Code Verification Using Symbolic Execution
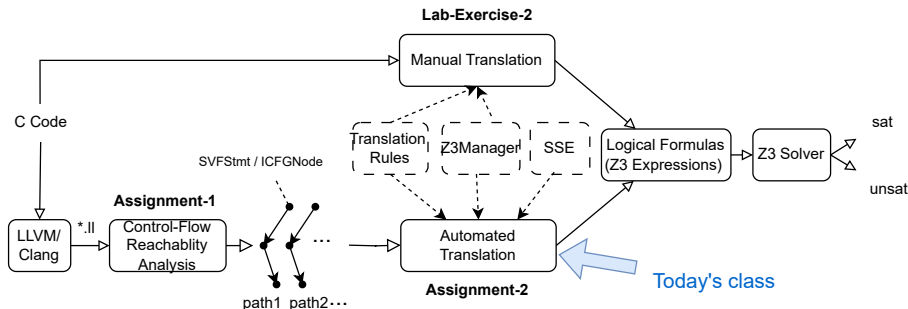
(Week 7)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Code Verification Using Static Symbolic Execution

- We will detail the algorithms of translating branches and calls/returns.
- We will showcase branches and interprocedural examples.

Translate Branches and Calls/Returns

Algorithm 1: translatePath(path)

```
1  foreach edge ∈ path do
2      if intra ← dyn_cast⟨Intra⟩(edge) then
3          if handleIntra(intra) == false then
4              return false
5      else if call ← dyn_cast⟨CallEdge⟩(edge) then
6          handleCall(call)
7      else if ret ← dyn_cast⟨RetEdge⟩(edge) then
8          handleRet(ret)
9      return true
```

Algorithm 2: handleIntra(intraEdge)

```
1  if intraEdge.getCondition() then
2      if !handleBranch(intraEdge) then
3          return false;
4      else
5          return handleNonBranch(intraEdge);
6  else
7      return handleNonBranch(intraEdge);
```

Algorithm 3: handleCall(callEdge)

```
1  // Your code starts here ..;
2
3  // For each CallPE lhs = rhs:
4  // (1) Retrieve two z3 expressions ⟨[c],rhs⟩ and
     ⟨[c'],lhs⟩ under contexts c (caller's context) and
     c' (callee's context); Note that pushCallingCtx
     will make c become c'.
5  // (2) Add ⟨[c],rhs⟩ ≡ ⟨[c'],lhs⟩ into the solver.
```

Algorithm 4: handleRet(retEdge)

```
1  // Your code starts here ..;
2
3  // For each RetPE lhs = rhs:
4  // (1) Retrieve two z3 expressions ⟨[c'],rhs⟩ and
     ⟨[c],lhs⟩ under contexts c (caller's context) and
     c' (callee's context); Note that popCallingCtx
     will make c' become c.
5  // (2) Add ⟨[c'],rhs⟩ ≡ ⟨[c],lhs⟩ into the solver.
```

`getCondition()` and `getSuccessorCondValue()`

Given a `if/else` branch on the ICFG as the following:
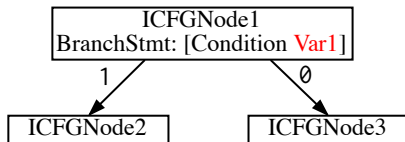
- `edge → getCondition()` returns the branch condition (of type `SVFValue*` which is a **boolean** (for if/else) or a **numeric** (for switch).
  - `edge → getCondition()` returns `nullptr` if this IntraCFGEdge is not a branch.
- Given the condition value, you could obtain the ID of the corresponding SVFVar (Var1) via `edge → getCondition() → getId()`.

`getCondition()` and `getSuccessorCondValue()`

Given a `if/else` branch on the ICFG as the following:
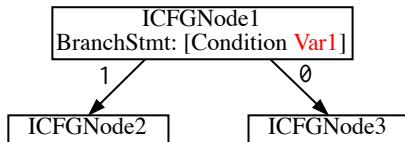
- `edge→getSuccessorCondValue()` returns the actual condition `succCondValue`, whose value is either 1 (for if branch to execute) or 0 (for the else branch to execute).
  - For example, the `succCondValue` is 1 on the edge from ICFGNode1 to ICFGNode2, and 0 on the edge from ICFGNode1 to ICFGNode3.
- When evaluating the feasibility of a branch edge (e.g., ICFGNode1 to ICFGNode2) given an ICFG path, check `sat` of `Var1 == succCondValue` against the solver's existing constraints.

Example 4: Branches

```
1  void main(int x){
2      int y;
3      if(x > 10) {
4          y = x + 1;
5      }
6      else {
7          y = 10;
8      }
9      svf_assert(y >= x + 1);
10 }
```

Source code

```
1  define void @main(i32 %x) #0 {
2  entry:
3    %cmp = icmp ugt i32 %x, 10
4    br i1 %cmp, label %if.then, label %if.else
5
6  if.then:
7    %add = add i32 %x, 1
8    br label %if.end
9
10 if.else:
11   br label %if.end
12
13 if.end: = %if.else, %if.then
14   %y.0 = phi i32 [%add, %if.then], [10, %if.else]
15   %add1 = add i32 %x, 1
16   %cmp2 = icmp uge i32 %y.0, %add1
17   call void @svf_assert(i1 zeroext %cmp2)
18   ret void
19 }
```

LLVM IR

Example 4: Branches

```
1  define void @main(i32 %x) #0 {
2  entry:
3    %cmp = icmp ugt i32 %x, 10
4    br i1 %cmp, label %if.then, label %if.else
5
6  if.then:
7    %add = add i32 %x, 1
8    br label %if.end
9
10 if.else:
11   br label %if.end
12
13 if.end: = %if.else, %if.then
14   %y.0 = phi i32 [%add, %if.then], [10, %if.else]
15   %add1 = add i32 %x, 1
16   %cmp2 = icmp uge i32 %y.0, %add1
17   call void @svf_assert(i1 zeroext %cmp2)
18   ret void
19 }
```
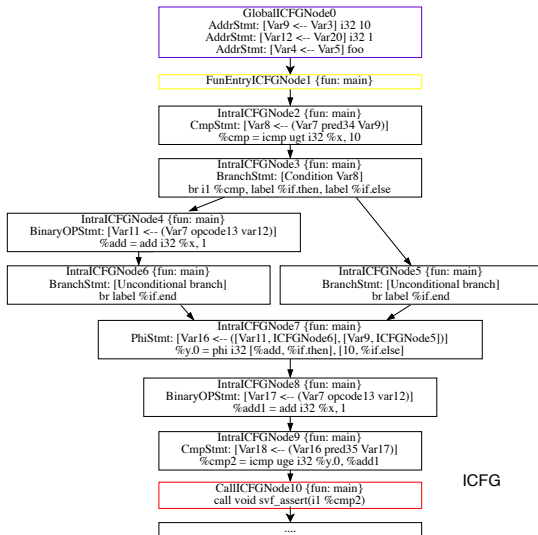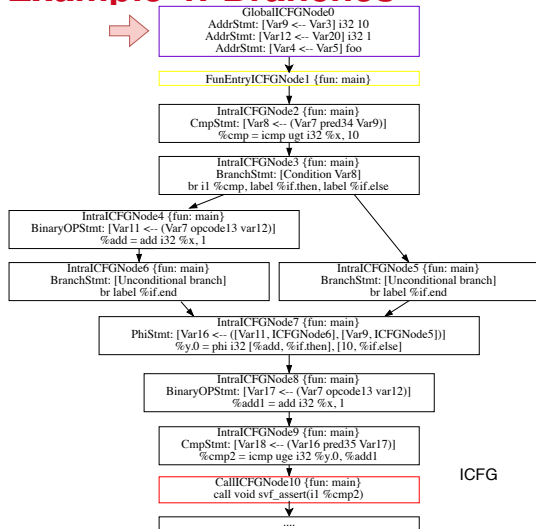
LLVM IR

Two ICFG paths:

- `if.then branch:`
  0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert*

- `if.else branch:`
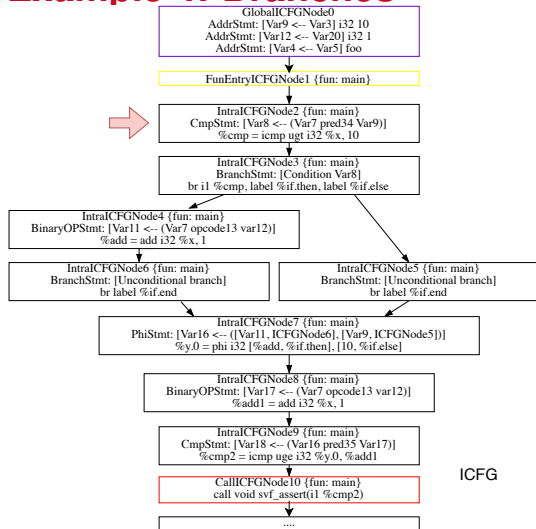 0 → 1 → 2 → 3 → 5 → 7 → 8 → 9 → *svf_assert*

