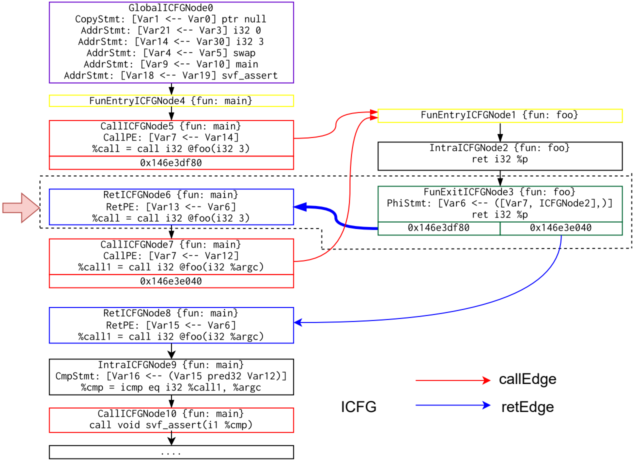
Algorithm 14: handleRet(retEdge)

```
rhs(getCtx());
if retPE ← retEdge.getRetPE() then
    rhs ← getZ3Expr(retPE.getRHSVarID());
popCallingCtx();
if retPE ← retEdge.getRetPE() then
    lhs ← getZ3Expr(retPE.getLHSVarID());
    addToSolver(lhs == rhs);
return true;
```

Note:retPE.getRHSVarID() returns ValVar6

getZ3Expr(ValVar6) binds ValVar6 with the current context and returns the Z3 Expression for ValVar6 (ctx:[5])

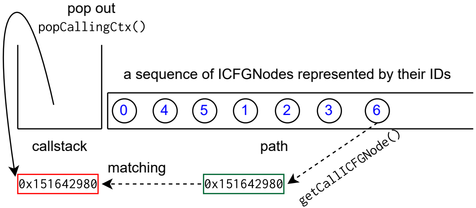
Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

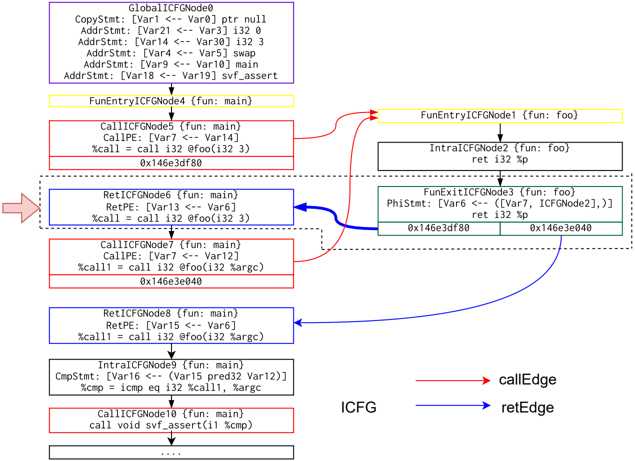
Algorithm 15: handleRet(retEdge)

```
rhs(getCtx());
if retPE ← retEdge.getRetPE() then
    rhs ← getZ3Expr(retPE.getRHSVarID());
popCallingCtx();
if retPE ← retEdge.getRetPE() then
    lhs ← getZ3Expr(retPE.getLHSVarID());
    addToSolver(lhs == rhs);
return true;
```



State of callstack while processing return edge from FunExitICFGNode3 to RetICFGNode6

Example 5: Interprocedural

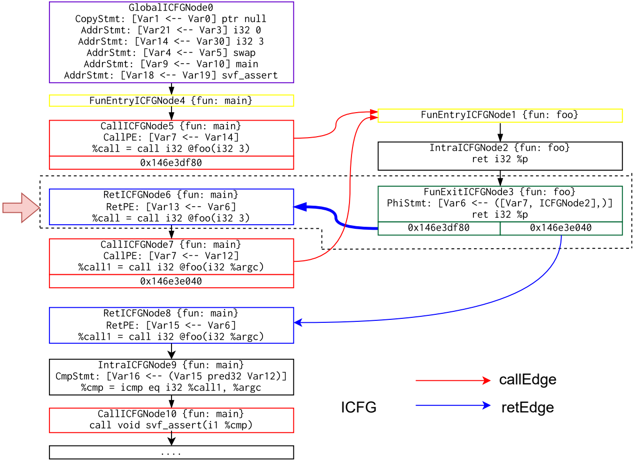


```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040
```

