

Lab: SVFIR and Control-Flow Reachability

(Week 2)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Quiz-1 + Lab-Exercise-1 + Assignment-1

- A set of quizzes on WebCMS (5 points) due on **Week 3 Tuesday 23:59**
 - LLVM compiler and its intermediate representation
 - Code graphs (including ICFG and PAG)
- Lab-Exercise-1 (5 points) due on **Week 3 Tuesday 23:59**
 - Implement a graph traversal on a general graph
- Assignment-1 (20 points) due on **Week 4 Tuesday 23:59**
 - **Control-flow**: Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and collect **feasible** paths from a source to a sink node on the ICFG.
 - **Data-flow**: Implement field-sensitive Andersen's inclusion-based constraint solving for points-to analysis
 - Implement a taint checker **using control-flow and data-flow analysis**.

Quiz-1 + Lab-Exercise-1 + Assignment-1

- A set of quizzes on WebCMS (5 points) due on **Week 3 Tuesday 23:59**
 - LLVM compiler and its intermediate representation
 - Code graphs (including ICFG and PAG)
- Lab-Exercise-1 (5 points) due on **Week 3 Tuesday 23:59**
 - Implement a graph traversal on a general graph
- Assignment-1 (20 points) due on **Week 4 Tuesday 23:59**
 - **Control-flow:** Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and collect **feasible** paths from a source to a sink node on the ICFG.
 - **Data-flow:** Implement field-sensitive Andersen's inclusion-based constraint solving for points-to analysis
 - Implement a taint checker **using control-flow and data-flow analysis**.
 - **Specification and code template:** <https://github.com/SVF-tools/Software-Security-Analysis/wiki/Assignment-1>
 - **SVF APIs for control- and data-flow analysis** <https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-API>

Understanding LLVM-IR and SVF-IR

- (1) Compile C programs under SVFIR/src into their LLVM IR and print their SVF IR (PAG, ICFG, Constraint Graph).
 - <https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVFIR>
 - Understand the mapping from a C program to its corresponding LLVM IR
 - Understand the mapping from LLVM IR to its corresponding SVF IR
- (2) Generate and visualize the graph representation of SVF IR (e.g., `example.ll.pag.dot`, `example.ll.icfg.dot`, `consG.ll.dot`).
 - <https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVFIR#4-visualize-icfg-constraint-graph-and-svfirpag-graph>
- (3) Write code to iterate SVFVars and print nodes/edges of PAG and ICFG.
 - <https://github.com/SVF-tools/Software-Security-Analysis/blob/main/SVFIR/SVFIR.cpp#L74-L98> (C++)
 - <https://github.com/SVF-tools/SVF-Python/tree/main/demo> (Python)
- (4) More about LLVM IR and SVF's graph representation
 - LLVM language manual <https://llvm.org/docs/LangRef.html>
 - SVF website <https://github.com/SVF-tools/SVF>

Context-Sensitive Control-Flow Reachability (Algorithm)

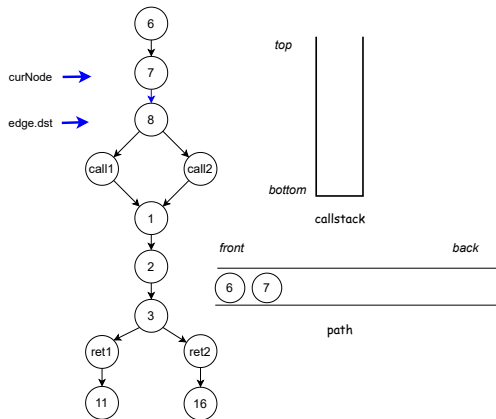
Algorithm 1: 1 Context sensitive control-flow reachability