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, ICFGNode5])]
%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: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow$
$6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow$ *svf_assert* (if.then branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \land Var12 \equiv 1 \land Var4 \equiv 0x7f000005$

```
-------------SVFVar and Value------------
ObjVar5 (0x7f000005)    Value: NULL
ValVar4                 Value: 0x7f000005
ValVar9                 Value: 10
ValVar12                Value: 1
                ...
-----------------------------------------
```
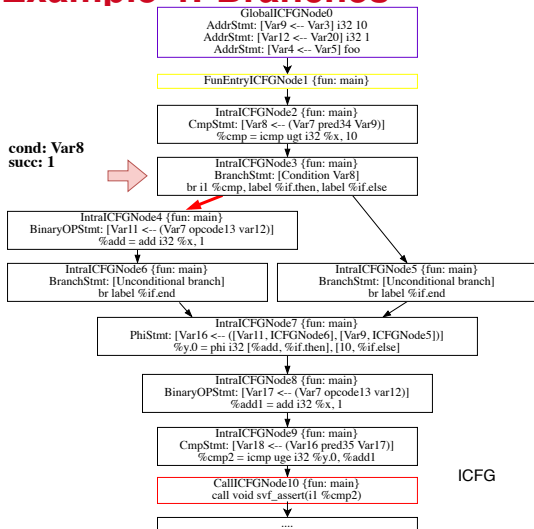
Example 4: Branches

Verifying ICFG path: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow svf\_assert$ (if.then branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \wedge Var12 \equiv 1 \wedge Var4 \equiv 0x7f000005$
ICFGNode 2	$\wedge \; Var8 \equiv ite(Var7 > Var9, 1, 0)$

```
-------------SVFVar and Value------------
ObjVar5 (0x7f000005)    Value: NULL
ValVar4                 Value: 0x7f000005
ValVar9                 Value: 10
ValVar12                Value: 1
ValVar7                 Value: 11
ValVar8                 Value: 1
                ...
-----------------------------------------
```
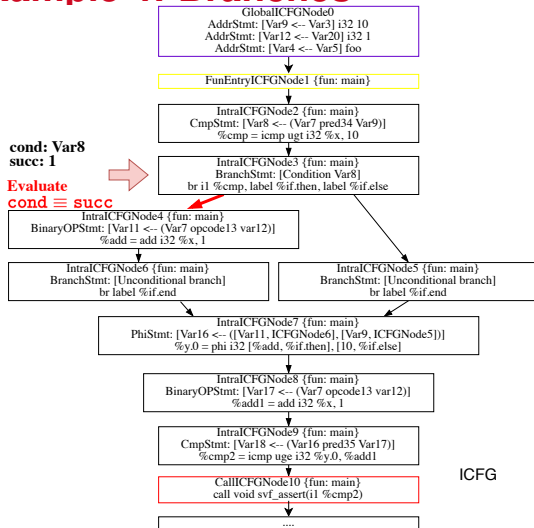
Example 4: Branches

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

cond: Var8
succ: 1

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, ICFGNode5])]
%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)`

2 **if** `intraEdge.getCondition()` &&
`!handleBranch(intraEdge)` **then**
4 | return false;
6 **else**
8 | `handleNonBranch(edge);`

Algorithm 6: `handleBranch(intraEdge)`

1 cond = ... // Retrieve branch condition boolean var;
2 succ = ... // Edge's succ condition value (0 or 1);
3 // Evaluate path feasibility;
4 // The current path/edge is feasible if 'cond' and 'succ' have the same value, when evaluated in the solver; otherwise this branch edge is infeasible;
5 **if** the brach edge is infeasible **then**
6 | return false;
7 **else**
8 | // add (cond≡succ) to solver if it holds;
9 | 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: [Var7 <-- (Var7 pred34 Var9)]
%cmp = icmp ugt i32 %x, 10

cond: Var8
succ: 1
Evaluate
cond ≡ succ

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

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

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

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode5])]
%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)`

2 **if** `intraEdge.getCondition()` *&&*
`!handleBranch(intraEdge)` **then**

4 | return `false`;

6 **else**

8 | `handleNonBranch(edge)`;

Algorithm 8: `handleBranch(intraEdge)`

1 `cond = ...` // Retrieve branch condition boolean var;
2 `succ = ...` // Edge's succ condition value (0 or 1);
3 // Evaluate path feasibility;
4 // The current path/edge is feasible if 'cond' and 'succ' have the same value, when evaluated in the solver; otherwise this branch edge is infeasible;
5 **if** the brach edge is infeasible **then**
6 | return `false`;
7 **else**
8 | // add (cond≡succ) to solver if it holds;
9 | return `true`;

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

Example 4: Branches

ICFG

Verifying ICFG path: 0 → 1 → 2 → 3 →4 → 6 → 7 → 8 → 9 → *svf_assert* (if.then branch)
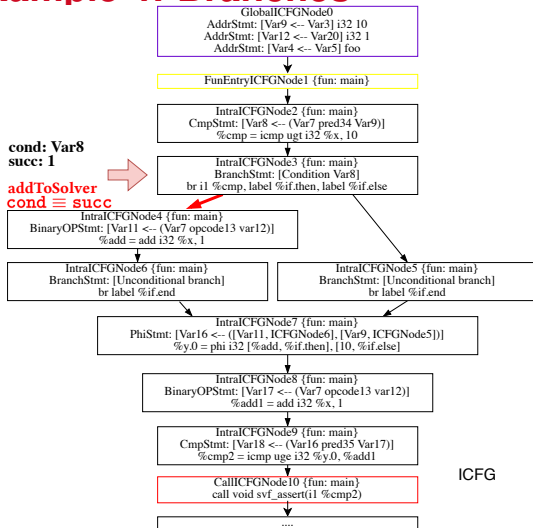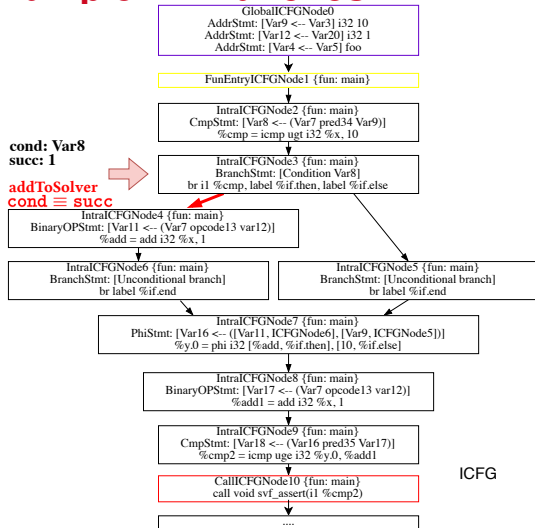
ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \wedge Var12 \equiv 1 \wedge Var4 \equiv 0x7f000005$
ICFGNode 2	$\wedge \quad Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge 3 → 4	

The constraint $Var8 \equiv 1$ is evaluated to be SAT.
The conditional ICFGEdge [ICFGNode3 → ICFGNode4] is feasible.

```
-------------SVFVar and Value------------
ObjVar5 (0x7f000005)   Value: NULL
ValVar4                Value: 0x7f000005
ValVar9                Value: 10
ValVar12               Value: 1
ValVar7                Value: 11
ValVar8                Value: 1
              ...
-----------------------------------------
```
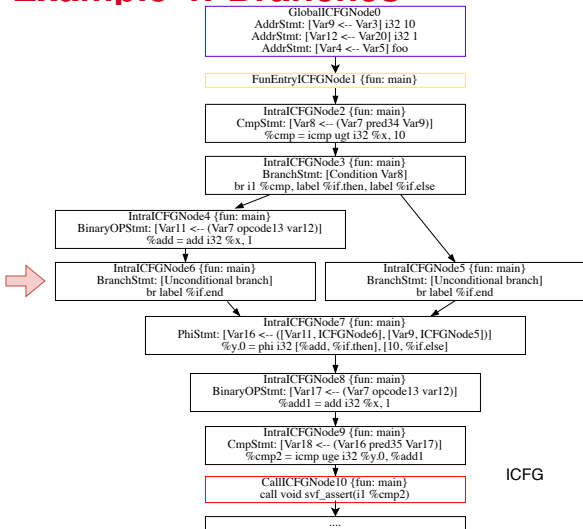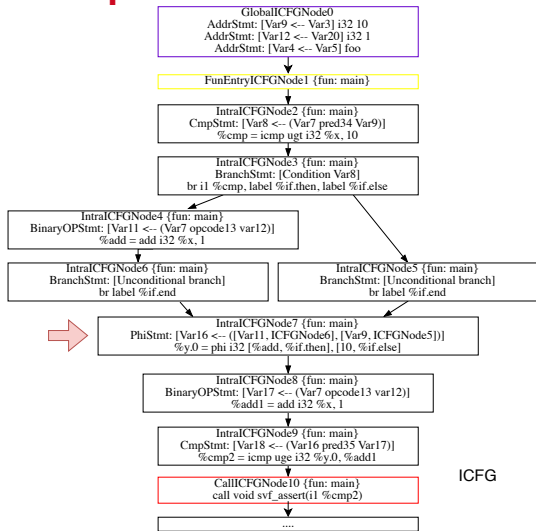
Example 4: Branches