Algorithm 16: handleRet(retEdge)

```
rhs(getCtx());
if retPE ← retEdge.getRetPE() then
    rhs ← getZ3Expr(retPE.getRHSVarID());
popCallingCtx();
if retPE ← retEdge.getRetPE() then
    lhs ← getZ3Expr(retPE.getLHSVarID());
    addToSolver(lhs == rhs);
return true;
```

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

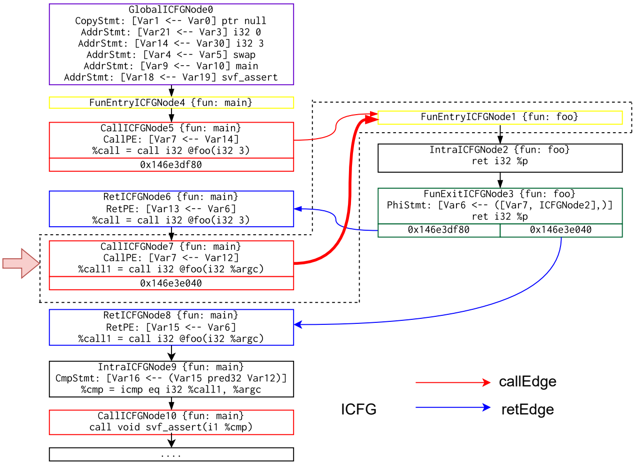
callEdge

retEdge

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
ValVar7 (ctx:[5])        Value: 3
ValVar6 (ctx:[5])        Value: 3
ValVar13                 Value: 3
                        ...
------------------------------------------
```

The values of Z3 expressions for each SVFVar after analyzing `RetICFGNode6`

Example 5: Interprocedural

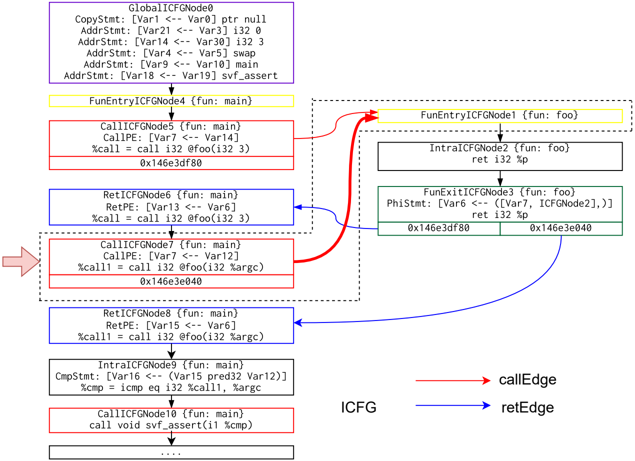


```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040
```

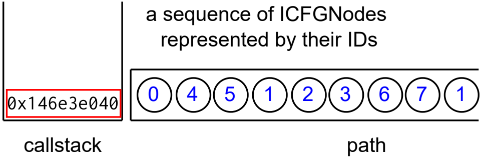
Algorithm 17: handleCall(callEdge)