Input : curNode : ICFGNode snk : ICFGNode path : vector(ICFGNode)
 callstack : vector(SVFInstruction) visited : set(ICFGNode, callstack);

```
1 dfs(curNode, snk)
2   pair = <curNode, callstack>;
3   if pair ∈ visited then
4   |   return;
5   visited.insert(pair);
6   path.push_back(curNode);
7   if src == snk then
8   |   collectICFGPath(path);
9   foreach edge ∈ curNode.getOutEdges() do
10  |   if edge.isIntraCFGEde() then
11  |   |   dfs(edge.dst, snk);
12  |   else if edge.isCallCFGEde() then
13  |   |   callstack.push_back(edge.getCallSite());
14  |   |   dfs(edge.dst, snk);
15  |   |   callstack.pop_back();
16  |   else if edge.isRetCFGEde() then
17  |   |   if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18  |   |   |   callstack.pop_back();
19  |   |   |   dfs(edge.dst, snk);
20  |   |   |   callstack.push_back(edge.getCallSite());
21  |   |   else if callstack == ∅ then
22  |   |   |   dfs(edge.dst, snk);
23   visited.erase(pair);
24   path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

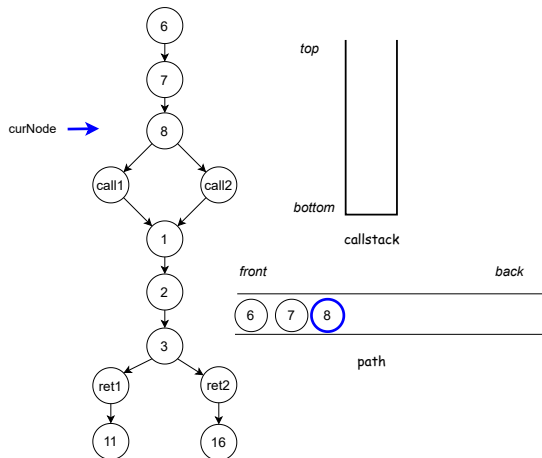


Algorithm 2: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = <curNode, callstack>;  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23  visited.erase(pair);  
24  path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

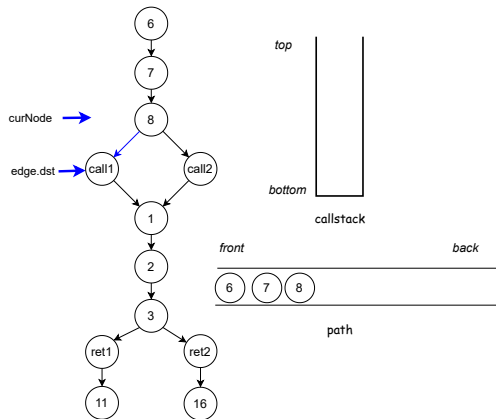


Algorithm 3: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;
1  dfs(curNode, snk)
2  pair = <curNode, callstack>;
3  if pair ∈ visited then
4    return;
5  visited.insert(pair);
6  path.push_back(curNode);
7  if src == snk then
8    collectICFGPath(path);
9  foreach edge ∈ curNode.getOutEdges() do
10   if edge.isIntraCFGEde() then
11     dfs(edge.dst, snk);
12   else if edge.isCallCFGEde() then
13     callstack.push_back(edge.getCallSite());
14     dfs(edge.dst, snk);
15     callstack.pop_back();
16   else if edge.isRetCFGEde() then
17     if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18       callstack.pop_back();
19       dfs(edge.dst, snk);
20       callstack.push_back(edge.getCallSite());
21     else if callstack == ∅ then
22       dfs(edge.dst, snk);
23  visited.erase(pair);
24  path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

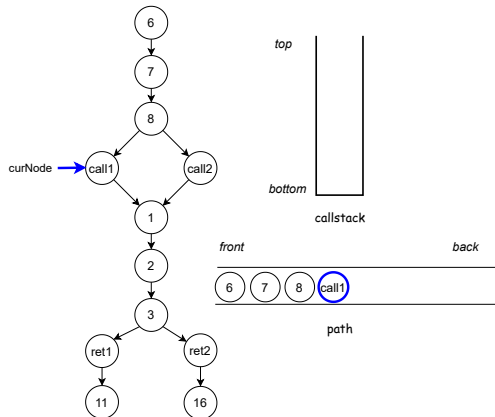


Algorithm 4: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = <curNode, callstack>;  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23  visited.erase(pair);  
24  path.pop_back();
```


Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

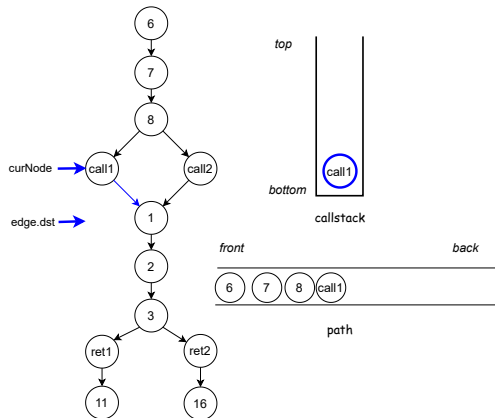


Algorithm 5: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;
1  dfs(curNode, snk)
2  pair = <curNode, callstack>;
3  if pair ∈ visited then
4  | return;
5  visited.insert(pair);
6  path.push_back(curNode);
7  if src == snk then
8  | collectICFGPath(path);
9  foreach edge ∈ curNode.getOutEdges() do
10 | if edge.isIntraCFGEde() then
11 | | dfs(edge.dst, snk);
12 | else if edge.isCallCFGEde() then
13 | | callstack.push_back(edge.getCallSite());
14 | | dfs(edge.dst, snk);
15 | | callstack.pop_back();
16 | else if edge.isRetCFGEde() then
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18 | | | callstack.pop_back();
19 | | | dfs(edge.dst, snk);
20 | | | callstack.push_back(edge.getCallSite());
21 | | else if callstack == ∅ then
22 | | | dfs(edge.dst, snk);
23 visited.erase(pair);
24 path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

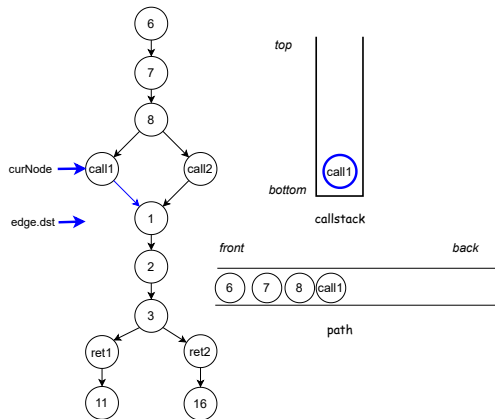


Algorithm 6: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23  visited.erase(pair);  
24  path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

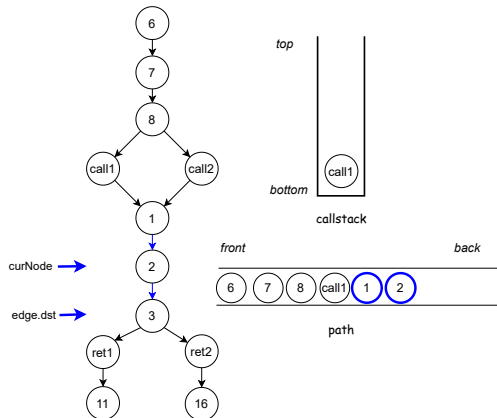


Algorithm 7: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23  visited.erase(pair);  
24  path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

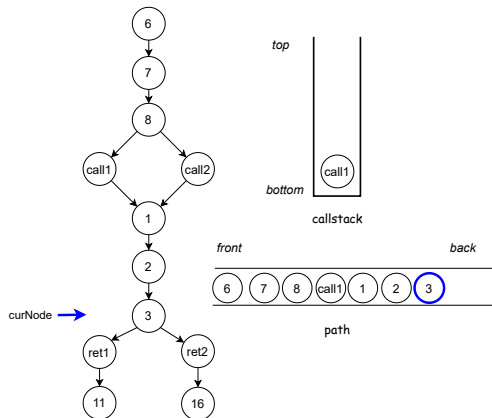


Algorithm 8: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = <curNode, callstack>;  
3  if pair ∈ visited then  
4  |   return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  |   collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | |   dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | |   callstack.push_back(edge.getCallSite());  
14 | |   dfs(edge.dst, snk);  
15 | |   callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | |   if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | |   callstack.pop_back();  
19 | | |   dfs(edge.dst, snk);  
20 | | |   callstack.push_back(edge.getCallSite());  
21 | |   else if callstack == ∅ then  
22 | | |   dfs(edge.dst, snk);  
23 visited.erase(pair);  
24 path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

