Automated Assertion Verification via Symbolic Execution

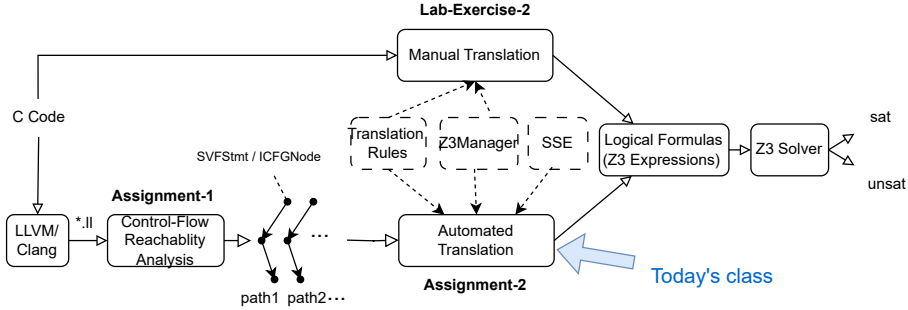
(Week 7)

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 for translating branches and calls/returns.**
- **We will showcase branches and interprocedural examples.**

Branches and Path Feasibility

Branch feasibility contributes to path feasibility.

A branch edge is feasible only if its expected successor value is consistent with the constraints already collected.

Branch Condition and Feasibility

Branch edge feasibility

For a branch edge with condition *cond* and edge value *succ*:

$$\Phi' = \Phi \wedge (cond \equiv succ)$$

- If SAT(Φ'), the edge is feasible and Φ' becomes the new path condition.
- If UNSAT(Φ'), no concrete execution can follow this path prefix, so the path can be discarded.
- This is why branch translation is both a **constraint-generation** step and a **path-pruning** step.

Implementation Roadmap: 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          if handleCall(call) == false then
7              return false
8      else if ret ← dyn_cast⟨RetEdge⟩(edge) then
9          if handleRet(ret) == false then
10             return false
11 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)`

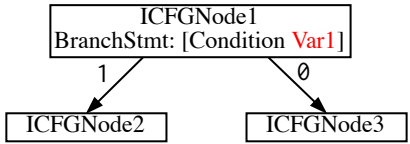
```
1  // Your code starts here.;
2
3  // For each CallPE lhs = rhs,
4  // (1) Save ⟨[c], rhs⟩ before calling
   pushCallingCtx; then retrieve ⟨[c′], lhs⟩ in the
   callee context.
5  // (2) Add ⟨[c], rhs⟩ ≡ ⟨[c′], lhs⟩ to the solver.
```

Algorithm 4: `handleRet(retEdge)`

```
1  // Your code starts here.;
2
3  // For each RetPE lhs = rhs,
4  // (1) Save ⟨[c′], rhs⟩ before calling popCallingCtx;
   then retrieve ⟨[c], lhs⟩ in the caller context.
5  // (2) Add ⟨[c′], rhs⟩ ≡ ⟨[c], lhs⟩ to the solver.
```

The Variable of the Branch Condition: `getCondition()`

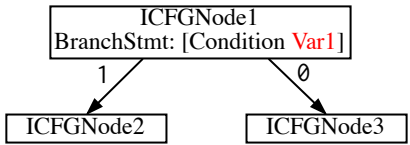
Given an `if/else` branch in the ICFG:



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

The ICFGEdge's Expected Value: `getSuccessorCondValue()`

Given an `if/else` branch in the ICFG:



- `edge` → `getSuccessorCondValue()` returns the edge's expected successor value `succCondValue`: **1** for the **then edge** and **0** for the **else edge**.
  - 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 whether the existing path constraints conjoined with `Var1` == `succCondValue` are `sat`.

Example 4: Branches - Source and LLVM IR

```c
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 sgt 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 sge i32 %y.0, %add1
  call void @svf_assert(i1 zeroext %cmp2)
  ret void
}
```

LLVM IR

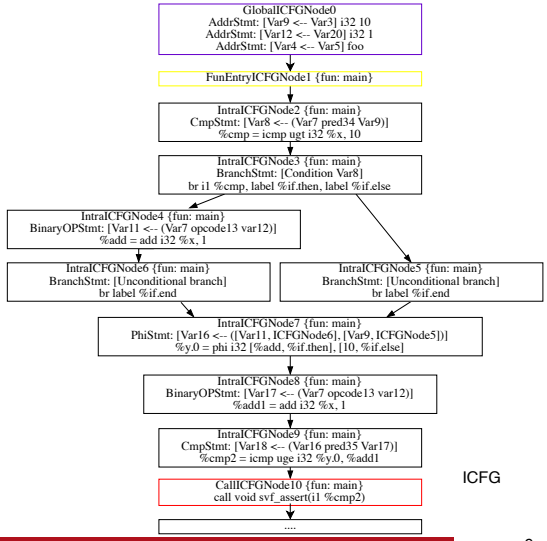
Example 4: Branches - ICFG

```
1  define void @main(i32 %x) #0 {
2  entry:
3    %cmp = icmp sgt 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 sge i32 %y.0, %add1
17   call void @svf_assert(i1 zeroext %cmp2)
18   ret void
19 }
```

LLVM IR

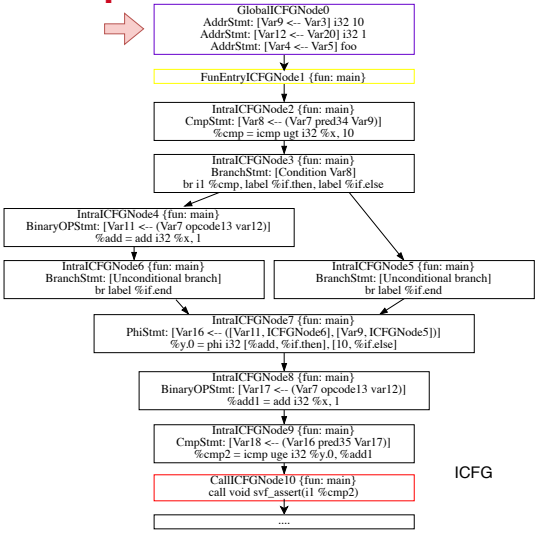
Two ICFG paths:

- `then` branch:
  0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → *svf_assert*
- `else` branch:
  0 → 1 → 2 → 3 → 5 → 7 → 8 → 9 → *svf_assert*



ICFG

Example 4: Branches - Start with Global Constants



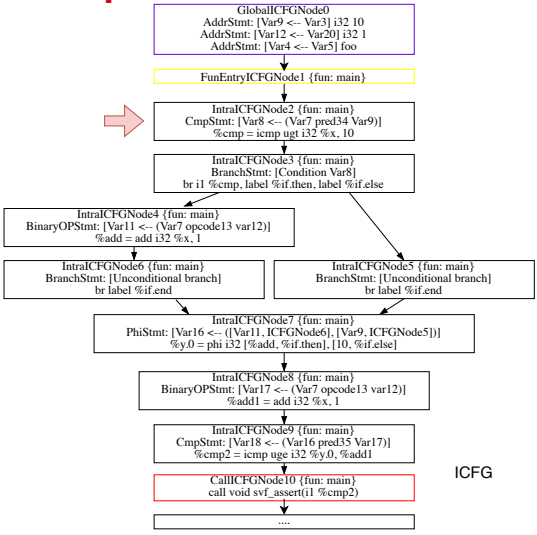
ICFG

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

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	`Var9 ≡ 10 ∧ Var12 ≡ 1 ∧ Var4 ≡ 0x7f000005`

```
------------One Satisfying Model------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
                  ...
--------------------------------------------
```

Example 4: Branches - Translate the Comparison



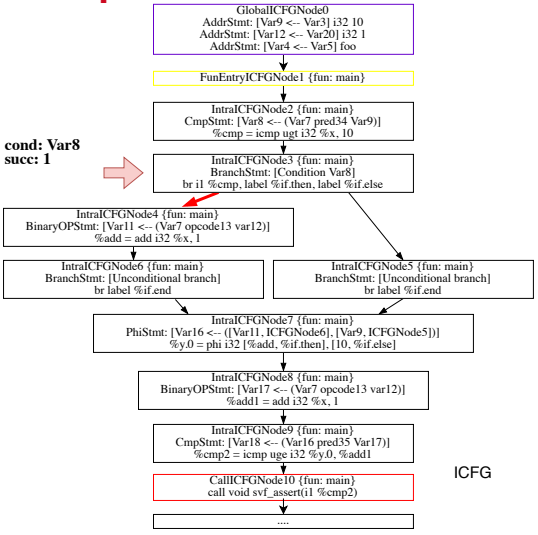
ICFG

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

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

```
------------One Satisfying Model------------
ObjVar5 (0x7f000005)    Value: NULL
ValVar4                 Value: 0x7f000005
ValVar9                 Value: 10
ValVar12                Value: 1
ValVar7                 Value: 11
ValVar8                 Value: 1
                  ...
--------------------------------------------
```

Example 4: Branches - Branch Feasibility Rule



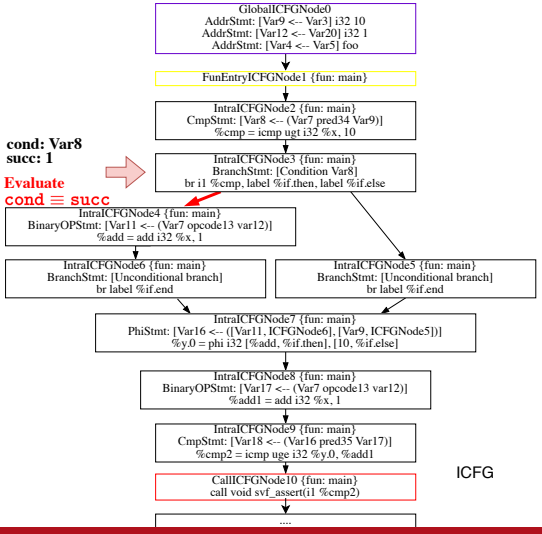
Algorithm 5: 3 `handleIntra(intraEdge)`

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

Algorithm 6: `handleBranch(intraEdge)`

```
1  cond = ... // Retrieve branch/switch cond variable;
2  succ = ... // Edge's successor condition/case value;
3  // Evaluate branch feasibility (via solver.push()/pop());
4  // Check SAT(path constraints ∧ 'cond == succ'); UNSAT means the edge is infeasible;
5  if the branch edge is infeasible then
6      return false;
7  else
8      // Add (cond≡succ) to the solver;
9      // Remove cond constraints from solver when backtracking in ICFG reachability analysis.;
10     return true;
```