```
expr_vector preCtxExprs(getCtx());
callPEs ← calledge → getCallPEs();
foreach callPE ∈ callPEs do
    preCtxExprs.push_back(rhs);
pushCallingCtx(calledge → getCallSite());
for i = 0; i < callPEs.size(); ++i do
    lhs ← getZ3Expr(callPEs[i] → getLHSVarID());
    addToSolver(lhs == preCtxExprs[i]);
return true;
```

Example 5: Interprocedural

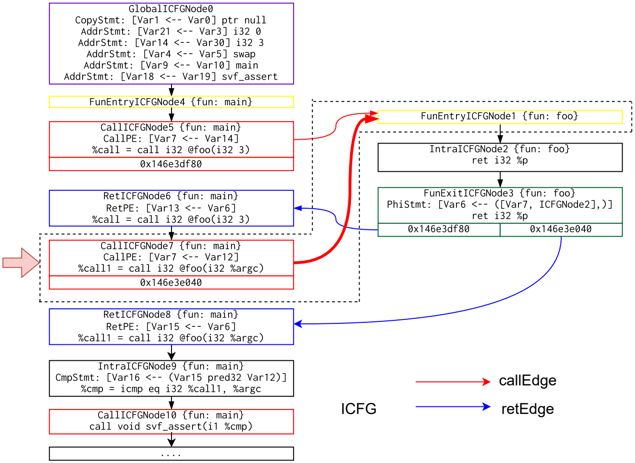


```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040
```



State of callstack after processing call edge between CallCFGNode7 and FunEntryICFGNode1

Example 5: Interprocedural

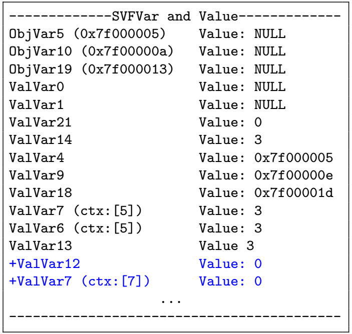


```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

callEdge

retEdge



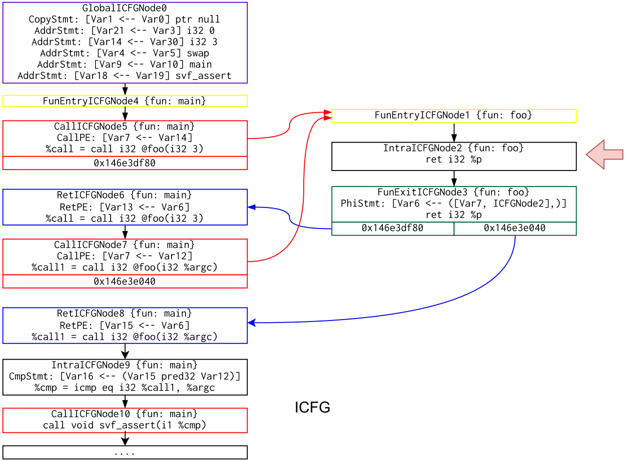
```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
ValVar7 (ctx:[5])        Value: 3
ValVar6 (ctx:[5])        Value: 3
ValVar13                 Value 3
+ValVar12                Value: 0
+ValVar7 (ctx:[7])       Value: 0
                ...
------------------------------------------
```

The values of Z3 expressions for each SVFVar after analyzing `CallICFGNode7`

* `ValVar12` is not initialized in the program, thus is evaluated as 0

* `ValVar7 (ctx:[7])` has value 0

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert
```

```
FunEntryICFGNode4 {fun: main}
```

```
CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80
```

```
RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)
```

```
CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040
```

```
RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)
```

```
IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc
```

```
CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)
```

```
....
```

```
FunEntryICFGNode1 {fun: foo}
```

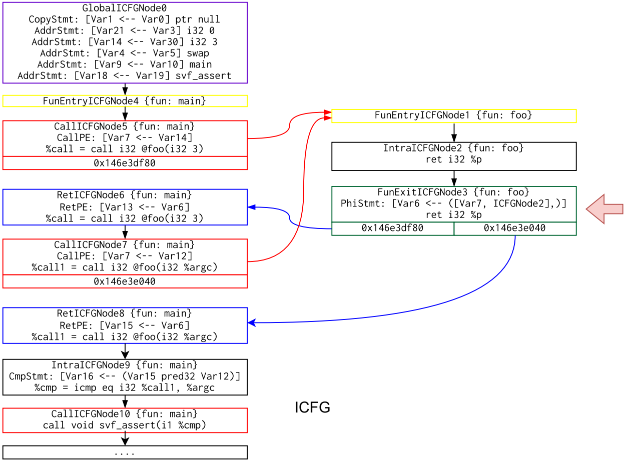
```
IntraICFGNode2 {fun: foo}
ret i32 %p
```

```
FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

`ret i32 %p` instruction.
Nothing needs to be done.
Continue.