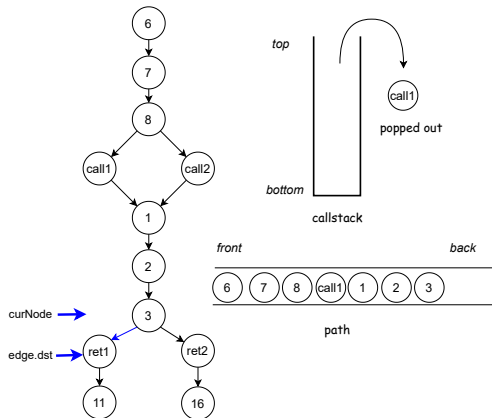


Algorithm 9: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;
1  dfs(curNode, snk)
2  pair = (curNode, callstack);
3  if pair ∈ visited then
4  | return;
5  visited.insert(pair);
6  path.push_back(curNode);
7  if src == snk then
8  | collectICFGPath(path);
9  foreach edge ∈ curNode.getOutEdges() do
10 | if edge.isIntraCFGEde() then
11 | | dfs(edge.dst, snk);
12 | else if edge.isCallCFGEde() then
13 | | callstack.push_back(edge.getCallSite());
14 | | dfs(edge.dst, snk);
15 | | callstack.pop_back();
16 | else if edge.isRetCFGEde() then
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18 | | | callstack.pop_back();
19 | | | dfs(edge.dst, snk);
20 | | | callstack.push_back(edge.getCallSite());
21 | | else if callstack == ∅ then
22 | | | dfs(edge.dst, snk);
23 visited.erase(pair);
24 path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

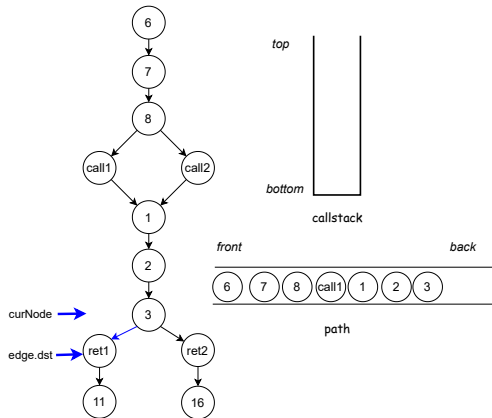


Algorithm 10: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23  visited.erase(pair);  
24  path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

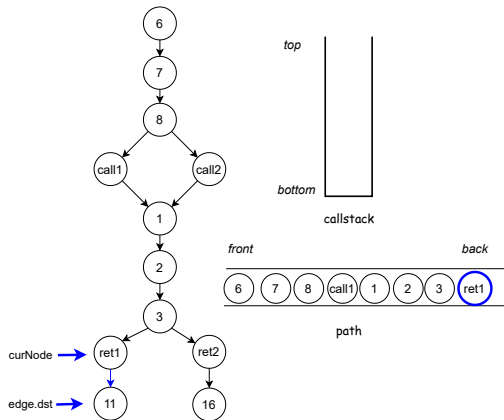


Algorithm 11: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23 visited.erase(pair);  
24 path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG



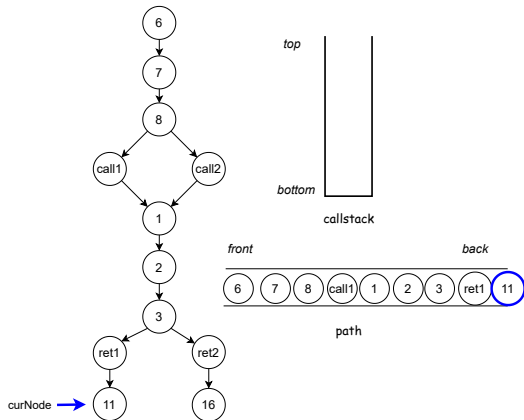
Algorithm 12: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;