Example 4: Branches - Query the Then Edge



ICFG

Algorithm 7: 3 `handleIntra(intraEdge)`

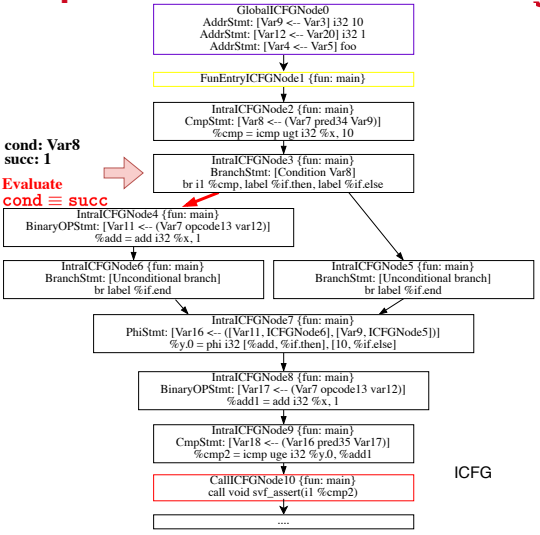
```
2  if intraEdge.getCondition() &&
     !handleBranch(intraEdge) then
4    return false;
6  else
8    handleNonBranch(edge);
```

Algorithm 8: `handleBranch(intraEdge)`

```
1  cond = ... // Retrieve branch/switch cond variable;
2  succ = ... // Edge's successor condition/case value;
3  // Evaluate branch feasibility (via
     solver.push()/pop());
4  // Check SAT(path constraints ∧ 'cond == succ');
     UNSAT means the edge is infeasible;
5  if the branch edge is infeasible then
6    return false;
7  else
8    // Add (cond≡succ) to the solver.;
9    // Remove cond constraints from solver when
     backtracking in ICFG reachability analysis.;
10   return true;
```

Note: `getSolver().push()/pop()` creates a new stack frame for evaluating the newly added ES constraints.

Example 4: Branches - Then Edge is Feasible



ICFG

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

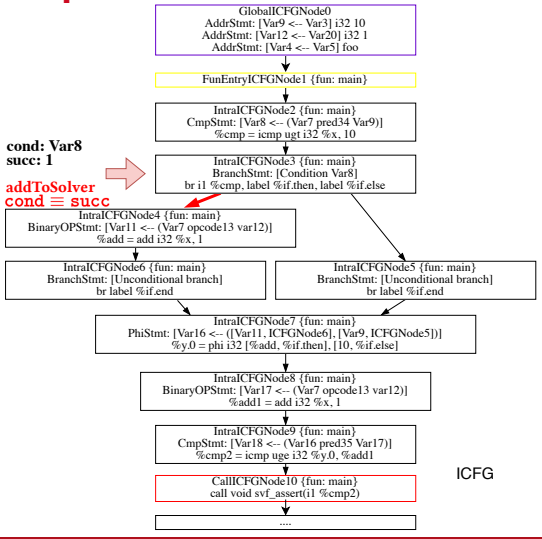
ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\texttt{Var9} \equiv 10 \wedge \texttt{Var12} \equiv 1 \wedge \texttt{Var4} \equiv \texttt{0x7f000005}$
ICFGNode 2	$\wedge \quad \texttt{Var8} \equiv \texttt{ite}(\texttt{Var7} > \texttt{Var9}, 1, 0)$
ICFGEdge 3 → 4	

The constraint $\texttt{Var8} \equiv 1$ is evaluated to be SAT.

The conditional ICFGEdge [ICFGNode3 → ICFGNode4] is feasible.

```
-------------One Satisfying Model-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 11
ValVar8                  Value: 1
                  ...
----------------------------------------------
```

Example 4: Branches - Add the Branch Constraint



ICFG

Algorithm 9: 3 `handleIntra(intraEdge)`

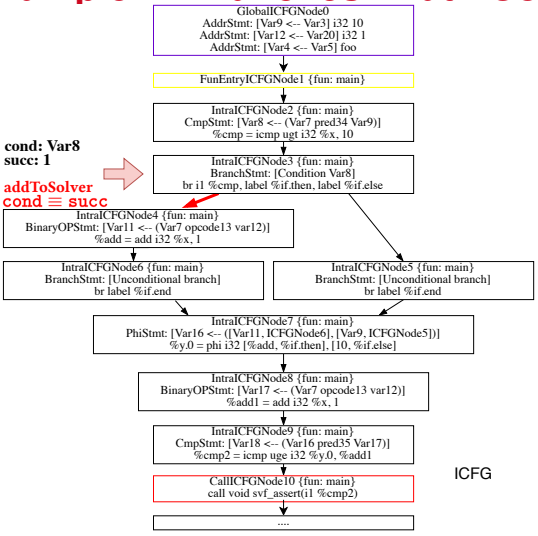
```
1  if intraEdge.getCondition() && !handleBranch(intraEdge) then
2      return false;
3  else
4      handleNonBranch(edge);
```

Algorithm 10: `handleBranch(intraEdge)`

```
1  cond = ... // Retrieve branch/switch cond variable;
2  succ = ... // Edge's successor condition/case value;
3  // Evaluate branch feasibility;
4  // Check SAT(path constraints ∧ 'cond == succ');
   UNSAT means the edge is infeasible;
5  if the branch edge is infeasible then
6      return false;
7  else
8      // Add (cond≡succ) to the solver.;
9      // Remove cond constraints from solver when
       backtracking in ICFG reachability analysis.;
10     return true;
```

Note: `res` is `sat`, so the conditional ICFGEdge [`ICFGNode3` → `ICFGNode4`] is **feasible**.

Example 4: Branches - Path Condition after Branch Evaluation



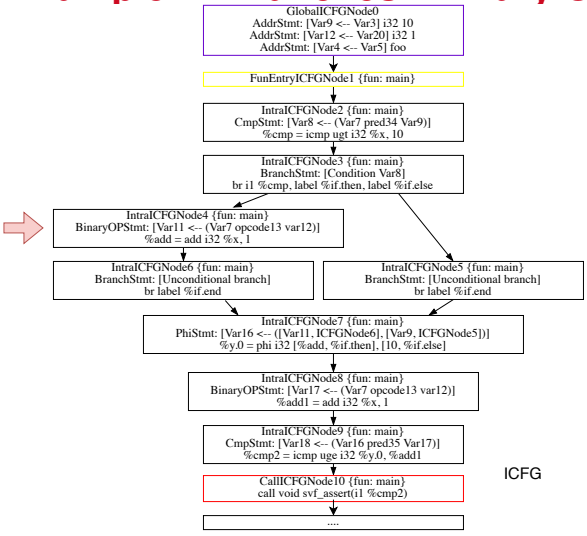
ICFG

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

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

```
-------------One Satisfying Model-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 11
ValVar8                  Value: 1
                  ...
----------------------------------------------
```

Example 4: Branches - Binary Operator



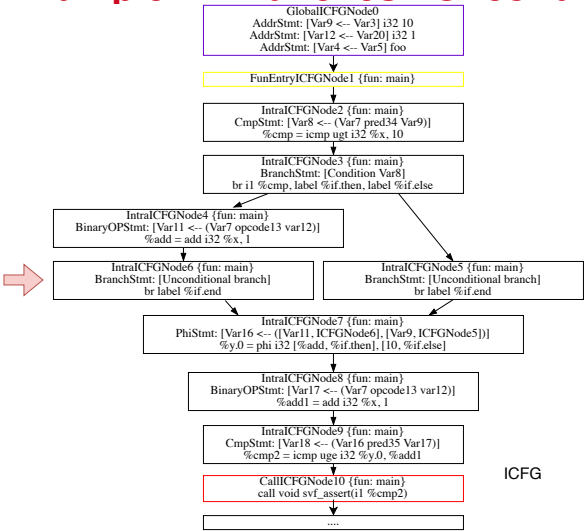
ICFG

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

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

```
------------One Satisfying Model------------
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 - Unconditional Branch

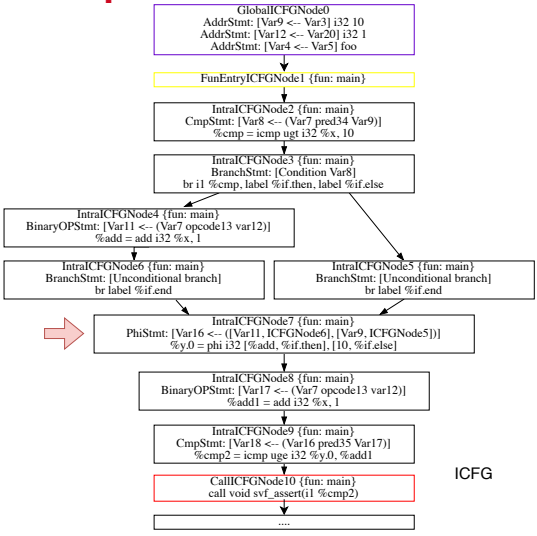


ICFG

Algorithm 11: 3 `handleIntra(intraEdge)`

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

Example 4: Branches - PHI Statement and Operand



ICFG

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

Algorithm 12: 3 `handlePhi(edge)`

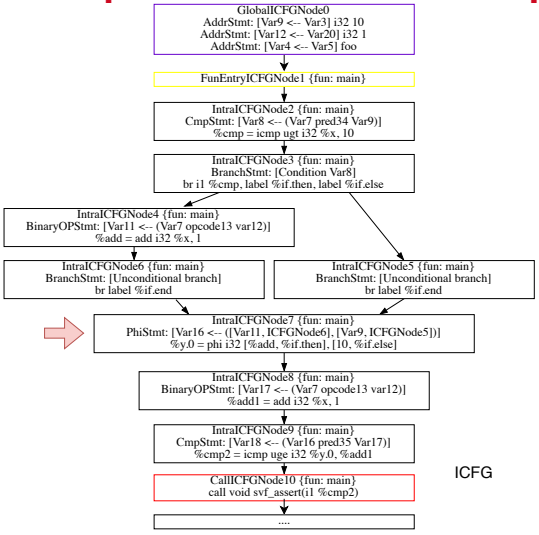
```
1 res ← getZ3Expr(phi.getResID());;
2 for i ← 0 to phi.getOpVarNum() − 1 do
3     if edge.srcNode() postdominates phi.getOpICFGNode(i)
        then
4         // Get def through the edge on current path
          ope ← getZ3Expr(phi.getOpVar(i).getId());
5         addToSolver(res == ope);
6         break;
```

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's definition (not necessary at immediate predecessor).

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

Example 4: Branches - PHI Operand



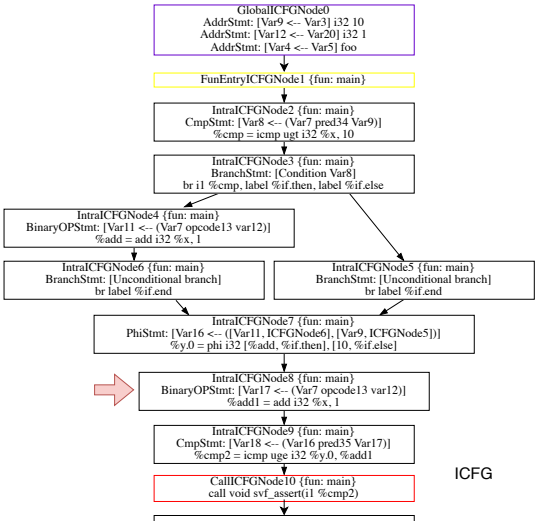
ICFG

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

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