Example 5: Interprocedural



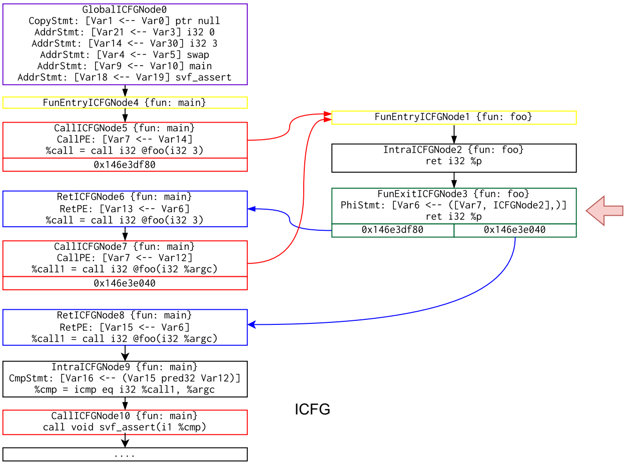
```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040
```

ICFG

Algorithm 18: 3 handlePhi(intraEdge)

```
res ← getZ3Expr(phi.getResID());
opINodeFound ← false;
for i ← 0 to phi.getOpVarNum() − 1 do
    if srcNode.getFun().postDominate(srcNode.getBB(),
    phi.getOpICFGNode(i).getBB()) then
        ope ← getZ3Expr(phi.getOpVar(i).getId());
        addToSolver(res == ope);
        opINodeFound ← true;
```

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert
```

```
FunEntryICFGNode4 {fun: main}
```

```
CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80
```

```
RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)
```

```
CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040
```

```
RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)
```

```
IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc
```

```
CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)
```

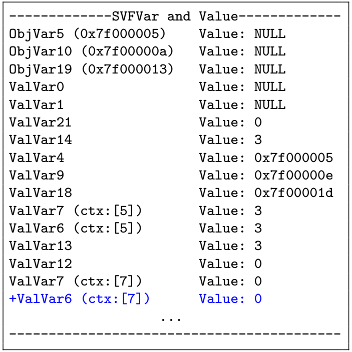
```
....
```

```
FunEntryICFGNode1 {fun: foo}
```

```
IntraICFGNode2 {fun: foo}
ret i32 %p
```

```
FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

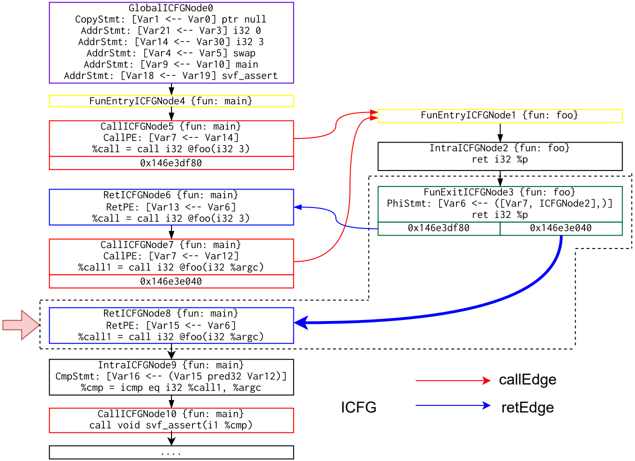


```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
ValVar7 (ctx:[5])        Value: 3
ValVar6 (ctx:[5])        Value: 3
ValVar13                 Value: 3
ValVar12                 Value: 0
ValVar7 (ctx:[7])        Value: 0
+ValVar6 (ctx:[7])       Value: 0
                   ...
------------------------------------------
```