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

FunEntryICFGNode1 {fun: main}

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

cond: Var8
succ: 1

addToSolver
cond ≡ succ

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

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

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

IntraICFGNode7 {fun: main}
PhiStmt: [Var16 <-- ([Var11, ICFGNode6], [Var9, ICFGNode5])]
%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)`

1 **if** intraEdge.getCondition() &&
  !handleBranch(intraEdge) **then**
2 | return false;
3 **else**
4 | handleNonBranch(edge);

Algorithm 10: `handleBranch(intraEdge)`

1 cond = ... // Retrieve branch condition boolean var;
2 succ = ... // Edge's succ condition value (0 or 1);
3 // Evaluate path feasibility;
4 // The current path/edge is feasible if 'cond' and 'succ' have the same value, when evaluated in the solver; otherwise this branch edge is infeasible;
5 **if** the brach edge is infeasible **then**
6 | return false;
7 **else**
8 | // add (cond≡succ) to solver if it holds;
9 | return true;

Note: res is sat, so the conditional ICFGEdge
[ICFGNode4 ← ICFGNode3] 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

cond: Var8
succ: 1

addToSolver
cond ≡ succ

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, ICFGNode5])]
%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: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow$
$6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow$ *svf_assert* (if.then branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \land Var12 \equiv 1 \land Var4 \equiv 0x7f000005$
ICFGNode 2	$\land \ Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge 3 → 4	$\land \ Var8 \equiv 1$

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

Example 4: Branches

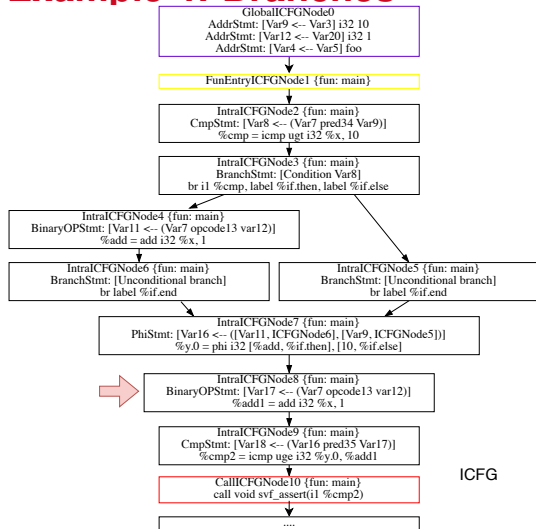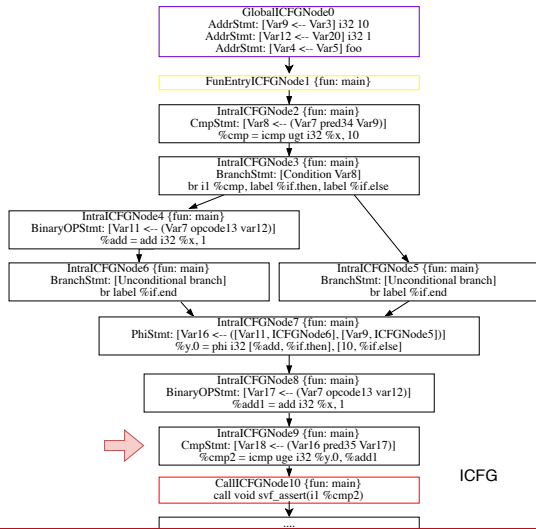Verifying ICFG path: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow$
$6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow svf\_assert$ (if.then branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \wedge Var12 \equiv 1 \wedge Var4 \equiv 0x7f000005$
ICFGNode 2	$\wedge \quad Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge $3 \rightarrow 4$	$\wedge \quad Var8 \equiv 1$
ICFGNode 4	$\wedge \quad Var11 \equiv Var7 + Var12$

```
------------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
                 ...
----------------------------------------
```
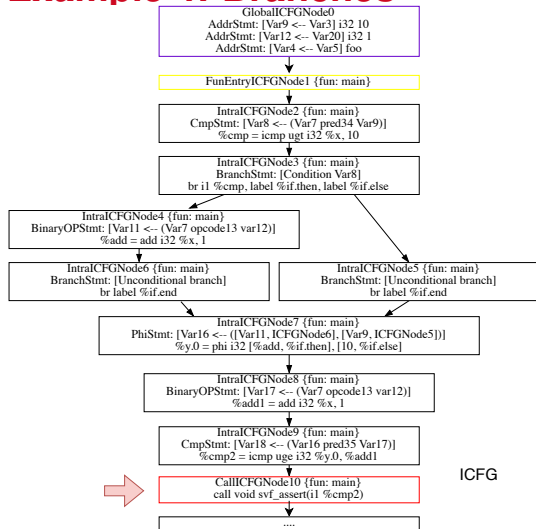
Example 4: Branches

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, ICFGNode5])]
%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 11: 3 `handleIntra(intraEdge)`

2 **if** intraEdge.getCondition() **&&**
   !handleBranch(intraEdge) **then**
4 | return false;

6 **else**
8 | 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, ICFGNode5])]
%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}
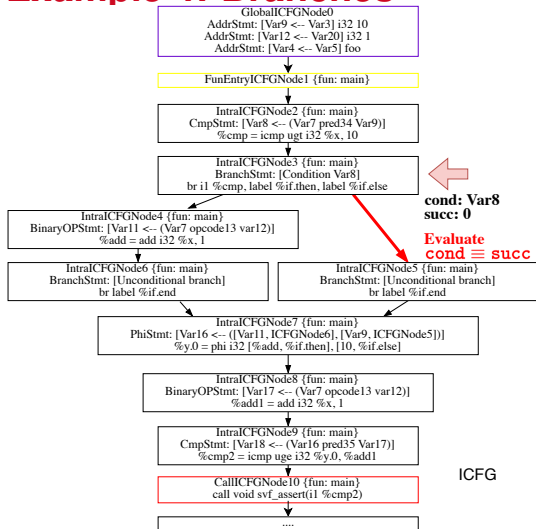call void svf_assert(i1 %cmp2)

....

ICFG

Verifying ICFG path: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow$
$6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow svf\_assert$ (if.then branch)

Algorithm 12: 3 `handlePhi(edge)`

1 `res ← getZ3Expr(phi.getResID());`
2 `opINodeFound ← false;`
3 **for** $i \leftarrow 0$ *to* `phi.getOpVarNum()` $- 1$ **do**
4   **if** `edge.srcNode()` *postdominates* `phi.getOpICFGNode(i)` **then**
5   `ope ← getZ3Expr(phi.getOpVar(i).getId());`
6   `addToSolver(res == ope);`
7   `opINodeFound ← true;`

Given `Var16 ← ([Var11, ICFGNode6], [Var9, ICFGNode5])`,
only `Var16 ≡ Var11` holds as we traverse the `if.then` branch
from `ICFGNode6`, where `Var11`'s definition originates

`edge.srcNode()`: ICFGNode6
`phi.getOpICFGNode(i)`: ICFGNode where *i-th*
phi operand var's definition originates.

ICFGNode *m* postdominates *n*: if all paths to the graph's exit
starting at *n* must go through *m* (a node postdominates itself).

16

Example 4: Branches

Verifying ICFG path: 0 → 1 → 2 → 3 → 4 →

6 → 7 → 8 → 9 → *svf_assert* (if.then branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \land Var12 \equiv 1 \land Var4 \equiv 0x7f000005$
ICFGNode 2	$\land \quad Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge 3 → 4	$\land \quad Var8 \equiv 1$
ICFGNode 4	$\land \quad Var11 \equiv Var7 + Var12$
ICFGNode 7	$\land \quad Var16 \equiv Var11$

```
-----------SVFVar and Value-----------
              ...
ValVar9              Value: 10
ValVar12             Value: 1
ValVar7              Value: 11
ValVar8              Value: 1
ValVar11             Value: 12
ValVar16             Value: 12
              ...
---------------------------------------
```


ICFG

Example 4: Branches

Verifying ICFG path: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow svf\_assert$ (if.then branch)