```
------------One Satisfying Model------------
                    ...
ValVar9                 Value: 10
ValVar12                Value: 1
ValVar7                 Value: 11
ValVar8                 Value: 1
ValVar11                Value: 12
ValVar16                Value: 12
                    ...
--------------------------------------------
```

Example 4: Branches - Translate Binary Operator



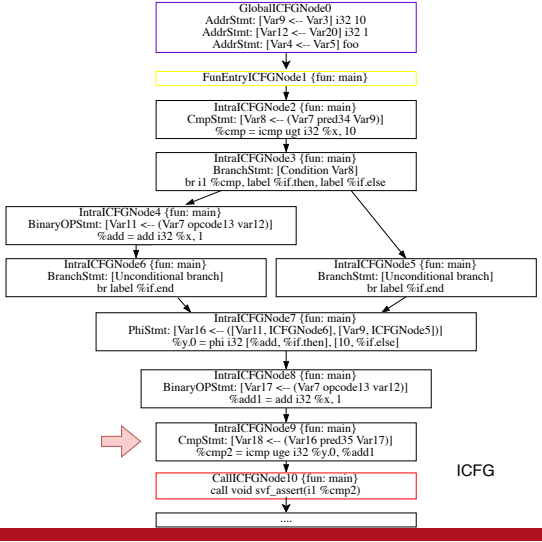
ICFG

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

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

```
-------------One Satisfying Model-------------
                      ...
ValVar9                 Value: 10
ValVar12                Value: 1
ValVar7                 Value: 11
ValVar8                 Value: 1
ValVar11                Value: 12
ValVar16                Value: 12
ValVar17                Value: 12
                      ...
```

Example 4: Branches - Translate CmpStmt



ICFG

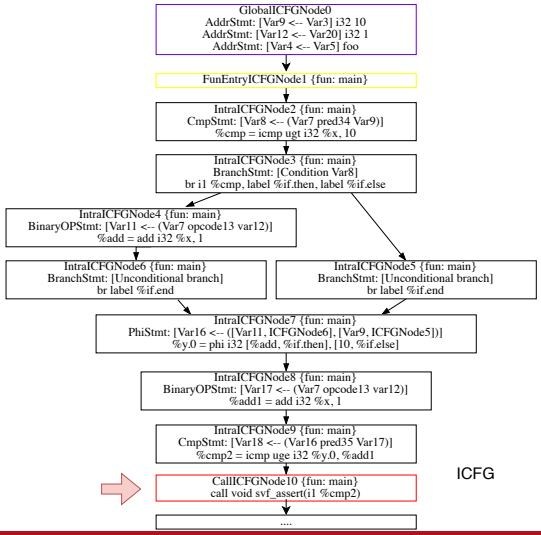
Verifying ICFG path: 0 → 1 → 2 → 3 → 4 →

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

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

```
----------One Satisfying Model----------
                    ...
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 - Translate Assertion



ICFG

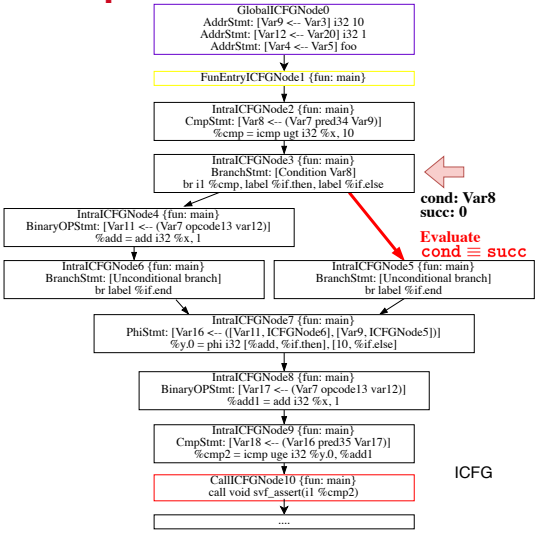
Verifying ICFG path: $0 \to 1 \to 2 \to 3 \to 4 \to 6 \to 7 \to 8 \to 9 \to$ *svf_assert* (then branch)

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

Solver yields **UNSAT** (i.e., no counter example),

Therefore, the assertion is verified.

Example 4: Branches - Else Branch



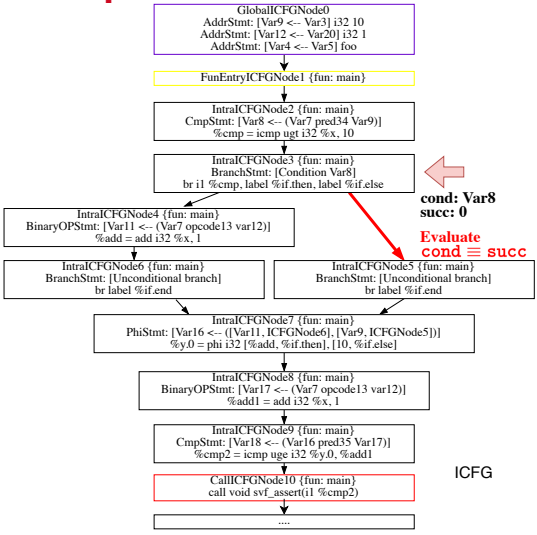
Algorithm 13: 3 `handleIntra(intraEdge)`

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

Algorithm 14: `handleBranch(intraEdge)`

```
1  cond = ... // Retrieve branch/switch cond variable;
2  succ = ... // Edge's successor condition/case value;
3  // Evaluate branch feasibility;
4  // Check SAT(path constraints ∧ 'cond == succ');
     UNSAT means the edge is infeasible;
5  if the branch edge is infeasible then
6      return false;
7  else
8      // Add (cond≡succ) to the solver.;
9      // Remove cond constraints from solver when
       backtracking in ICFG reachability analysis.;
10     return true;
```

Example 4: Branches - Check the Else Branch



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

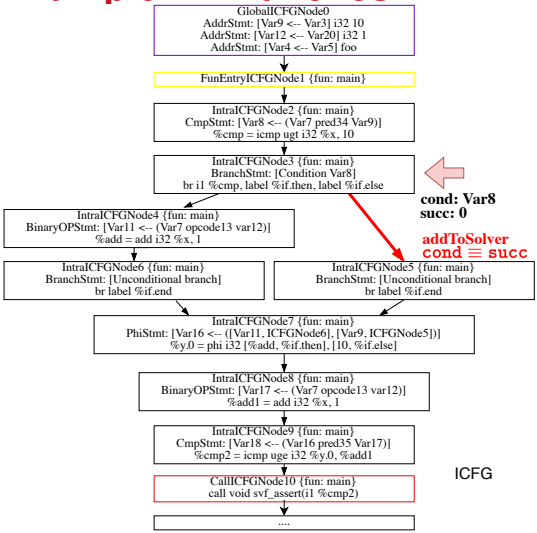
ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\texttt{Var9} \equiv 10 \wedge \texttt{Var12} \equiv 1 \wedge \texttt{Var4} \equiv \texttt{0x7f000005}$
ICFGNode 2	$\wedge \quad \texttt{Var8} \equiv \texttt{ite}(\texttt{Var7} > \texttt{Var9}, 1, 0)$
ICFGEdge 3 $\rightarrow$ 5	

The constraint $\texttt{Var8} \equiv 0$ is evaluated to be SAT

The conditional ICFGEdge [ICFGNode3 $\rightarrow$ ICFGNode5] is feasible.

```
------------One Satisfying Model------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 10
ValVar8                  Value: 0
                  ...
--------------------------------------------
```

Example 4: Branches



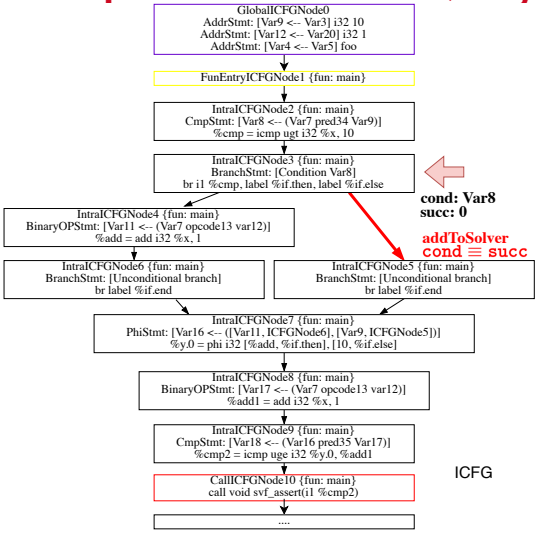
Algorithm 15: 3 `handleIntra(intraEdge)`