The values of Z3 expressions for each SVFVar after analyzing `FuncExitICFGNode3`

`ValVar6 (ctx:[7])` has value 0

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040
```

ICFG

callEdge

retEdge

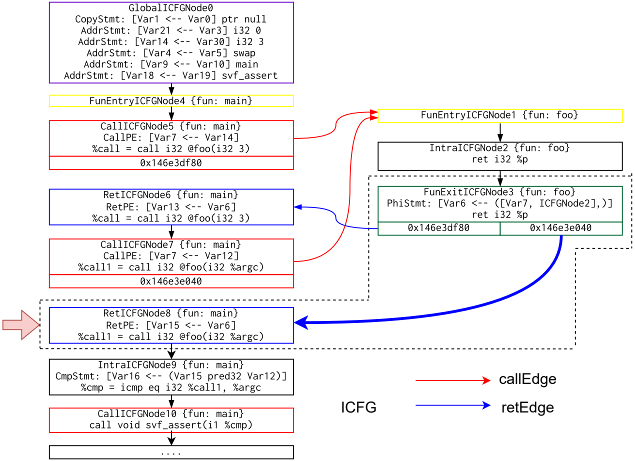
Algorithm 19: handleRet(retEdge)

```
rhs(getCtx());
if retPE ← retEdge.getRetPE() then
    rhs ← getZ3Expr(retPE.getRHSVarID());
popCallingCtx();
if retPE ← retEdge.getRetPE() then
    lhs ← getZ3Expr(retPE.getLHSVarID());
    addToSolver(lhs == rhs);
return true;
```

Note: `retPE.getRHSVarID()` returns `ValVar6`

`getZ3Expr(ValVar6)` binds `ValVar6` with the current context and returns the Z3 Expression for `ValVar6 (ctx:[7])`

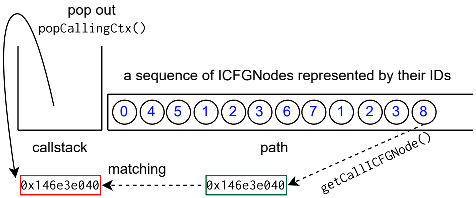
Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

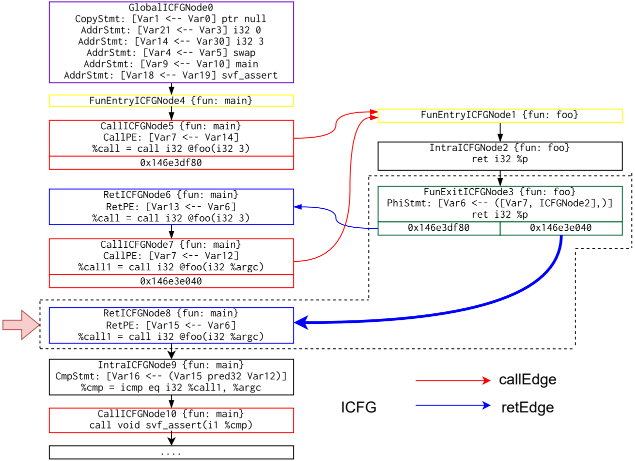
Algorithm 20: handleRet(retEdge)