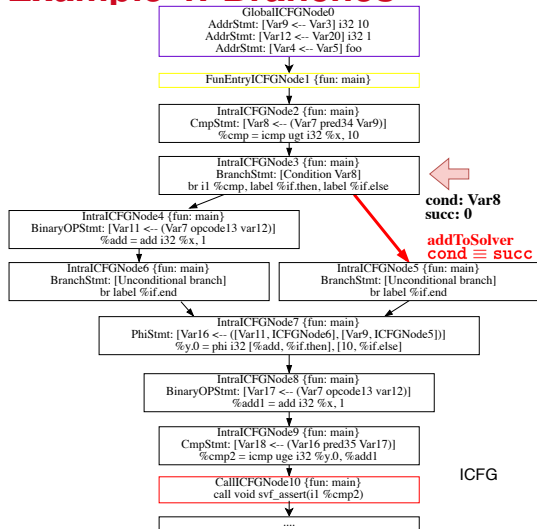| ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \wedge Var12 \equiv 1 \wedge Var4 \equiv 0x7f000005$
ICFGNode 2	$\wedge \quad Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge $3 \rightarrow 4$	$\wedge \quad Var8 \equiv 1$
ICFGNode 4	$\wedge \quad Var11 \equiv Var7 + Var12$
ICFGNode 7	$\wedge \quad Var16 \equiv Var11$
ICFGNode 8	$\wedge \quad Var17 \equiv Var7 + Var12$

```
------------SVFVar and Value------------
                    ...
ValVar9              Value: 10
ValVar12             Value: 1
ValVar7              Value: 11
ValVar8              Value: 1
ValVar11             Value: 12
ValVar16             Value: 12
ValVar17             Value: 12
                    ...
----------------------------------------
```

Example 4: Branches

Verifying ICFG path: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow$

$6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow svf\_assert$ (if.then branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \wedge Var12 \equiv 1 \wedge Var4 \equiv 0x7f000005$
ICFGNode 2	$\wedge \ \ Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge $3 \rightarrow 4$	$\wedge \ \ Var8 \equiv 1$
ICFGNode 4	$\wedge \ \ Var11 \equiv Var7 + Var12$
ICFGNode 7	$\wedge \ \ Var16 \equiv Var11$
ICFGNode 8	$\wedge \ \ Var17 \equiv Var7 + Var12$
ICFGNode 9	$\wedge \ \ Var18 \equiv ite(Var16 \geq Var17, 1, 0)$

```
------------SVFVar and Value------------
                    ...
ValVar9             Value: 10
ValVar12            Value: 1
ValVar7             Value: 11
ValVar8             Value: 1
ValVar11            Value: 12
ValVar16            Value: 12
ValVar17            Value: 12
ValVar18            Value: 1
                    ...
```

Example 4: Branches

ICFG

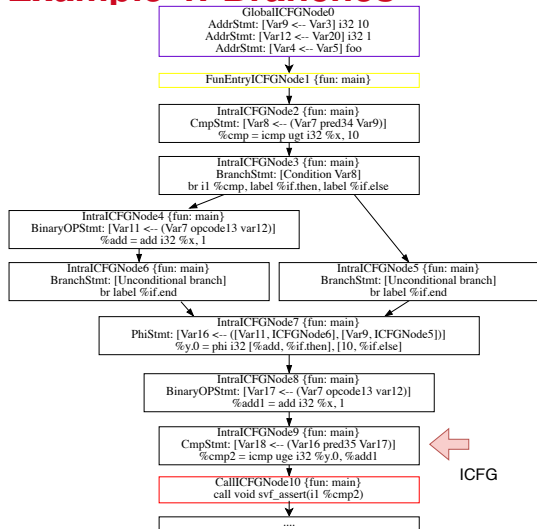Verifying ICFG path: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow$ *svf_assert* (if.then branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\text{Var9} \equiv 10 \wedge \text{Var12} \equiv 1 \wedge \text{Var4} \equiv \text{0x7f000005}$
ICFGNode 2	$\wedge \ \text{Var8} \equiv \text{ite}(\text{Var7} > \text{Var9}, 1, 0)$
ICFGEdge $3 \rightarrow 4$	$\wedge \ \text{Var8} \equiv 1$
ICFGNode 4	$\wedge \ \text{Var11} \equiv \text{Var7} + \text{Var12}$
ICFGNode 7	$\wedge \ \text{Var16} \equiv \text{Var11}$
ICFGNode 8	$\wedge \ \text{Var17} \equiv \text{Var7} + \text{Var12}$
ICFGNode 9	$\wedge \ \text{Var18} \equiv \text{ite}(\text{Var16} \geq \text{Var17}, 1, 0)$
ICFGNode 10	$\wedge \ \text{Var18} \equiv 0$ (negation of the assert condition)

Solver yields **UNSAT** (i.e., no counter example), therefore, the assertion is successfully verified!!

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
```
cond: Var8
succ: 0

Evaluate
cond ≡ succ

```
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, ICFGNode5])]
%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 13: 3 handleIntra(intraEdge)

2 **if** intraEdge.getCondition() *&&*
  !handleBranch(intraEdge) **then**
4 | return false;

6 **else**
8 | handleNonBranch(edge);

Algorithm 14: handleBranch(intraEdge)

1 cond = . . . // Retrieve branch condition boolean var;
2 succ = . . . // Edge's succ condition value (0 or 1);
3 // Evaluate path feasibility;
4 // The current path/edge is feasible if 'cond' and
  'succ' have the same value, when evaluated in the
  solver; otherwise this branch edge is infeasible;
5 **if** the brach edge is infeasible **then**
6 | return false;
7 **else**
8 | // add (cond≡succ) to solver if it holds;
9 | return true;

Example 4: Branches

Verifying ICFG path: $0 \to 1 \to 2 \to 3 \to 5 \to 7 \to 8 \to 9 \to svf\_assert$ (if.else branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\text{Var9} \equiv 10 \wedge \text{Var12} \equiv 1 \wedge \text{Var4} \equiv \text{0x7f000005}$
ICFGNode 2	$\wedge \quad \text{Var8} \equiv \text{ite}(\text{Var7} > \text{Var9}, 1, 0)$
ICFGEdge 3 $\to$ 4	

The constraint $\text{Var8} \equiv 0$ is evaluated to be SAT
The conditional ICFGEdge [$\text{ICFGNode3} \to \text{ICFGNode5}$] is feasible.

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

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
```

cond: Var8
succ: 0

addToSolver
cond ≡ succ

```
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, ICFGNode5])]
%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 15: 3 `handleIntra(intraEdge)`

2 if intraEdge.getCondition() *&&*
!handleBranch(intraEdge) **then**

4 | return false;

6 else

8 | handleNonBranch(edge);

Algorithm 16: `handleBranch(intraEdge)`

1 cond = ... // Retrieve branch condition boolean var;

2 succ = ... // Edge's succ condition value (0 or 1);

3 // Evaluate path feasibility;

4 // The current path/edge is feasible if 'cond' and 'succ' have the same value, when evaluated in the solver; otherwise this branch edge is infeasible;

5 if the brach edge is infeasible **then**

6 | return false;

7 else

8 | // add (cond≡succ) to solver if it holds;

9 | return true;

Example 4: Branches

Verifying ICFG path: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow$
$7 \rightarrow 8 \rightarrow 9 \rightarrow svf\_assert$ (if.else branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \land Var12 \equiv 1 \land Var4 \equiv 0x7f000005$
ICFGNode 2	$\land\ Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge $3 \rightarrow 5$	$\land\ Var8 \equiv 0$

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

Example 4: Branches

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

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \land Var12 \equiv 1 \land Var4 \equiv 0x7f000005$
ICFGNode 2	$\land \quad Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge $3 \to 5$	$\land \quad Var8 \equiv 0$
ICFGNode 7	$\land \quad Var16 \equiv Var9$

```
------------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
            ...
----------------------------------------
```

Example 4: Branches

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var9 \equiv 10 \land Var12 \equiv 1 \land Var4 \equiv 0x7f000005$
ICFGNode 2	$\land \quad Var8 \equiv ite(Var7 > Var9, 1, 0)$
ICFGEdge 3 → 5	$\land \quad Var8 \equiv 0$
ICFGNode 7	$\land \quad Var16 \equiv Var9$
ICFGNode 8	$\land \quad Var17 \equiv Var7 + Var12$
ICFGNode 9	$\land \quad Var18 \equiv ite(Var16 \geq Var17, 1, 0)$

```
------------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
               ...
----------------------------------------
```

ICFG

Example 4: Branches

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

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\text{Var9} \equiv 10 \wedge \text{Var12} \equiv 1 \wedge \text{Var4} \equiv 0x7f000005$
ICFGNode 2	$\wedge \quad \text{Var8} \equiv \text{ite}(\text{Var7} > \text{Var9}, 1, 0)$
ICFGEdge $3 \rightarrow 5$	$\wedge \quad \text{Var8} \equiv 0$
ICFGNode 7	$\wedge \quad \text{Var16} \equiv \text{Var9}$
ICFGNode 8	$\wedge \quad \text{Var17} \equiv \text{Var7} + \text{Var12}$
ICFGNode 9	$\wedge \quad \text{Var18} \equiv \text{ite}(\text{Var16} \geq \text{Var17}, 1, 0)$
ICFGNode 10	$\wedge \quad \text{Var18} \equiv 0$ (negation of the assert condition)

Solver yields **SAT**, a counterexample exists:

$(\text{Var16} \equiv 10 \wedge \text{Var17} \equiv 11)$.