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

Algorithm 16: `handleBranch(intraEdge)`

```
1  cond = ... // Retrieve branch/switch cond variable;
2  succ = ... // Edge's successor condition/case value;
3  // Evaluate branch feasibility;
4  // Check SAT(path constraints ∧ 'cond == succ');
   UNSAT means the edge is infeasible;
5  if the branch edge is infeasible then
6      return false;
7  else
8      // Add (cond≡succ) to the solver.;
9      // Remove cond constraints from solver when
       backtracking in ICFG reachability analysis.;
10     return true;
```

Example 4: Branches - Query the Else Edge



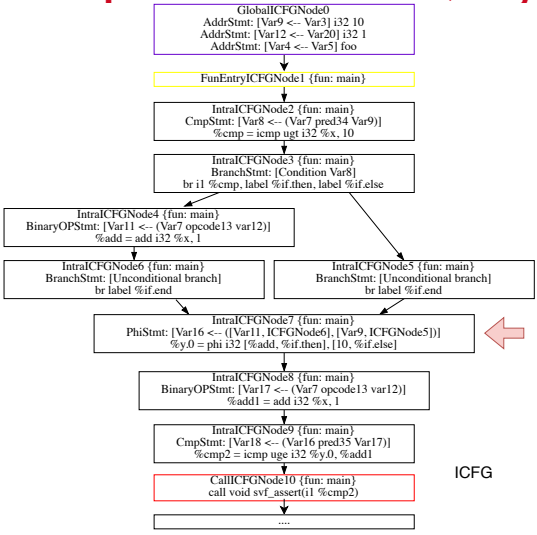
Verifying ICFG path: 0 → 1 → 2 → 3 → 5 →

7 → 8 → 9 → *svf_assert* (else branch)

ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\texttt{Var9} \equiv 10 \wedge \texttt{Var12} \equiv 1 \wedge \texttt{Var4} \equiv \texttt{0x7f000005}$
ICFGNode 2	$\wedge \quad \texttt{Var8} \equiv \texttt{ite}(\texttt{Var7} > \texttt{Var9}, 1, 0)$
ICFGEdge 3 → 5	$\wedge \quad \texttt{Var8} \equiv 0$

```
-------------One Satisfying Model-------------
ObjVar5 (0x7f000005)     Value: NULL
ValVar4                  Value: 0x7f000005
ValVar9                  Value: 10
ValVar12                 Value: 1
ValVar7                  Value: 10
ValVar8                  Value: 0
                  ...
----------------------------------------------
```

Example 4: Branches - Query the Else Edge



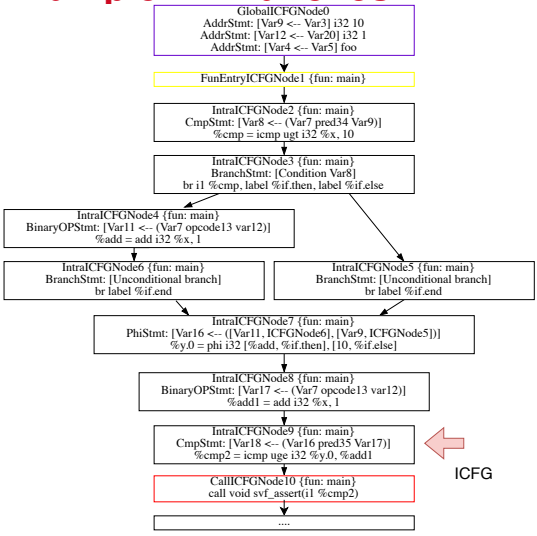
ICFG

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

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

```
-------------One Satisfying Model-------------
ObjVar5 (0x7f000005)    Value: NULL
ValVar4                 Value: 0x7f000005
ValVar9                 Value: 10
ValVar12                Value: 1
ValVar7                 Value: 10
ValVar8                 Value: 0
ValVar16                Value: 10
                  ...
----------------------------------------------
```

Example 4: Branches

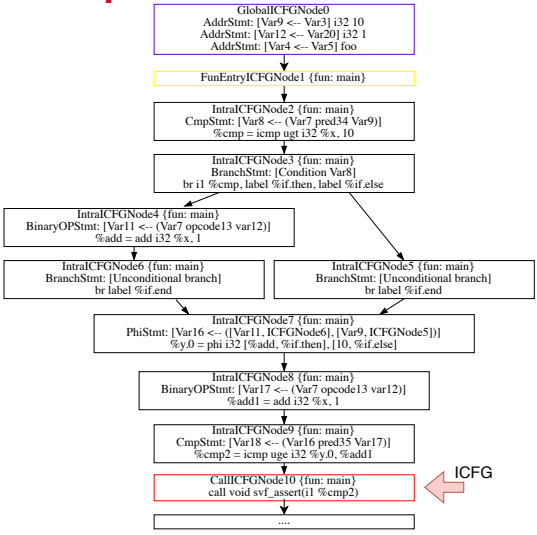


Verifying ICFG path: 0 → 1 → 2 → 3 → 5 → 7 → 8 → 9 → *svf_assert* (else branch)

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

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

Example 4: Branches - Else Branch Violates the Assertion



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

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

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

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

The assertion is violated.

Calls, Returns and Calling Contexts

Interprocedural paths need context-sensitive expressions.

The same local variable in a callee can denote different symbolic values under different callsites.

What changes from Part I?

The path condition Φ is still accumulated as before, but each expression is now keyed by both SVF variable ID and calling context κ.

`lhs` and `rhs` When Handling Calls/Returns

Let us see the example of `lhs` and `rhs` variables for call and parameter passing (i.e., `CallPE` and `RetPE`)

```
1 int foo(int x){ // CallPE: x = m;  lhs: x, rhs: m
2     int y = x;
3     return y;
4 }
5
6 main(){
7     int m = 0;
8     n = foo(m); // RetPE: n = y;   lhs: n, rhs : y
9 }
```

Parameter passing is from actual parameter `m` at the callsite (Line 8) to formal parameter `x` at the entry of `foo`. Return value passing is from return variable `y` in `foo` to `n` at the callsite (Line 8).

Why Context-Sensitive Expressions Matter

```
1  int id(int x) {
2      return x;
3  }
4
5  void main() {
6      int a = id(1);  // call1
7      int b = id(2);  // call2
8      assert(a != b);
9  }
```

- The same formal parameter `x` in `id` is used by two different calls.
- Without calling context, formulas from different invocations may be mixed.
- With context sensitivity, each expression is identified by:

$$\langle callingCtx, NodeID \rangle$$

Call context	Formula
$[call_1]$	$\langle [call_1], x \rangle \equiv 1$
$[call_2]$	$\langle [call_2], x \rangle \equiv 2$

Calling Context Switching During Path Translation

caller context **c** $\xrightarrow{\text{call edge}}$ **callee** context **c'** $= c{+}callsite$ $\xrightarrow{\text{return edge}}$ back to c

When traversing	Context operation
Call edge	`pushCallingCtx(callsite)`
Inside callee	use context-sensitive `getZ3Expr(NodeID)`
Return edge	`popCallingCtx()`

Why this matters

The same local variable in the same function may represent different symbolic values under different callsites.

Context-Sensitive `getZ3Expr(NodeID)`

- `callingCtx`: the current calling-context stack, represented as a sequence of callsites (`ICFGNode`s) during path traversal
- `pushCallingCtx(ICFGNode*)` pushes a callsite onto the current context stack
- `popCallingCtx()` removes the top callsite from the current context stack
- `getZ3Expr(NodeID)` in Assignment-2 is context-sensitive, i.e., retrieving a Z3 expression based on **the variable ID and current calling context**.

Context-Sensitive `getZ3Expr(NodeID)`

- `callingCtx`: the current calling-context stack, represented as a sequence of callsites (`ICFGNode`s) during path traversal
- `pushCallingCtx(ICFGNode*)` pushes a callsite onto the current context stack
- `popCallingCtx()` removes the top callsite from the current context stack
- `getZ3Expr(NodeID)` in Assignment-2 is context-sensitive, i.e., retrieving a Z3 expression based on **the variable ID and current calling context**.
- Interprocedural context-sensitive variable copies:
  - $\mathtt{r} = \mathtt{call\_f}(\ldots, \mathtt{q}, \ldots) \quad \mathtt{f}(\ldots, \mathtt{p}, \ldots)\{\ldots \mathtt{return}\ \mathtt{z}\}$
  - Context-insensitive Z3 formula $\mathtt{p} \equiv \mathtt{q}, \quad \mathtt{r} \equiv \mathtt{z}$
  - Context-sensitive Z3 formula (each expression corresponds to a pair) $\langle [\mathtt{c}'], \mathtt{p} \rangle \equiv \langle [\mathtt{c}], \mathtt{q} \rangle \quad \langle [\mathtt{c}], \mathtt{r} \rangle \equiv \langle [\mathtt{c}'], \mathtt{z} \rangle$, where $\mathtt{c}' \equiv \mathtt{c} + \mathtt{ICFGNode}_{\mathtt{call\_f}}$ (Assignment-2)
  - Correctly use `pushCallingCtx` and `popCallingCtx` before getting an Z3 expression.