```
rhs(getCtx());
if retPE ← retEdge.getRetPE() then
    rhs ← getZ3Expr(retPE.getRHSVarID());
popCallingCtx();
if retPE ← retEdge.getRetPE() then
    lhs ← getZ3Expr(retPE.getLHSVarID());
    addToSolver(lhs == rhs);
return true;
```



State of callstack while processing return edge from FunExitICFGNode3 to RetICFGNode8

Example 5: Interprocedural



GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

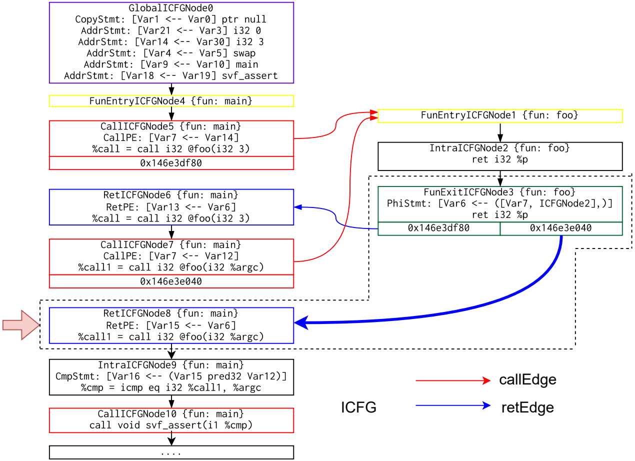
IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80 | 0x146e3e040

Algorithm 21: handleRet(retEdge)

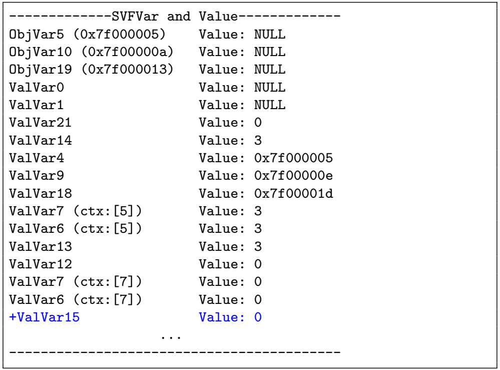
```
rhs(getCtx());
if retPE ← retEdge.getRetPE() then
    rhs ← getZ3Expr(retPE.getRHSVarID());
popCallingCtx();
if retPE ← retEdge.getRetPE() then
    lhs ← getZ3Expr(retPE.getLHSVarID());
    addToSolver(lhs == rhs);
return true;
```

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

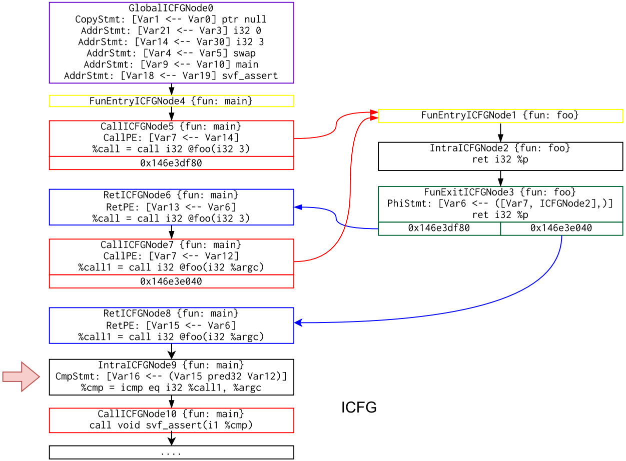
ICFG
callEdge
retEdge



```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
ValVar7 (ctx:[5])        Value: 3
ValVar6 (ctx:[5])        Value: 3
ValVar13                 Value: 3
ValVar12                 Value: 0
ValVar7 (ctx:[7])        Value: 0
ValVar6 (ctx:[7])        Value: 0
+ValVar15                Value: 0
                    ...
------------------------------------------
```

The values of Z3 expressions for each SVFVar after analyzing `RetICFGNode8`

Example 5: Interprocedural



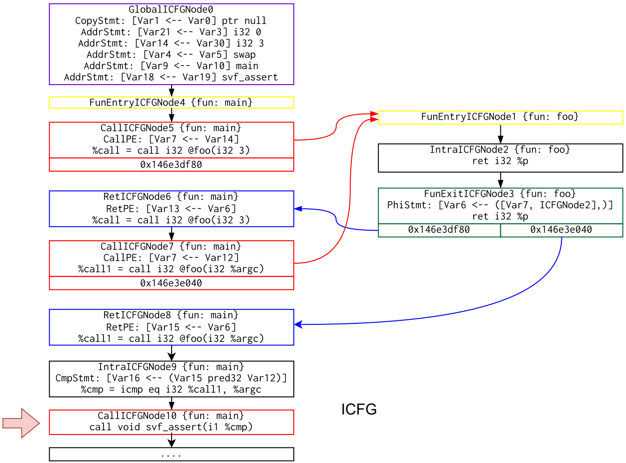
```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert

FunEntryICFGNode4 {fun: main}

CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80

RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)

CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040

RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)

IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc

CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)

....