The assertion is violated and fails!

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:[ℓ₇]
8      y = foo(argc); //ctx:[ℓ₈]
9      svf_assert(y == argc);
10 }
```

$[\ell_7]$: calling context of foo at ℓ_7

$[\ell_8]$: calling context of 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:[ℓ_7]
8    y = foo(argc); //ctx:[ℓ_8]
9    svf_assert(y == argc);
10 }
```

[ℓ_7]: calling context of foo at ℓ_7

[ℓ_8]: calling context of foo at ℓ_8

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:[ℓ₇]
8    y = foo(argc); //ctx:[ℓ₈]
9    svf_assert(y == argc);
10 }
```

$[\ell_7]$: calling context of foo at ℓ_7

$[\ell_8]$: calling context of foo at ℓ_8

Concrete Execution
(Concrete states)

One execution:

```
argc  :  0
push calling context (calling foo at ℓ₇)
  p   :  3
calling context pop (returning from foo at ℓ₂)
  x   :  3
push calling context (calling foo at ℓ₈)
  p   :  0
pop calling context (returning from foo ℓ₂)
  y   :  0
```

Symbolic Execution
(Symbolic states)

```
 argc   :  getZ3Expr(argc)
push abstract calling context (current ctx:[ℓ₇])
⟨[ℓ₇],p⟩  :  3
   x   :  getZ3Expr(⟨[ℓ₇],p⟩)
pop abstract calling context (current ctx:[ ])
push abstract calling context (current ctx:[ℓ₈])
⟨[ℓ₈],p⟩  :  getZ3Expr(argc)
   y   :  getZ3Expr(⟨[ℓ₈],p⟩)
pop abstract calling context (current ctx:[ ])
```

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:[ℓ₇]
8    y = foo(argc); //ctx:[ℓ₈]
9    svf_assert(y == argc);
10 }
```

$[\ell_7]$: calling context of foo at ℓ_7

$[\ell_8]$: calling context of foo at ℓ_8

Concrete Execution
(Concrete states)

One execution:

```
argc  :  0
push calling context (calling foo at ℓ₇)
  p   :  3
calling context pop (returning from foo at ℓ₂)
  x   :  3
push calling context (calling foo at ℓ₈)
  p   :  0
pop calling context (returning from foo ℓ₂)
  y   :  0
```

Symbolic Execution
(Symbolic states)

```
 argc   :  getZ3Expr(argc)
push abstract calling context (current ctx:[ℓ₇])
⟨[ℓ₇], p⟩ :  3
   x    :  getZ3Expr(⟨[ℓ₇], p⟩)
pop abstract calling context (current ctx:[ ])
push abstract calling context (current ctx:[ℓ₈])
⟨[ℓ₈], p⟩ :  getZ3Expr(argc)
   y    :  getZ3Expr(⟨[ℓ₈], p⟩)
pop abstract calling context (current ctx:[ ])
```

Checking non-existence of counterexamples:

$\psi(N_1) \wedge \psi(N_2) \wedge \ldots \psi(N_i) \wedge \neg\psi(Q)$	Satisfiability	Counterexample
$\langle[\ell_7], p\rangle \equiv 3 \wedge x \equiv \langle[\ell_7], p\rangle \wedge \langle[\ell_8], p\rangle \equiv \text{argc} \wedge y \equiv \langle[\ell_7], p\rangle \wedge y \neq \text{argc}$	unsat	$\varnothing$

foo's argument p needs to be **differentiated and renamed** as $\langle[\ell_7], p\rangle$ and $\langle[\ell_8], p\rangle$ due to two calling contexts,

$[\ell_7]$ and $[\ell_8]$ to mimic the runtime call stack which holds the local variable p.

Example 5: Interprocedural

SSE::getZ3Expr(SVFVarID) in `Assignment-2`

- Get an Z3 expression based on SVFVarID and the current calling context `callingCtx`
- `callingCtx` is maintained per ICFG path by calling SSE::pushCallingCtx and SSE::popCallingCtx when handling `CallCFGEdge` and `RetCFGEdge`.

```
1 z3::expr SSE::getZ3Expr(NodeID idx) const {
2     return z3Mgr->getZ3Expr(idx, callingCtx);
3 }
```

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);
8    y = foo(argc);
9    svf_assert(y == argc);
10 }
```

Source code

```
1  define i32 @foo(i32 %p) #0 {
2  entry:
3    ret i32 %p
4  }
5
6  define i32 @main(i32 %argc) #0 {
7  entry:
8    %call = call i32 @foo(i32 3)
9    %call1 = call i32 @foo(i32 %argc)
10   %cmp = icmp eq i32 %call1, %argc
11   call void @svf_assert(i1 zeroext %cmp)
12   ret i32 0
13 }
```

LLVM IR
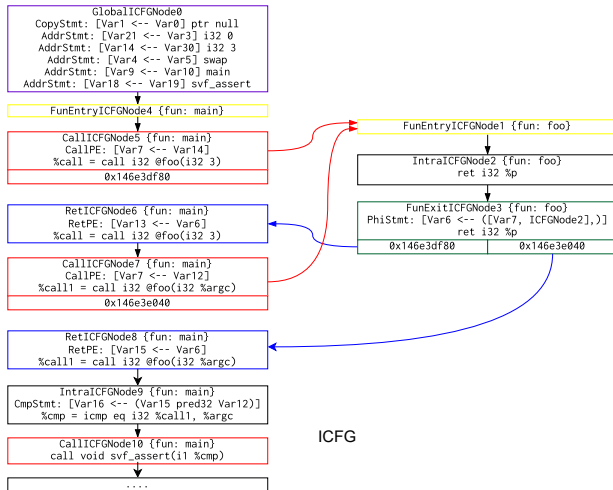
Example 5: Interprocedural
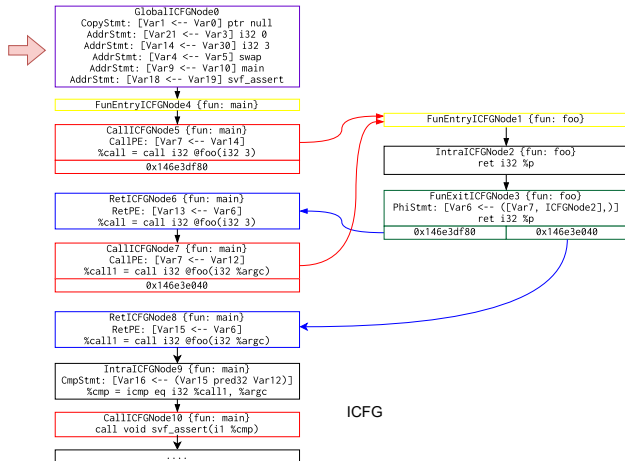
LLVM IR

ICFG

Example 5: Interprocedural

ICFG

Verifying ICFG path: 0 → 4 → 5 → 1 → 2 → 3 → 6 → 7 → 1 → 2 → 3 → 8 → 9 → svf_assert

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var21 \equiv 0 \wedge Var14 \equiv 3 \wedge \ldots$

```
-------------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
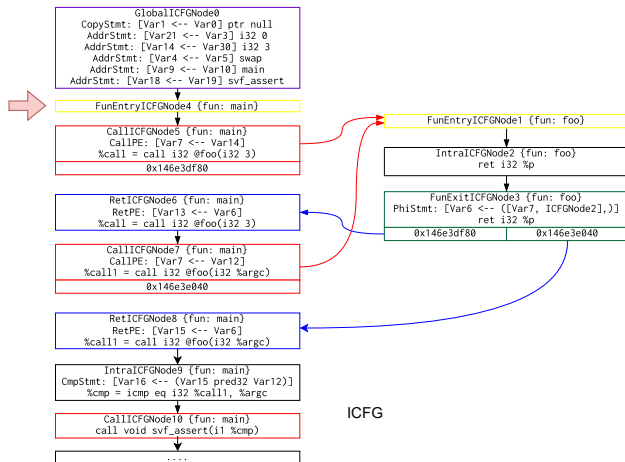
Example 5: Interprocedural
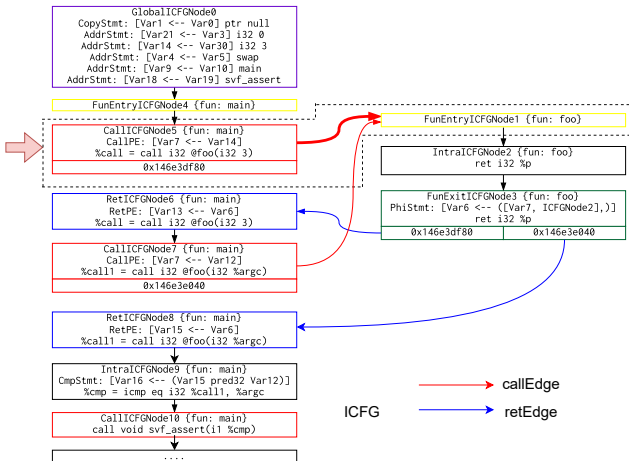
Algorithm 18: `handleCall(callEdge)`

1 // (1) For each CallPE lhs = rhs: retrieve rhs expr ⟨[c], rhs⟩ under callingCtx c (caller's context)

2 // (2) pushCallingCtx using current calliste ICFGNode; Note that pushCallingCtx will make c become c'. ;

3 // (3) For each CallPE lhs = rhs: retrieve lhs expr ⟨[c'], lhs⟩ under callingCtx c' (callee's context) ;

4 // (4) For each CallPE lhs = rhs: add ⟨[c], rhs⟩ ≡ ⟨[c'], lhs⟩ into the solver.