Example 5: Interprocedural - Concrete vs Symbolic Execution

```
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 }
```

[ℓ_7]: calling context of `foo` at ℓ_7

[ℓ_8]: calling context of `foo` at ℓ_8

Example 5: Interprocedural - Concrete vs Symbolic Execution

```
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 }
```

[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 - Concrete vs Symbolic Execution

```
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 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)

`argc : getZ3Expr(argc)`

push abstract calling context (current ctx:[ℓ_7])

$\langle[\ell_7], \mathrm{p}\rangle$: 3

`x : getZ3Expr(`$\langle[\ell_7], \mathrm{p}\rangle$`)`

pop abstract calling context (current ctx:[ ])

push abstract calling context (current ctx:[ℓ_8])

$\langle[\ell_8], \mathrm{p}\rangle$: `getZ3Expr(argc)`

`y : getZ3Expr(`$\langle[\ell_8], \mathrm{p}\rangle$`)`

pop abstract calling context (current ctx:[ ])

Example 5: Interprocedural - Concrete vs Symbolic Execution

```
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 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)

`argc` : `getZ3Expr(argc)`

push abstract calling context (current ctx:$[\ell_7]$)

$\langle[\ell_7], \mathrm{p}\rangle$: 3

`x` : `getZ3Expr(`$\langle[\ell_7], \mathrm{p}\rangle$`)`

pop abstract calling context (current ctx:[ ])

push abstract calling context (current ctx:$[\ell_8]$)

$\langle[\ell_8], \mathrm{p}\rangle$: `getZ3Expr(argc)`

`y` : `getZ3Expr(`$\langle[\ell_8], \mathrm{p}\rangle$`)`

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
$\langle[\ell_7], \mathrm{p}\rangle \equiv 3 \wedge \mathrm{x} \equiv \langle[\ell_7], \mathrm{p}\rangle \wedge \langle[\ell_8], \mathrm{p}\rangle \equiv \mathtt{argc} \wedge \mathrm{y} \equiv \langle[\ell_7], \mathrm{p}\rangle \wedge \mathrm{y} \neq \mathtt{argc}$	unsat	$\varnothing$

`foo`'s argument `p` needs to be **differentiated and renamed** as $\langle[\ell_7], \mathrm{p}\rangle$ and $\langle[\ell_8], \mathrm{p}\rangle$ 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 - Context Operations

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

- Obtain a Z3 expression from an 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 – Source and LLVM IR

```
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

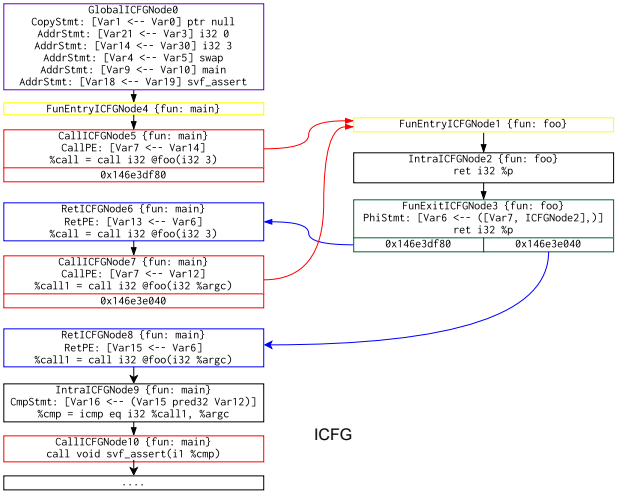
```
1  define i32 @foo(i32 %p) #0 {
2  entry:
3    ret i32 %p
4  }
5
6  define i32 @main(i32 %argc) #0 {
7  entry:
8    %call = call i32 @foo(i32 3)
9    %call1 = call i32 @foo(i32 %argc)
10   %cmp = icmp eq i32 %call1, %argc
11   call void @svf_assert(i1 zeroext %cmp)
12   ret i32 0
13 }
```

LLVM IR

Example 5: Interprocedural – Two Calls, One Callee

```
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

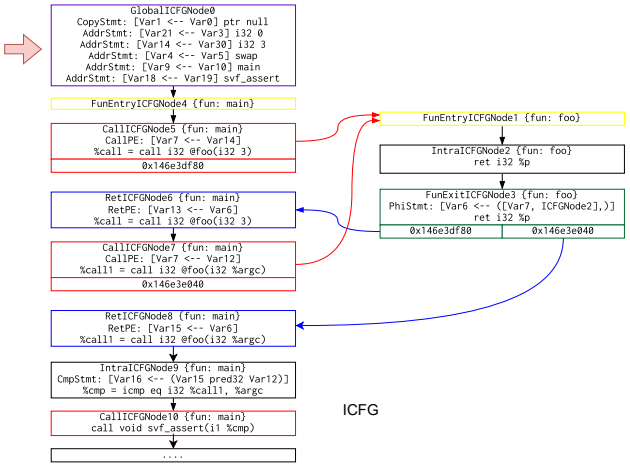


ICFG

Example 5: Interprocedural - Initialize the Path Condition

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 ≡ 0 ∧ Var14 ≡ 3 ∧ ...

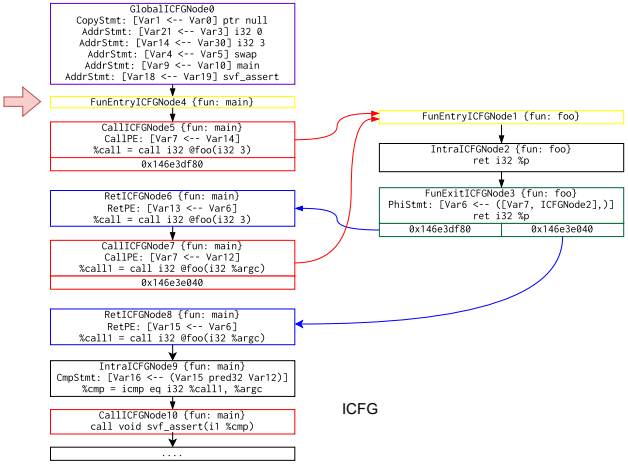


ICFG

```
-------------One Satisfying Model-------------
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 – Call/Return Roadmap

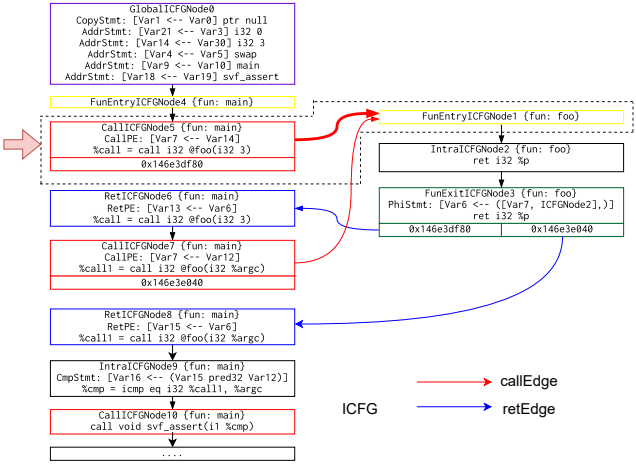


ICFG

Algorithm 17: 2 `translatePath(path)`

```
2   foreach edge ∈ path do
4       if IntraEdge ← dyn_cast⟨IntraCFGEdge⟩(edge)
          then
6           if handleIntra(IntraEdge) == false then
8               return false;
10      else if CallEdge ← dyn_cast⟨CallCFGEdge⟩(edge)
          then
12          handleCall(CallEdge);
14      else if RetEdge ← dyn_cast⟨RetCFGEdge⟩(edge)
          then
16          handleRet(RetEdge);
18      else
20          assert(false &&"what other edges we have?");
21  return true;
```

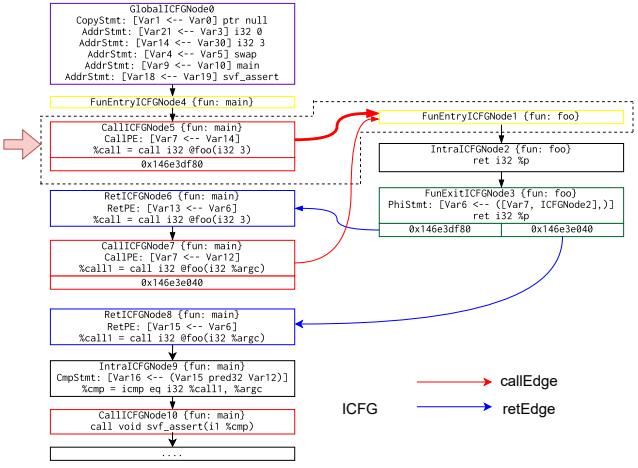
Example 5: Interprocedural - First Call Enters foo



Algorithm 18: `handleCall(callEdge)`

```
1 // (1) For each CallPE lhs = rhs, retrieve and
  save the RHS expression ⟨[c], rhs⟩ in caller
  context c.
2 // (2) Call pushCallingCtx with the current
   callsite ICFG node; this changes the context
   from c to c'.  ;
3 // (3) For each CallPE lhs = rhs, retrieve the
  LHS expression ⟨[c'], lhs⟩ in callee context c'.  ;
4 // (4) For each CallPE lhs = rhs, add
  ⟨[c], rhs⟩ ≡ ⟨[c'], lhs⟩ to the solver.
5 return true;
```

Example 5: Interprocedural - Push Context for Second Call



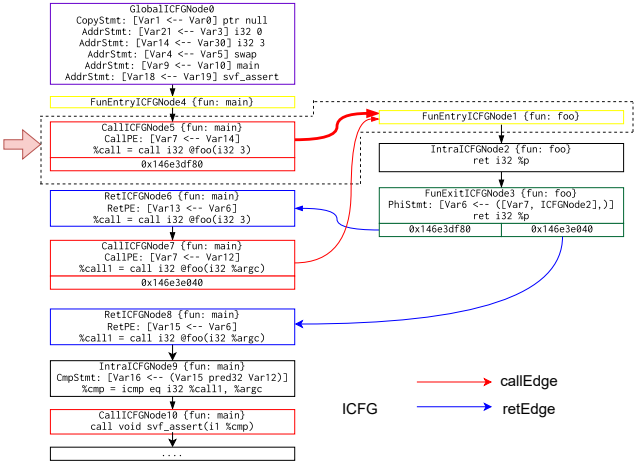
Verifying ICFG path: 0 → 4 → 5 →1 → 2 → 3 →
6 → 7 → 1 → 2 → 3 → 8 → 9 → *svf_assert*

Calling Context

After processing the call edge

`CallICFGNode5` → `FunEntryICFGNode1`

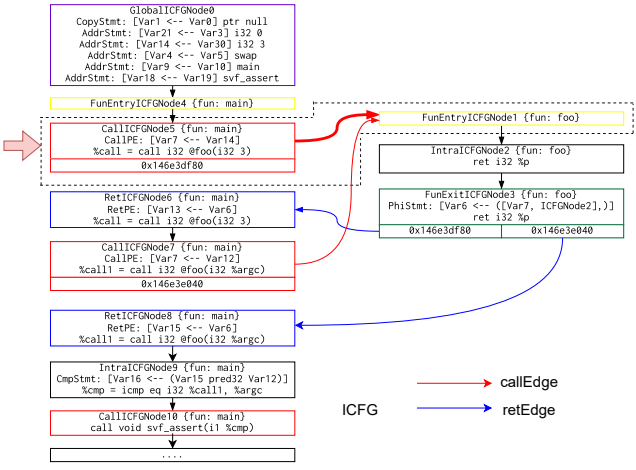
Example 5: Interprocedural - Bind Formal/Actual Parameters



Algorithm 19: `handleCall(callEdge)`