FunEntryICFGNode1 {fun: foo}

IntraICFGNode2 {fun: foo}
ret i32 %p

FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
ValVar7 (ctx:[5])        Value: 3
ValVar6 (ctx:[5])        Value: 3
ValVar13                 Value: 3
ValVar12                 Value: 0
ValVar7 (ctx:[7])        Value: 0
ValVar6 (ctx:[7])        Value: 0
ValVar15                 Value: 0
+ValVar16                Value: 1
                   ...
------------------------------------------
```

The values of Z3 expressions for each SVFVar after analyzing `IntraICFGNode9`

Example 5: Interprocedural



```
GlobalICFGNode0
CopyStmt: [Var1 <-- Var0] ptr null
AddrStmt: [Var21 <-- Var3] i32 0
AddrStmt: [Var14 <-- Var30] i32 3
AddrStmt: [Var4 <-- Var5] swap
AddrStmt: [Var9 <-- Var10] main
AddrStmt: [Var18 <-- Var19] svf_assert
```

```
FunEntryICFGNode4 {fun: main}
```

```
CallICFGNode5 {fun: main}
CallPE: [Var7 <-- Var14]
%call = call i32 @foo(i32 3)
0x146e3df80
```

```
RetICFGNode6 {fun: main}
RetPE: [Var13 <-- Var6]
%call = call i32 @foo(i32 3)
```

```
CallICFGNode7 {fun: main}
CallPE: [Var7 <-- Var12]
%call1 = call i32 @foo(i32 %argc)
0x146e3e040
```

```
RetICFGNode8 {fun: main}
RetPE: [Var15 <-- Var6]
%call1 = call i32 @foo(i32 %argc)
```

```
IntraICFGNode9 {fun: main}
CmpStmt: [Var16 <-- (Var15 pred32 Var12)]
%cmp = icmp eq i32 %call1, %argc
```

```
CallICFGNode10 {fun: main}
call void svf_assert(i1 %cmp)
```

```
....
```

```
FunEntryICFGNode1 {fun: foo}
```

```
IntraICFGNode2 {fun: foo}
ret i32 %p
```

```
FunExitICFGNode3 {fun: foo}
PhiStmt: [Var6 <-- ([Var7, ICFGNode2],)]
ret i32 %p
0x146e3df80    0x146e3e040
```

ICFG

```
-------------SVFVar and Value-------------
ObjVar5 (0x7f000005)     Value: NULL
ObjVar10 (0x7f00000a)    Value: NULL
ObjVar19 (0x7f000013)    Value: NULL
ValVar0                  Value: NULL
ValVar1                  Value: NULL
ValVar21                 Value: 0
ValVar14                 Value: 3
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 0x7f00000e
ValVar18                 Value: 0x7f00001d
ValVar7 (ctx:[5])        Value: 3
ValVar6 (ctx:[5])        Value: 3
ValVar13                 Value: 3
ValVar12                 Value: 0
ValVar7 (ctx:[7])        Value: 0
ValVar6 (ctx:[7])        Value: 0
ValVar15                 Value: 0
ValVar16                 Value: 1
                              ...
------------------------------------------
```

The assertion is successfully verified!!

START: 0 → 4 → 5 → 1 → 2 → 3 → 6 → 7 → 1 → 2

→ 3 → 8 → 9 →→ *svf_assert*

What’s next?

- (1) Understand SSE algorithms and examples in the slides
- (2) Finish implementing the automated translation from code to Z3 formulas using `SSE` and `Z3SSEMgr` in Assignment 2