5 return true;

Example 5: Interprocedural

Verifying ICFG path: $0 \to 4 \to 5 \to 1 \to 2 \to 3 \to 6 \to 7 \to 1 \to 2 \to 3 \to 8 \to 9 \to svf\_assert$

Calling Context

After processing the call edge

`CallICFGNode5 → FunEntryICFGNode1`

Example 5: Interprocedural
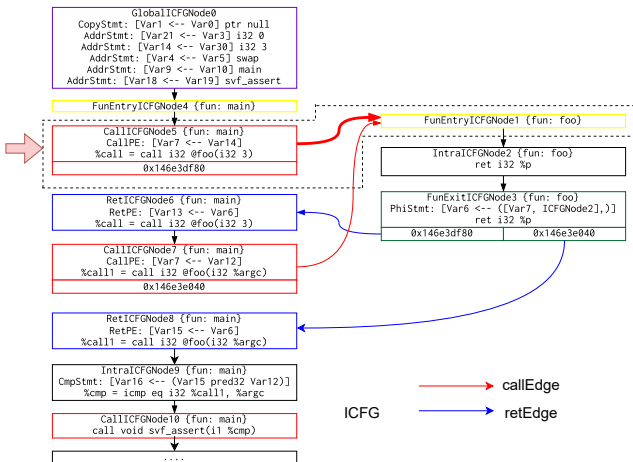
Algorithm 19: `handleCall(callEdge)`

1 // (1) For each CallPE lhs = rhs: retrieve rhs expr $\langle [c], rhs \rangle$ under callingCtx c (caller's context)

2 // (2) pushCallingCtx using current calliste ICFGNode; Note that pushCallingCtx will make c become c'. ;

3 // (3) For each CallPE lhs = rhs: retrieve lhs expr $\langle [c'], lhs \rangle$ under callingCtx c' (callee's context) ;

4 // (4) For each CallPE lhs = rhs: add $\langle [c], rhs \rangle \equiv \langle [c'], lhs \rangle$ into the solver.

5 **return** true;

Example 5: Interprocedural

Verifying ICFG path: $0 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9 \rightarrow svf\_assert$

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\mathtt{Var21} \equiv 0 \wedge \mathtt{Var14} \equiv 3 \wedge \ldots$
ICFGNode 5	$\wedge \ \langle [\mathtt{ICFGNode5}], \mathtt{Var7} \rangle \equiv \mathtt{Var14}$

```
-------------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                  Value: 3
                  ...
------------------------------------------
```

Note: SVFVar and Value table is printed under the calling context [`CallICFGNode5`]

Example 5: Interprocedural

Verifying ICFG path: 0 → 4 → 5 → 1 → 2 → 3 → 6 → 7 → 1 → 2 → 3 → 8 → 9 → *svf_assert*

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

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 20: 3 handlePhi(edge)

1 res ← getZ3Expr(phi.getResID());
2 opINodeFound ← false;
3 for i ← 0 *to* phi.getOpVarNum() − 1 **do**
4   **if** edge.srcNode() *postdominates*
     phi.getOpICFGNode(i) **then**
5   ope ← getZ3Expr(phi.getOpVar(i).getId());
6   addToSolver(res == ope);
7   opINodeFound ← true;

Example 5: Interprocedural

ICFG

Verifying ICFG path: $0 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9 \rightarrow svf\_assert$

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\text{Var21} \equiv 0 \wedge \text{Var14} \equiv 3 \wedge \ldots$
ICFGNode 5	$\wedge \ \langle[\text{ICFGNode5}], \text{Var7}\rangle \equiv \text{Var14}$
ICFGNode 3	$\wedge \ \langle[\text{ICFGNode5}], \text{Var6}\rangle \equiv \langle[\text{ICFGNode5}], \text{Var7}\rangle$

```
-------------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                 Value: 3
ValVar6                 Value: 3
              ...
------------------------------------------
```

Note: SVFVar and Value table is printed under the calling context [`CallICFGNode5`]

Example 5: Interprocedural
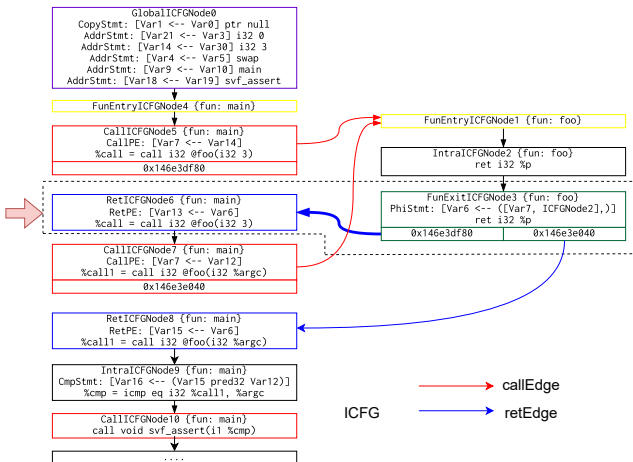
Algorithm 21: `handleRet`(retEdge)

1. <mark>// (1) For each RetPE lhs = rhs: retrieve rhs</mark>
   <mark>expr ⟨[c'], rhs⟩ under callingCtx c' (callee's</mark>
   <mark>context);</mark>
2. // (2) popCallingCtx(); This will make c' become
   c (caller's context)
3. // (3) For each RetPE lhs = rhs: retrieve lhs
   expr ⟨[c], lhs⟩ under callingCtx c (caller's
   context)
4. // (4) Add ⟨[c'], rhs⟩ ≡ ⟨[c], lhs⟩ into the solver.
5. return true;

Note: retPE.getRHSVarID() returns ValVar6
`getZ3Expr(ValVar6)` binds `ValVar6` with the current
`callingCtx` [ICFGNode5] and returns the Z3 expression for
[ICFGNode5], Var6
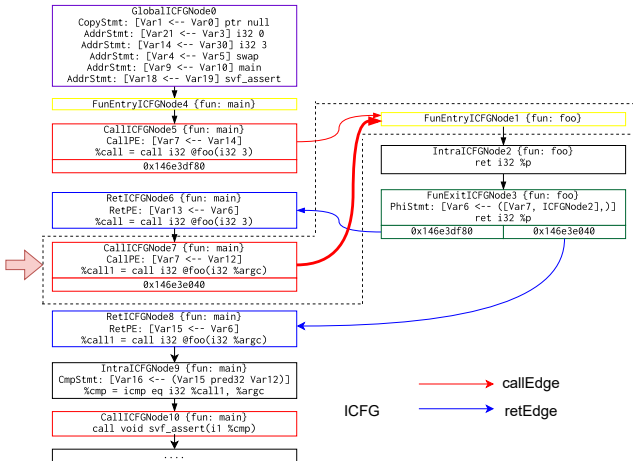
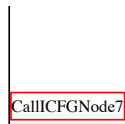# Example 5: Interprocedural

Example 5: Interprocedural

Algorithm 23: `handleRet`(retEdge)

1 // (1) For each RetPE lhs = rhs: retrieve rhs expr $\langle [c'], \text{rhs} \rangle$ under callingCtx c' (callee's context);

2 // (2) popCallingCtx(); This will make c' become c (caller's context)

3 // (3) For each RetPE lhs = rhs: retrieve lhs expr $\langle [c], \text{lhs} \rangle$ under callingCtx c (caller's context)

4 // (4) Add $\langle [c'], \text{rhs} \rangle \equiv \langle [c], \text{lhs} \rangle$ into the solver.