1  dfs(curNode, snk)
2  pair = <curNode, callstack>;
3  if pair ∈ visited then
4  | return;
5  visited.insert(pair);
6  path.push_back(curNode);
7  if src == snk then
8  | collectICFGPath(path);
9  foreach edge ∈ curNode.getOutEdges() do
10 | if edge.isIntraCFGEde() then
11 | | dfs(edge.dst, snk);
12 | else if edge.isCallCFGEde() then
13 | | callstack.push_back(edge.getCallSite());
14 | | dfs(edge.dst, snk);
15 | | callstack.pop_back();
16 | else if edge.isRetCFGEde() then
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18 | | | callstack.pop_back();
19 | | | dfs(edge.dst, snk);
20 | | | callstack.push_back(edge.getCallSite());
21 | | else if callstack == ∅ then
22 | | | dfs(edge.dst, snk);
23 visited.erase(pair);
24 path.pop_back();
```


Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

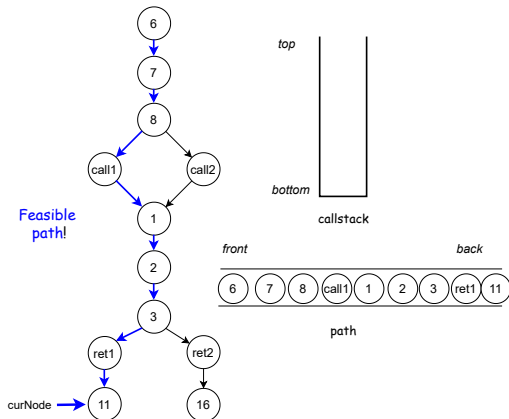


Algorithm 13: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;
1  dfs(curNode, snk)
2  pair = <curNode, callstack>;
3  if pair ∈ visited then
4  | return;
5  visited.insert(pair);
6  path.push_back(curNode);
7  if src == snk then
8  | collectICFGPath(path);
9  foreach edge ∈ curNode.getOutEdges() do
10 | if edge.isIntraCFGEde() then
11 | | dfs(edge.dst, snk);
12 | else if edge.isCallCFGEde() then
13 | | callstack.push_back(edge.getCallSite());
14 | | dfs(edge.dst, snk);
15 | | callstack.pop_back();
16 | else if edge.isRetCFGEde() then
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18 | | | callstack.pop_back();
19 | | | dfs(edge.dst, snk);
20 | | | callstack.push_back(edge.getCallSite());
21 | | else if callstack == ∅ then
22 | | | dfs(edge.dst, snk);
23 visited.erase(pair);
24 path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG



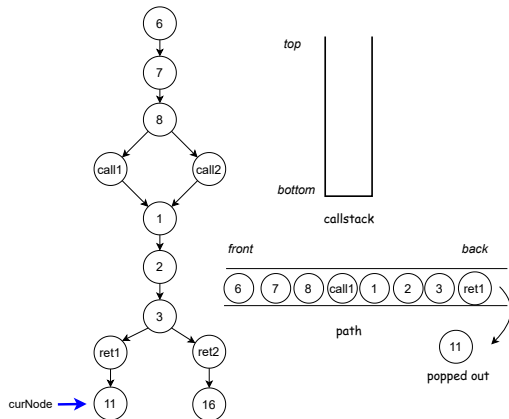
Algorithm 14: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;

1 dfs(curNode, snk)
2   pair = (curNode, callstack);
3   if pair ∈ visited then
4     return;
5   visited.insert(pair);
6   path.push_back(curNode);
7   if src == snk then
8     collectICFGPath(path);
9   foreach edge ∈ curNode.getOutEdges() do
10    if edge.isIntraCFGEde() then
11      dfs(edge.dst, snk);
12    else if edge.isCallCFGEde() then
13      callstack.push_back(edge.getCallSite());
14      dfs(edge.dst, snk);
15      callstack.pop_back();
16    else if edge.isRetCFGEde() then
17      if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18        callstack.pop_back();
19        dfs(edge.dst, snk);
20        callstack.push_back(edge.getCallSite());
21      else if callstack == ∅ then
22        dfs(edge.dst, snk);
23   visited.erase(pair);
24   path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

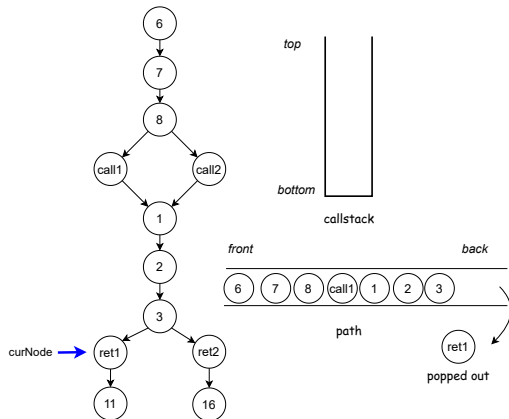