```
1 // (1) For each CallPE lhs = rhs, retrieve and
  save the RHS expression ⟨[c], rhs⟩ in caller
  context c.
2 // (2) Call pushCallingCtx with the current
  callsite ICFG node; this changes the context
  from c to c'.  ;
3 // (3) For each CallPE lhs = rhs, retrieve the
  LHS expression ⟨[c'], lhs⟩ in callee context c'.  ;
4 // (4) For each CallPE lhs = rhs, add
    ⟨[c], rhs⟩ ≡ ⟨[c'], lhs⟩ to the solver.
5 return true;
```

Example 5: Interprocedural - Entering into the Callee



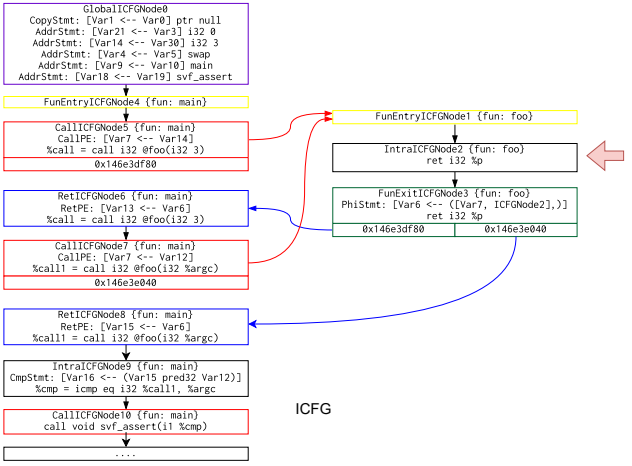
Verifying ICFG path: $0 \to 4 \to 5 \to 1 \to 2 \to 3 \to 6 \to 7 \to 1 \to 2 \to 3 \to 8 \to 9 \to$ *svf_assert*

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

```
-------------One Satisfying Model-------------
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: The satisfying-model table is printed under the calling context [`CallICFGNode5`]

Example 5: Interprocedural - Returning to Caller

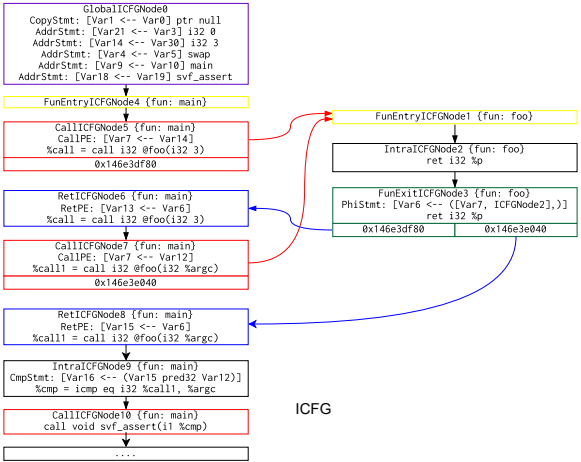


ICFG

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 - PHI as the Collection of Returns

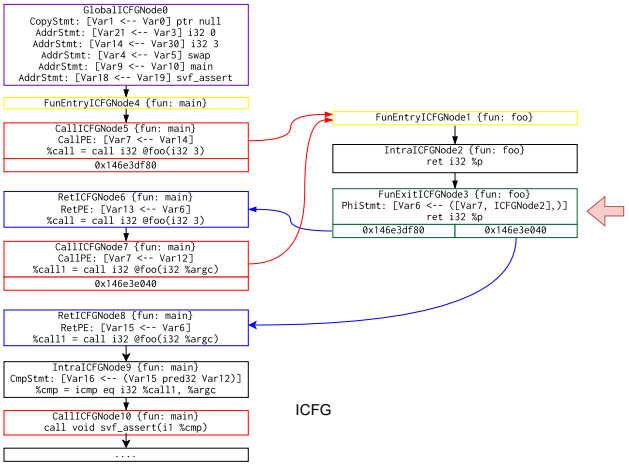


ICFG

Algorithm 20: 3 `handlePhi(edge)`

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

Example 5: Interprocedural - Path Conditions After Return



ICFG

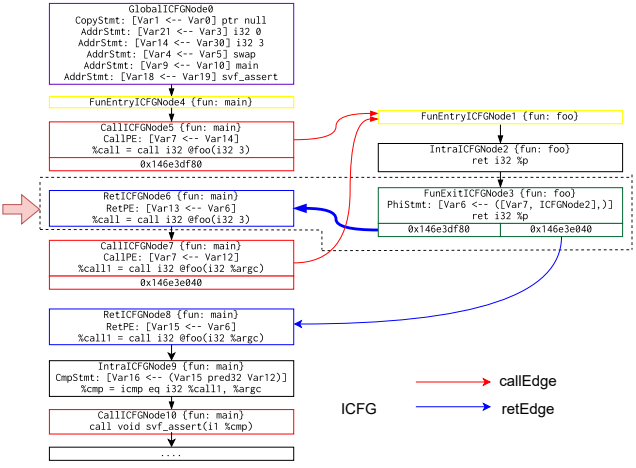
Verifying ICFG path: 0 → 4 → 5 → 1 → 2 → 3 → 6 → 7 → 1 → 2 → 3 → 8 → 9 → *svf_assert*

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

```
------------One Satisfying Model------------
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: The satisfying-model table is printed under the calling context [CallICFGNode5]

Example 5: Interprocedural - Bind Formal/Actual Return

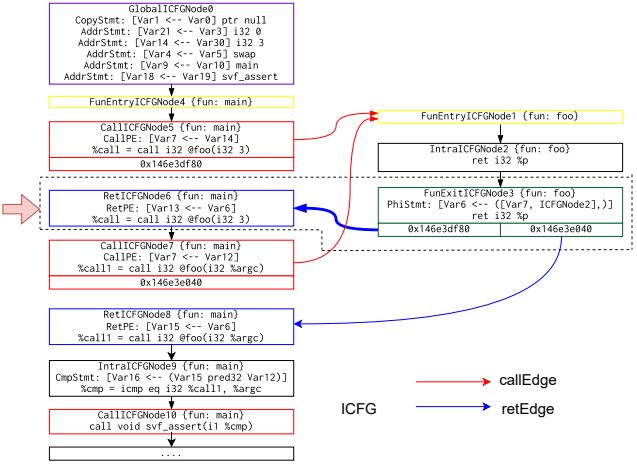


Algorithm 21: `handleRet(retEdge)`

```
1 // (1) For each RetPE lhs = rhs, retrieve and
    save the RHS expression ⟨[c′],rhs⟩ in callee
    context c'.
2 // (2) Call popCallingCtx() to restore caller
    context c.
3 // (3) For each RetPE lhs = rhs, retrieve the LHS
    expression ⟨[c],lhs⟩ in caller context c.
4 // (4) Add ⟨[c′],rhs⟩ ≡ ⟨[c],lhs⟩ to the solver.
5   return true;
```

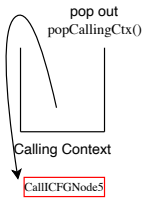
Note: `retPE.getRHSVarID()` returns `ValVar6`
`getZ3Expr(ValVar6)` binds `ValVar6` with the current `callingCtx` [ICFGNode5] and returns the Z3 expression ⟨[ICFGNode5], Var6⟩

Example 5: Interprocedural - Context Switch for Return Value



Algorithm 22: `handleRet(retEdge)`

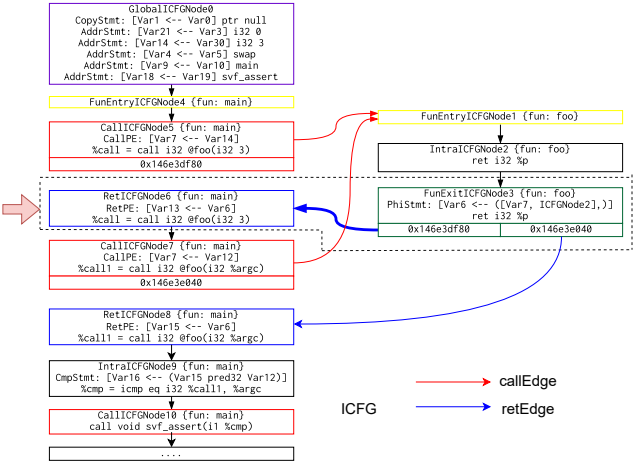
```
1 // (1) For each RetPE lhs = rhs, retrieve and
  save the RHS expression ⟨[c'],rhs⟩ in callee context
  c'.
2 // (2) Call popCallingCtx() to restore caller
  context c.
3 // (3) For each RetPE lhs = rhs, retrieve the LHS
   expression ⟨[c],lhs⟩ in caller context c.
4 // (4) Add ⟨[c'],rhs⟩ ≡ ⟨[c],lhs⟩ to the solver.
5   return true;
```



After processing the return edge

`FunExitICFGNode3` → `RetICFGNode6`

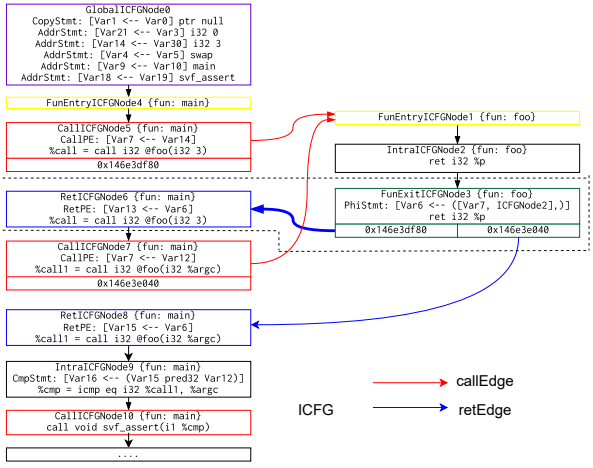
Example 5: Interprocedural - Adding Return Equation to Solver



Algorithm 23: `handleRet(retEdge)`