5   return true;

Example 5: Interprocedural

Verifying ICFG path: $0 \to 4 \to 5 \to 1 \to 2 \to 3 \to 6 \to 7 \to 1 \to 2 \to 3 \to 8 \to 9 \to svf\_assert$

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\mathtt{Var21} \equiv 0 \land \mathtt{Var14} \equiv 3 \land \ldots$
ICFGNode 5	$\land \quad \langle[\mathtt{ICFGNode5}], \mathtt{Var7}\rangle \equiv \mathtt{Var14}$
ICFGNode 3	$\land \quad \langle[\mathtt{ICFGNode5}], \mathtt{Var6}\rangle \equiv \langle[\mathtt{ICFGNode5}], \mathtt{Var7}\rangle$
ICFGNode 6	$\land \quad \mathtt{Var13} \equiv \langle[\mathtt{ICFGNode5}], \mathtt{Var6}\rangle$

```
------------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
ValVar13                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   → callEdge
       → retEdge

Algorithm 24: `handleCall(callEdge)`

1 // (1) For each CallPE lhs = rhs: retrieve rhs expr ⟨[c], rhs⟩ under callingCtx c (caller's context)

2 // (2) pushCallingCtx using current calliste ICFGNode; Note that pushCallingCtx will make c become c'. ;

3 // (3) For each CallPE lhs = rhs: retrieve lhs expr ⟨[c'], lhs⟩ under callingCtx c' (callee's context) ;

4 // (4) For each CallPE lhs = rhs: add ⟨[c], rhs⟩ ≡ ⟨[c'], lhs⟩ into the solver.

5 return true;

Example 5: Interprocedural

Verifying ICFG path: $0 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow$
$6 \rightarrow 7 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9 \rightarrow$ *svf_assert*

Calling Context

After processing the call edge
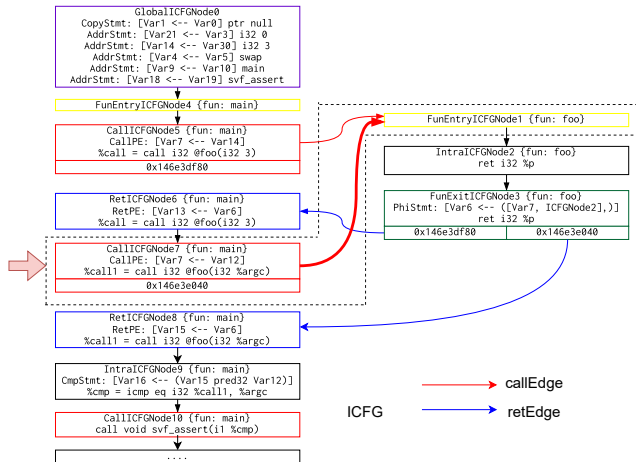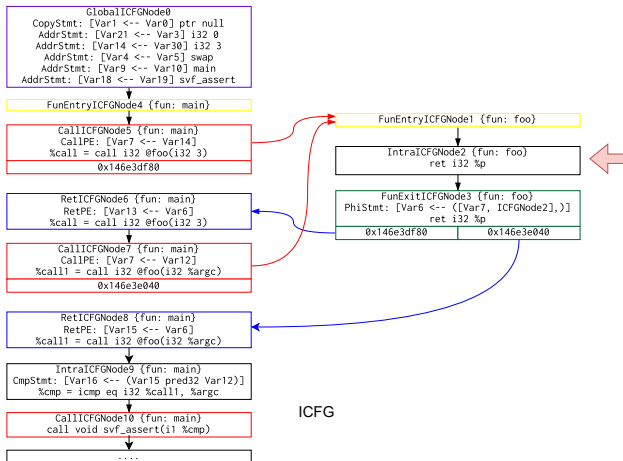
`CallCFGNode7 → FunEntryICFGNode`

Example 5: Interprocedural
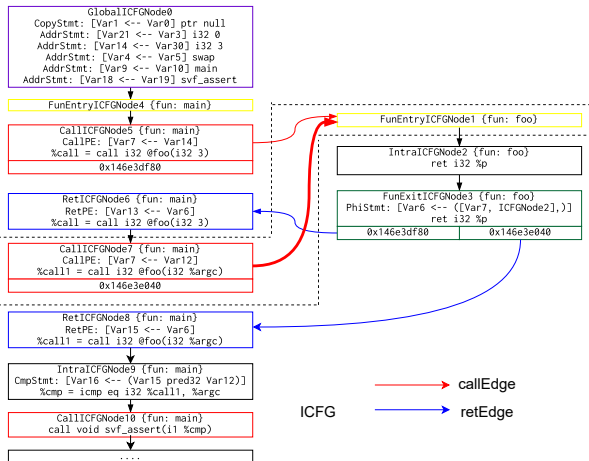
Algorithm 25: `handleCall(callEdge)`

1 // (1) For each CallPE lhs = rhs: retrieve rhs expr ⟨[c], rhs⟩ under callingCtx c (caller's context)

2 // (2) pushCallingCtx using current calliste ICFGNode; Note that pushCallingCtx will make c become c'. ;

3 // (3) For each CallPE lhs = rhs: retrieve lhs expr ⟨[c'], lhs⟩ under callingCtx c' (callee's context) ;

4 // (4) For each CallPE lhs = rhs: add ⟨[c], rhs⟩ ≡ ⟨[c'], lhs⟩ into the solver.

5 **return** true;

Example 5: Interprocedural

Verifying ICFG path: $0 \to 4 \to 5 \to 1 \to 2 \to 3 \to 6 \to 7 \to 1 \to 2 \to 3 \to 8 \to 9 \to svf\_assert$

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var21 \equiv 0 \land Var14 \equiv 3 \land \dots$
ICFGNode 5	$\land \ \langle [ICFGNode5], Var7 \rangle \equiv Var14$
ICFGNode 3	$\land \ \langle [ICFGNode5], Var6 \rangle \equiv \langle [ICFGNode5], Var7 \rangle$
ICFGNode 6	$\land \ Var13 \equiv \langle [ICFGNode5], Var6 \rangle$
ICFGNode 7	$\land \ \langle [ICFGNode7], Var7 \rangle \equiv Var12$

```
------------SVFVar and Value------------
                  ...
ValVar21              Value: 0
ValVar14              Value: 3
                  ...
ValVar13              Value 3
ValVar12              Value: 0
ValVar7               Value: 0
                  ...
----------------------------------------
```

Note: SVFVars and their values in table are under the calling context [`CallICFGNode7`].

`ValVar12` is uninitialized, thus evaluated as 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

Verifying ICFG path: $0 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9 \rightarrow$ *svf_assert*
`ret i32 %p` instruction.
Nothing needs to be done.

Example 5: Interprocedural

Algorithm 26: 3 handlePhi(edge)

1   res ← getZ3Expr(phi.getResID());
2   opINodeFound ← false;
3 **for** i ← 0 *to* phi.getOpVarNum() − 1 **do**
4   **if** edge.srcNode() *postdominates*
      phi.getOpICFGNode(i) **then**
5   ope ← getZ3Expr(phi.getOpVar(i).getId());
6   addToSolver(res == ope);
7   opINodeFound ← true;

Example 5: Interprocedural

ICFG

Verifying ICFG path: $0 \to 4 \to 5 \to 1 \to 2 \to 3 \to 6 \to 7 \to 1 \to 2 \to 3 \to 8 \to 9 \to svf\_assert$

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var21 \equiv 0 \wedge Var14 \equiv 3 \wedge \ldots$
ICFGNode 5	$\wedge \quad \langle[\text{ICFGNode5}], Var7\rangle \equiv Var14$
ICFGNode 3	$\wedge \quad \langle[\text{ICFGNode5}], Var6\rangle \equiv \langle[\text{ICFGNode5}], Var7\rangle$
ICFGNode 6	$\wedge \quad Var13 \equiv \langle[\text{ICFGNode5}], Var6\rangle$
ICFGNode 7	$\wedge \quad \langle[\text{ICFGNode7}], Var7\rangle \equiv Var12$
ICFGNode 3	$\wedge \quad \langle[\text{ICFGNode7}], Var6\rangle \equiv \langle[\text{ICFGNode7}], Var7\rangle$