```
1 // (1) For each RetPE lhs = rhs, retrieve and
  save the RHS expression ⟨[c'],rhs⟩ in callee context
  c'.
2 // (2) Call popCallingCtx() to restore caller
  context c.
3 // (3) For each RetPE lhs = rhs, retrieve the LHS
   expression ⟨[c],lhs⟩ in caller context c.
4 // (4) Add ⟨[c'],rhs⟩ ≡ ⟨[c],lhs⟩ to the solver.
5   return true;
```

Example 5: Interprocedural - Handling the Second Call



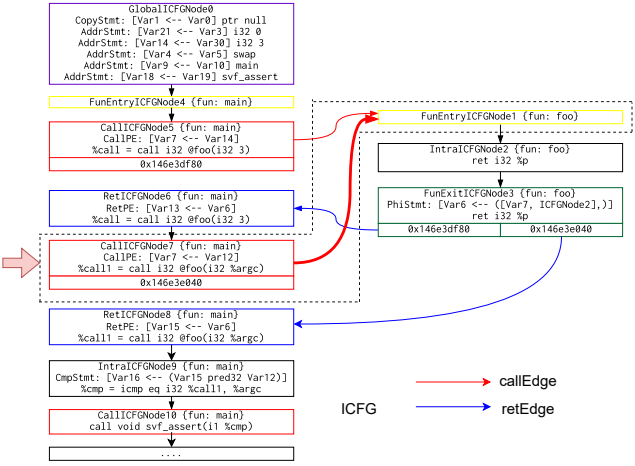
Verifying ICFG path: $0 \to 4 \to 5 \to 1 \to 2 \to 3 \to$

$6 \to 7 \to 1 \to 2 \to 3 \to 8 \to 9 \to$ *svf_assert*

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

```
-------------One Satisfying Model-------------
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
                  ...
----------------------------------------------
```

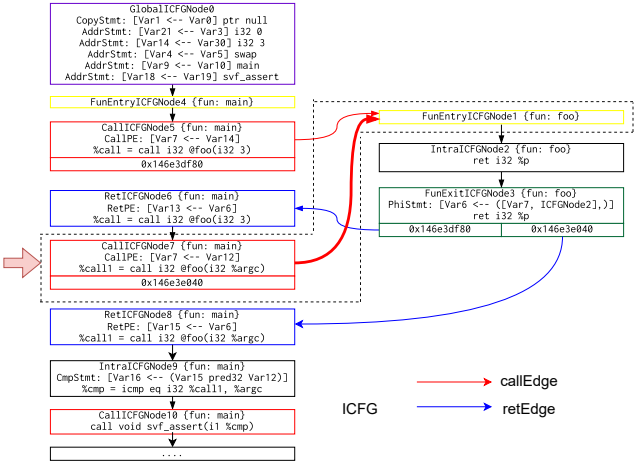
Example 5: Interprocedural - Pushing Calling Context



Algorithm 24: `handleCall(callEdge)`

```
1 // (1) For each CallPE lhs = rhs, retrieve and
  save the RHS expression ⟨[c], rhs⟩ in caller
  context c.
2 // (2) Call pushCallingCtx with the current
   callsite ICFG node; this changes the context
   from c to c'.  ;
3 // (3) For each CallPE lhs = rhs, retrieve the
  LHS expression ⟨[c'], lhs⟩ in callee context c'.  ;
4 // (4) For each CallPE lhs = rhs, add
  ⟨[c], rhs⟩ ≡ ⟨[c'], lhs⟩ to the solver.
5 return true;
```

Example 5: Interprocedural - Pushing Calling Context



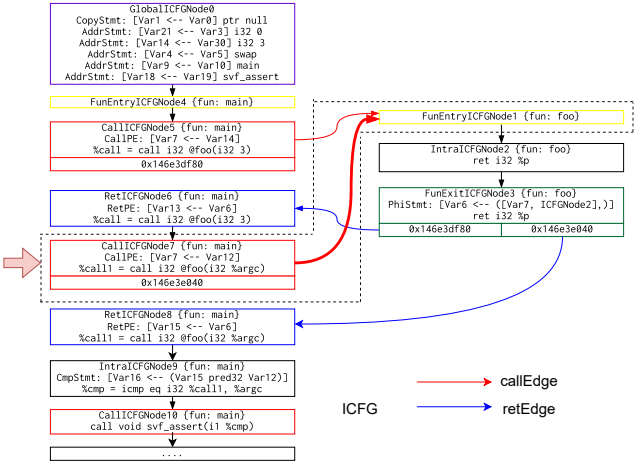
Verifying ICFG path: $0 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9 \rightarrow$ *svf_assert*

Calling Context

After processing the call edge

`CallCFGNode7` → `FunEntryICFGNode`

Example 5: Interprocedural - Retrieve Var under Callee Context



Algorithm 25: `handleCall(callEdge)`

1 // (1) For each CallPE lhs = rhs, retrieve and save the RHS expression $\langle [c], rhs \rangle$ in caller context c.

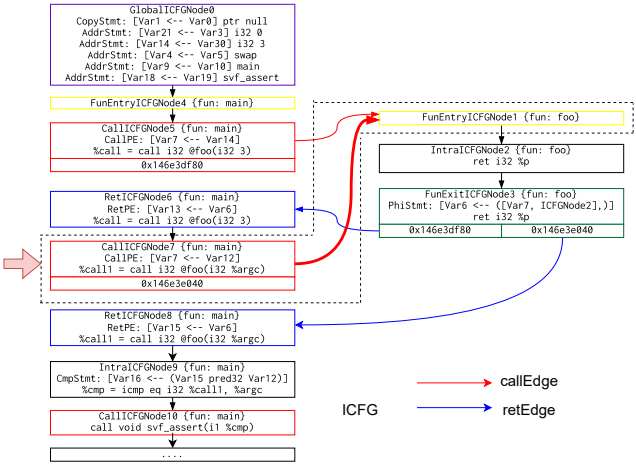
2 // (2) Call pushCallingCtx with the current callsite ICFG node; this changes the context from c to c'. ;

3 // (3) For each CallPE lhs = rhs, retrieve the LHS expression $\langle [c'], lhs \rangle$ in callee context c'. ;

4 // (4) For each CallPE lhs = rhs, add $\langle [c], rhs \rangle \equiv \langle [c'], lhs \rangle$ to 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*

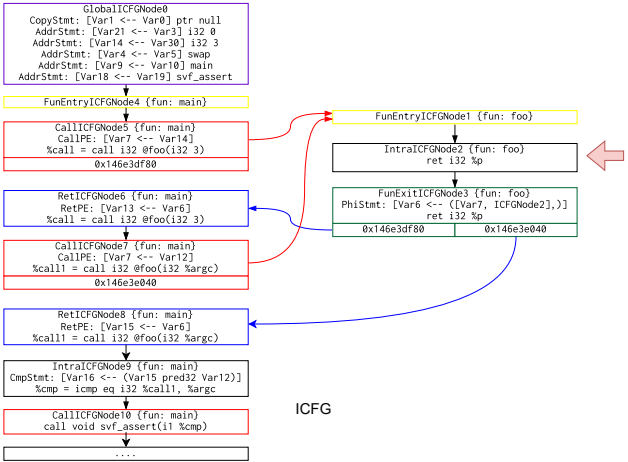
ICFG Node/Edge	Constraints in the solver
ICFGNode 0	$\texttt{Var21} \equiv 0 \wedge \texttt{Var14} \equiv 3 \wedge \ldots$
ICFGNode 5	$\wedge \ \langle[\texttt{ICFGNode5}], \texttt{Var7}\rangle \equiv \langle[\,], \texttt{Var14}\rangle$
ICFGNode 3	$\wedge \ \langle[\texttt{ICFGNode5}], \texttt{Var6}\rangle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

```
------------One Satisfying Model-------------
                    ...
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 - Handling Return



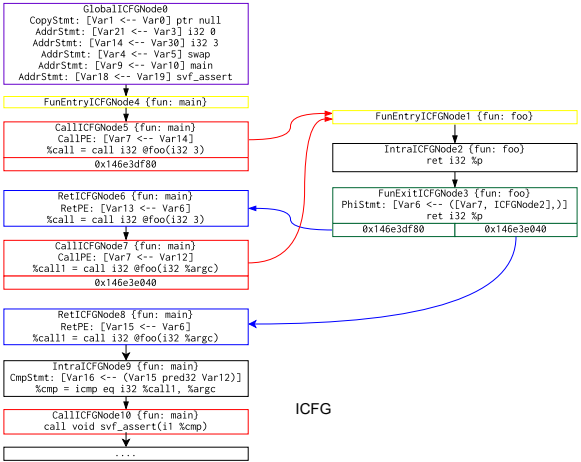
ICFG

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 - Handling Return at PHI

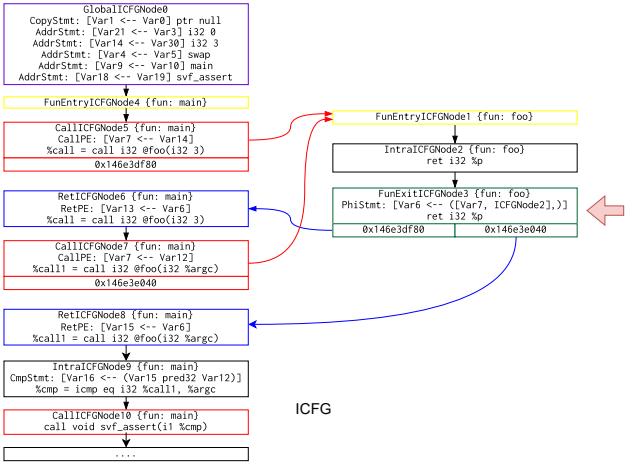


ICFG

Algorithm 26: 3 `handlePhi(edge)`

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

Example 5: Interprocedural



ICFG

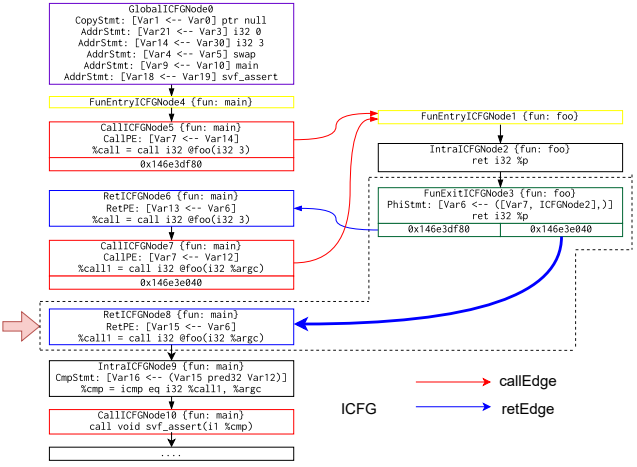
Verifying ICFG path: $0 \to 4 \to 5 \to 1 \to 2 \to 3 \to 6 \to 7 \to 1 \to 2 \to 3 \to 8 \to 9 \to$ *svf_assert*

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

```
-------------One Satisfying Model-------------
                    ...
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 - Retrieve Var under Callee Context

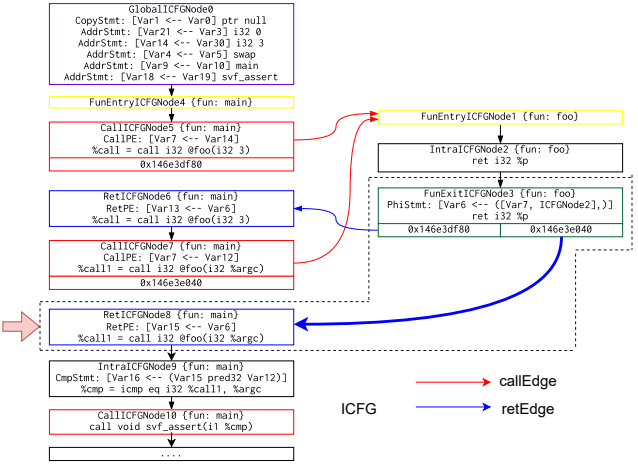


Algorithm 27: `handleRet(retEdge)`

```
1 // (1) For each RetPE lhs = rhs, retrieve and
    save the RHS expression ⟨[c′],rhs⟩ in callee
    context c'.
2 // (2) Call popCallingCtx() to restore caller
  context c.
3 // (3) For each RetPE lhs = rhs, retrieve the LHS
  expression ⟨[c],lhs⟩ in caller context c.
4 // (4) Add ⟨[c′],rhs⟩ ≡ ⟨[c],lhs⟩ to the solver.
5   return true;
```

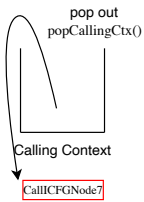
Note: `retPE.getRHSVarID()` returns `ValVar6`
`getZ3Expr(ValVar6)` binds `ValVar6` with the current `callingCtx` `[ICFGNode7]` and returns the Z3 expression for ⟨[ICFGNode7], Var6⟩

Example 5: Interprocedural - Pop Calling Context



Algorithm 28: `handleRet(retEdge)`

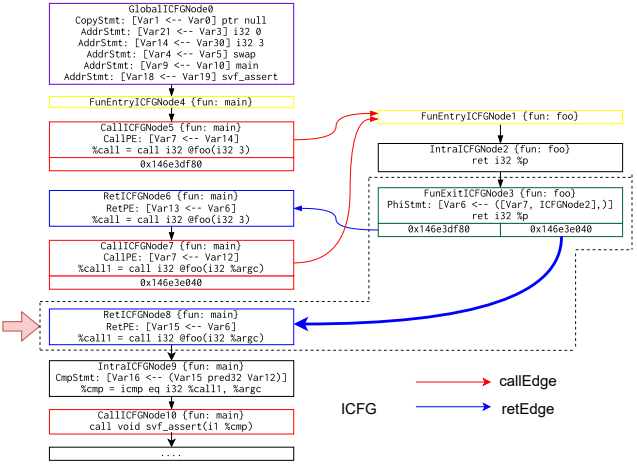
```
1 // (1) For each RetPE lhs = rhs, retrieve and
  save the RHS expression ⟨[c'],rhs⟩ in callee context
  c'.
2 // (2) Call popCallingCtx() to restore caller
   context c.
3 // (3) For each RetPE lhs = rhs, retrieve the LHS
  expression ⟨[c],lhs⟩ in caller context c.
4 // (4) Add ⟨[c'],rhs⟩ ≡ ⟨[c],lhs⟩ to the solver.
5   return true;
```



After processing the return edge

`FunExitICFGNode3` → `RetICFGNode8`

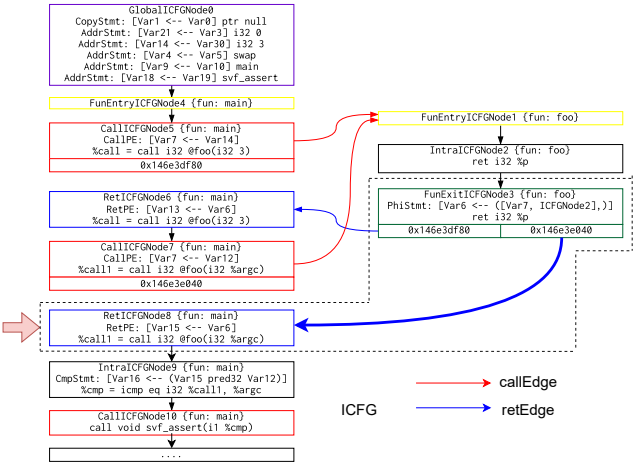
Example 5: Interprocedural - Retrieve Var under Caller Context



Algorithm 29: `handleRet(retEdge)`

```
1 // (1) For each RetPE lhs = rhs, retrieve and
  save the RHS expression ⟨[c'], rhs⟩ in callee context
  c'.
2 // (2) Call popCallingCtx() to restore caller
  context c.
3 // (3) For each RetPE lhs = rhs, retrieve the LHS
   expression ⟨[c], lhs⟩ in caller context c.
4 // (4) Add ⟨[c'], rhs⟩ ≡ ⟨[c], lhs⟩ to the solver.
5   return true;
```

Example 5: Interprocedural

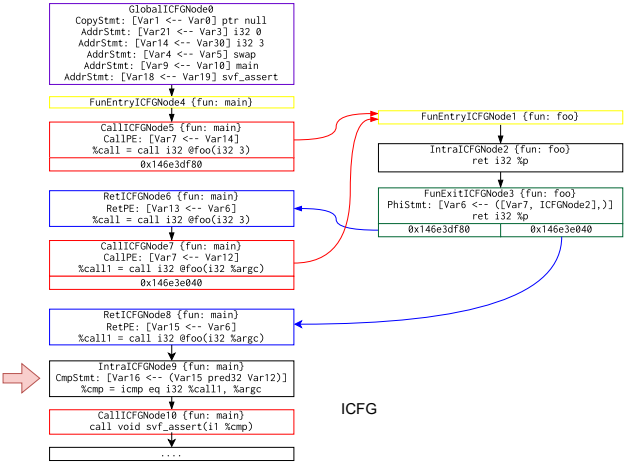


Verifying ICFG path: 0 → 4 → 5 → 1 → 2 → 3 → 6 → 7 → 1 → 2 → 3 →8 → 9 → *svf_assert*

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

```
------------One Satisfying Model------------
                    ...
ValVar21                Value: 0
ValVar14                Value: 3
                    ...
ValVar13                Value: 3
ValVar12                Value: 0
ValVar15                Value: 0
                    ...
--------------------------------------------
```

Example 5: Interprocedural



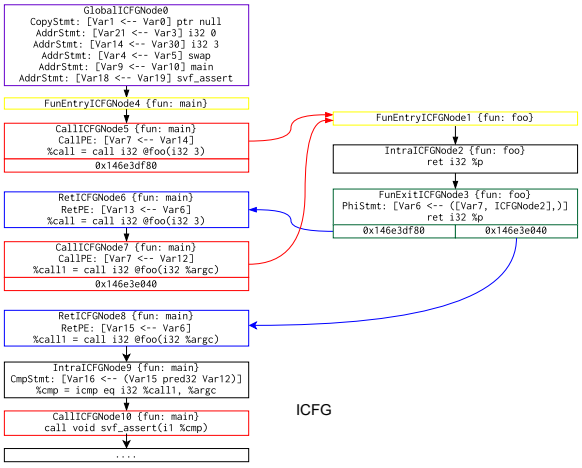
ICFG

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	$\texttt{Var21} \equiv 0 \wedge \texttt{Var14} \equiv 3 \wedge \ldots$
ICFGNode 5	$\wedge \quad \langle [\texttt{ICFGNode5}], \texttt{Var7} \rangle \equiv \langle [\,], \texttt{Var14} \rangle$
ICFGNode 3	$\wedge \quad \langle [\texttt{ICFGNode5}], \texttt{Var6} \rangle \equiv \langle [\texttt{ICFGNode5}], \texttt{Var7} \rangle$
ICFGNode 6	$\wedge \quad \langle [\,], \texttt{Var13} \rangle \equiv \langle [\texttt{ICFGNode5}], \texttt{Var6} \rangle$
ICFGNode 7	$\wedge \quad \langle [\texttt{ICFGNode7}], \texttt{Var7} \rangle \equiv \langle [\,], \texttt{Var12} \rangle$
ICFGNode 3	$\wedge \quad \langle [\texttt{ICFGNode7}], \texttt{Var6} \rangle \equiv \langle [\texttt{ICFGNode7}], \texttt{Var7} \rangle$
ICFGNode 8	$\wedge \quad \langle [\,], \texttt{Var15} \rangle \equiv \langle [\texttt{ICFGNode7}], \texttt{Var6} \rangle$
ICFGNode 9	$\wedge \quad \texttt{Var16} \equiv \texttt{ite}(\texttt{Var15} \equiv \texttt{Var12}, 1, 0)$

```
-------------One Satisfying Model-------------
                      ...
ValVar13                    Value: 3
ValVar12                    Value: 0
ValVar15                    Value: 0
ValVar16                    Value: 1
                      ...
----------------------------------------------
```

Example 5: Interprocedural - Assertion Violated



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

Solver yields **UNSAT**, meaning no counter example.

The assertion is verified.

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