```
------------SVFVar and Value------------
                    ...
ValVar21                    Value: 0
ValVar14                    Value: 3
                    ...
ValVar13                    Value: 3
ValVar12                    Value: 0
ValVar7                     Value: 0
ValVar6                     Value: 0
                    ...
----------------------------------------
```

Note: SVFVars and their values in table are under the calling context [`CallICFGNode7`].

Example 5: Interprocedural
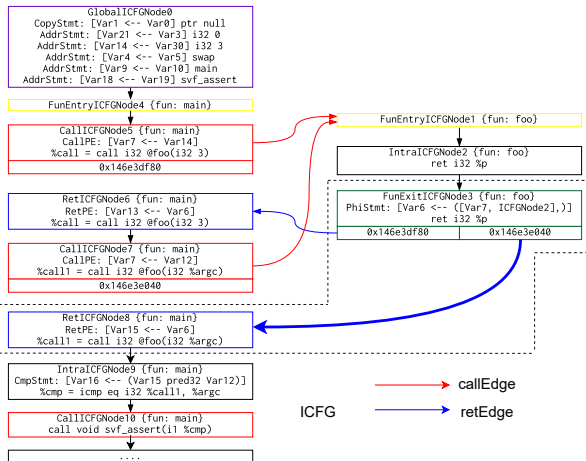
Algorithm 27: `handleRet(retEdge)`
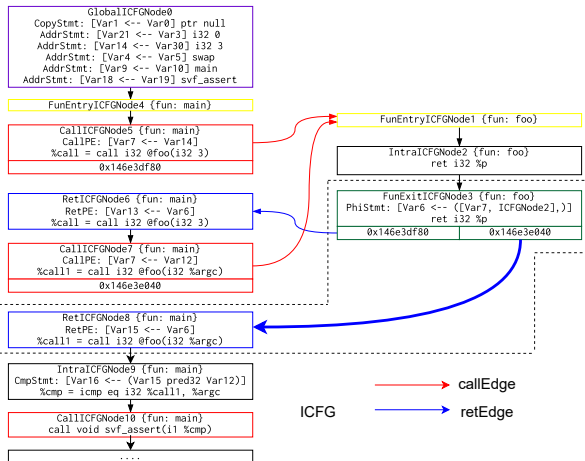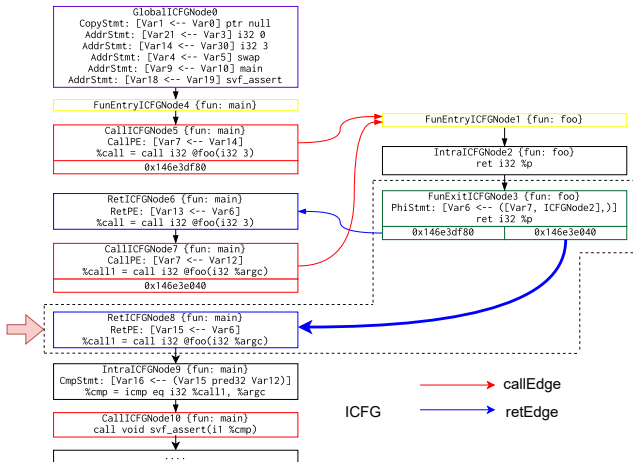
1 `// (1) For each RetPE lhs = rhs:  retrieve rhs`
`expr ⟨[c'], rhs⟩ under callingCtx c' (callee's`
`context);`

2 `// (2) popCallingCtx(); This will make c' become`
`c (caller's context)`

3 `// (3) For each RetPE lhs = rhs:  retrieve lhs`
`expr ⟨[c], lhs⟩ under callingCtx c (caller's`
`context)`

4 `// (4) Add ⟨[c'], rhs⟩ ≡ ⟨[c], lhs⟩ into the solver.`

5 `return true;`

Note: retPE.getRHSVarID() returns ValVar6
getZ3Expr(ValVar6) binds ValVar6 with the current
callingCtx [ICFGNode7] and returns the Z3 expression for
⟨[ICFGNode7], Var6⟩

Example 5: Interprocedural
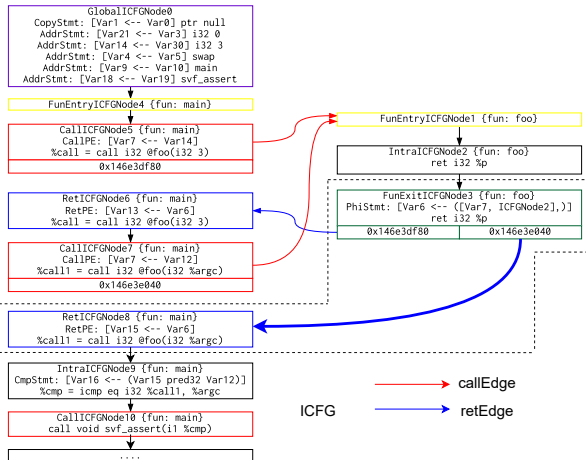
Algorithm 28: `handleRet(retEdge)`

1 `// (1) For each RetPE lhs = rhs: retrieve rhs expr ⟨[c'], rhs⟩ under callingCtx c' (callee's context);`

2 `// (2) popCallingCtx(); This will make c' become c (caller's context)`

3 `// (3) For each RetPE lhs = rhs: retrieve lhs expr ⟨[c], lhs⟩ under callingCtx c (caller's context)`

4 `// (4) Add ⟨[c'], rhs⟩ ≡ ⟨[c], lhs⟩ into the solver.`

5 `return true;`
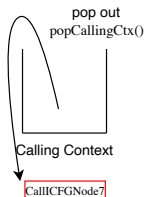
After processing the return edge

`FunExitICFGNode3 → RetICFGNode8`

Example 5: Interprocedural

Algorithm 29: `handleRet(retEdge)`

1 // (1) For each RetPE lhs = rhs: retrieve rhs expr $\langle[c'], rhs\rangle$ under callingCtx c' (callee's context);

2 // (2) popCallingCtx(); This will make c' become c (caller's context)

3 // (3) For each RetPE lhs = rhs: retrieve lhs expr $\langle[c], lhs\rangle$ under callingCtx c (caller's context)

4 // (4) Add $\langle[c'], rhs\rangle \equiv \langle[c], lhs\rangle$ into the solver.

5   return true;

Example 5: Interprocedural

Verifying ICFG path: 0 → 4 → 5 → 1 → 2 → 3 → 6 → 7 → 1 → 2 → 3 → 8 → 9 → svf_assert

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var21 \equiv 0 \wedge Var14 \equiv 3 \wedge \ldots$
ICFGNode 5	$\wedge \ \langle[ICFGNode5],Var7\rangle \equiv Var14$
ICFGNode 3	$\wedge \ \langle[ICFGNode5],Var6\rangle \equiv \langle[ICFGNode5],Var7\rangle$
ICFGNode 6	$\wedge \ Var13 \equiv \langle[ICFGNode5],Var6\rangle$
ICFGNode 7	$\wedge \ \langle[ICFGNode7],Var7\rangle \equiv Var12$
ICFGNode 3	$\wedge \ \langle[ICFGNode7],Var6\rangle \equiv \langle[ICFGNode7],Var7\rangle$
ICFGNode 8	$\wedge \ Var15 \equiv \langle[ICFGNode7],Var6\rangle$

```
------------SVFVar and Value------------
                     ...
ValVar21                Value: 0
ValVar14                Value: 3
                     ...
ValVar13                Value: 3
ValVar12                Value: 0
ValVar15                Value: 0
                     ...
----------------------------------------
```

Example 5: Interprocedural

Verifying ICFG path: 0 → 4 → 5 → 1 → 2 → 3 →
6 → 7 → 1 → 2 → 3 → 8 → 9 → svf_assert

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$Var21 \equiv 0 \wedge Var14 \equiv 3 \wedge \ldots$
ICFGNode 5	$\wedge \ \langle [ICFGNode5], Var7 \rangle \equiv Var14$
ICFGNode 3	$\wedge \ \langle [ICFGNode5], Var6 \rangle \equiv \langle [ICFGNode5], Var7 \rangle$
ICFGNode 6	$\wedge \ Var13 \equiv \langle [ICFGNode5], Var6 \rangle$
ICFGNode 7	$\wedge \ \langle [ICFGNode7], Var7 \rangle \equiv Var12$
ICFGNode 3	$\wedge \ \langle [ICFGNode7], Var6 \rangle \equiv \langle [ICFGNode7], Var7 \rangle$
ICFGNode 8	$\wedge \ Var15 \equiv \langle [ICFGNode7], Var6 \rangle$
ICFGNode 9	$\wedge \ Var16 \equiv ite(Var15 \equiv Var12, 1, 0)$

```
------------SVFVar and Value------------
                     ...
ValVar13               Value: 3
ValVar12               Value: 0
ValVar15               Value: 0
ValVar16               Value: 1
                     ...
----------------------------------------
```

Example 5: Interprocedural

Verifying ICFG path: $0 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9 \rightarrow$ *svf_assert*

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\text{Var21} \equiv 0 \wedge \text{Var14} \equiv 3 \wedge \ldots$
ICFGNode 5	$\wedge \ \langle [\text{ICFGNode5}], \text{Var7} \rangle \equiv \text{Var14}$
ICFGNode 3	$\wedge \ \langle [\text{ICFGNode5}], \text{Var6} \rangle \equiv \langle [\text{ICFGNode5}], \text{Var7} \rangle$
ICFGNode 6	$\wedge \ \ \text{Var13} \equiv \langle [\text{ICFGNode5}], \text{Var6} \rangle$
ICFGNode 7	$\wedge \ \ \langle [\text{ICFGNode7}], \text{Var7} \rangle \equiv \text{Var12}$
ICFGNode 3	$\wedge \ \langle [\text{ICFGNode7}], \text{Var6} \rangle \equiv \langle [\text{ICFGNode7}], \text{Var7} \rangle$
ICFGNode 8	$\wedge \ \ \text{Var15} \equiv \langle [\text{ICFGNode7}], \text{Var6} \rangle$
ICFGNode 9	$\wedge \ \ \text{Var16} \equiv \text{ite}(\text{Var15} \equiv \text{Var12}, 1, 0)$
ICFGNode 10	$\wedge \ \ \text{Var16} \equiv 0$ (negation of the assert condition)

Solver yields **UNSAT**, meaning no counter example.
The assertion is verified successfully!!

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