Algorithm 15: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23 | visited.erase(pair);  
24 | path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

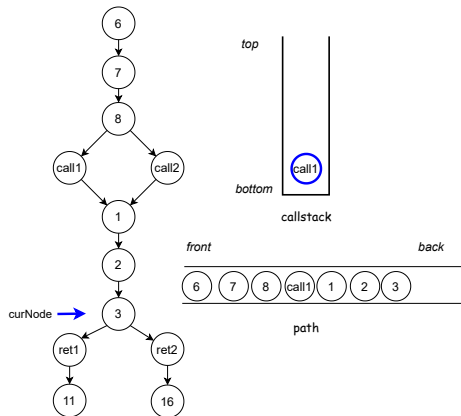


Algorithm 16: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23 | visited.erase(pair);  
24 | path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG



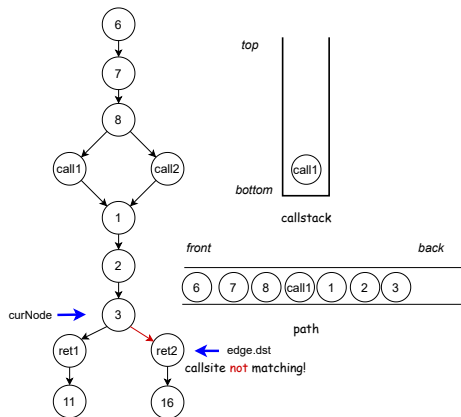
Algorithm 17: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;

1  dfs(curNode, snk)
2  pair = <curNode, callstack>;
3  if pair ∈ visited then
4    return;
5  visited.insert(pair);
6  path.push_back(curNode);
7  if src == snk then
8    collectICFGPath(path);
9  foreach edge ∈ curNode.getOutEdges() do
10   if edge.isIntraCFGEde() then
11     dfs(edge.dst, snk);
12   else if edge.isCallCFGEde() then
13     callstack.push_back(edge.getCallSite());
14     dfs(edge.dst, snk);
15     callstack.pop_back();
16   else if edge.isRetCFGEde() then
17     if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18       callstack.pop_back();
19       dfs(edge.dst, snk);
20       callstack.push_back(edge.getCallSite());
21     else if callstack == ∅ then
22       dfs(edge.dst, snk);
23  visited.erase(pair);
24  path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

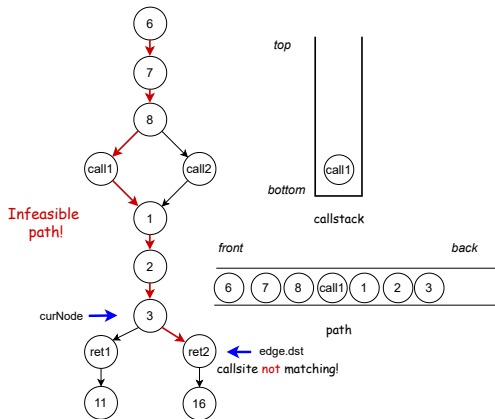


Algorithm 18: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = <curNode, callstack>;  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23  visited.erase(pair);  
24  path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG



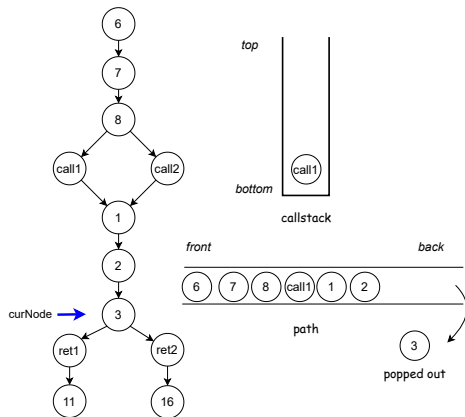
Algorithm 19: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;

1 dfs(curNode, snk)
2   pair = <curNode, callstack>;
3   if pair ∈ visited then
4     return;
5   visited.insert(pair);
6   path.push_back(curNode);
7   if src == snk then
8     collectICFGPath(path);
9   foreach edge ∈ curNode.getOutEdges() do
10    if edge.isIntraCFGEde() then
11      dfs(edge.dst, snk);
12    else if edge.isCallCFGEde() then
13      callstack.push_back(edge.getCallSite());
14      dfs(edge.dst, snk);
15      callstack.pop_back();
16    else if edge.isRetCFGEde() then
17      if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18        callstack.pop_back();
19        dfs(edge.dst, snk);
20        callstack.push_back(edge.getCallSite());
21      else if callstack == ∅ then
22        dfs(edge.dst, snk);
23   visited.erase(pair);
24   path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

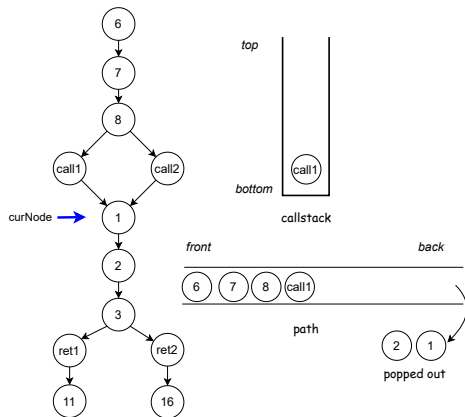


Algorithm 20: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23 | visited.erase(pair);  
24 | path.pop_back();
```


Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

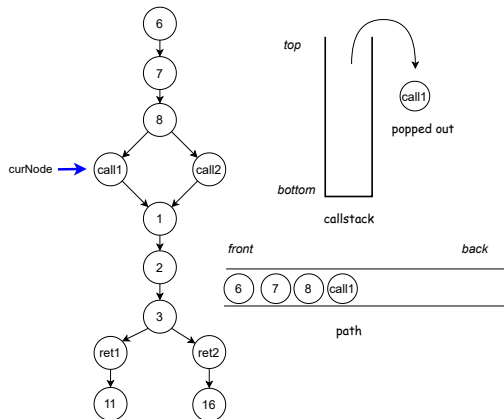


Algorithm 21: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23 | visited.erase(pair);  
24 | path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

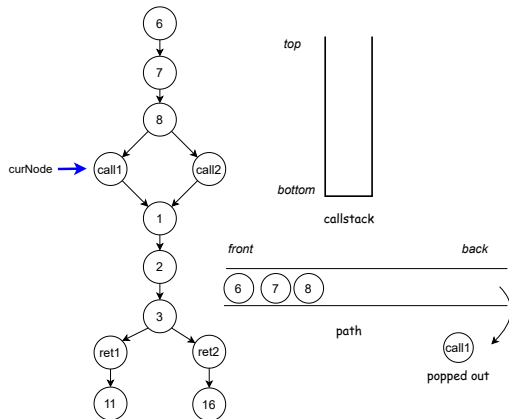


Algorithm 22: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = <curNode, callstack>;  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23  visited.erase(pair);  
24  path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG

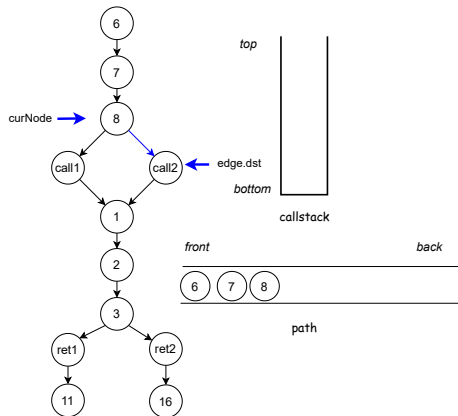


Algorithm 23: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23 | visited.erase(pair);  
24 | path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG



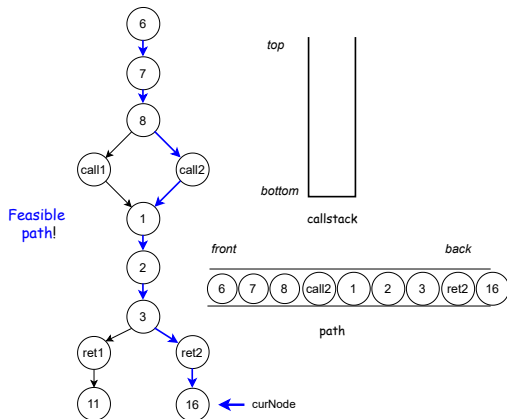
Algorithm 24: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;

1 dfs(curNode, snk)
2   pair = <curNode, callstack>;
3   if pair ∈ visited then
4     return;
5   visited.insert(pair);
6   path.push_back(curNode);
7   if src == snk then
8     collectICFGPath(path);
9   foreach edge ∈ curNode.getOutEdges() do
10    if edge.isIntraCFGEde() then
11      dfs(edge.dst, snk);
12    else if edge.isCallCFGEde() then
13      callstack.push_back(edge.getCallSite());
14      dfs(edge.dst, snk);
15      callstack.pop_back();
16    else if edge.isRetCFGEde() then
17      if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18        callstack.pop_back();
19        dfs(edge.dst, snk);
20        callstack.push_back(edge.getCallSite());
21      else if callstack == ∅ then
22        dfs(edge.dst, snk);
23   visited.erase(pair);
24   path.pop_back();
```

Context-Sensitive Control-Flow Reachability

A feasible path from node 6 to node 11 on ICFG



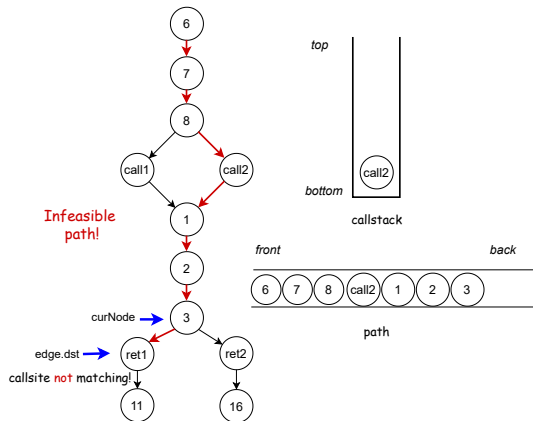
Algorithm 25: 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;

1 dfs(curNode, snk)
2   pair = <curNode, callstack>;
3   if pair ∈ visited then
4     return;
5   visited.insert(pair);
6   path.push_back(curNode);
7   if src == snk then
8     collectICFGPath(path);
9   foreach edge ∈ curNode.getOutEdges() do
10    if edge.isIntraCFGEde() then
11      dfs(edge.dst, snk);
12    else if edge.isCallCFGEde() then
13      callstack.push_back(edge.getCallSite());
14      dfs(edge.dst, snk);
15      callstack.pop_back();
16    else if edge.isRetCFGEde() then
17      if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
18        callstack.pop_back();
19        dfs(edge.dst, snk);
20        callstack.push_back(edge.getCallSite());
21      else if callstack == ∅ then
22        dfs(edge.dst, snk);
23   visited.erase(pair);
24   path.pop_back();
```

Context-Sensitive Control-Flow Reachability

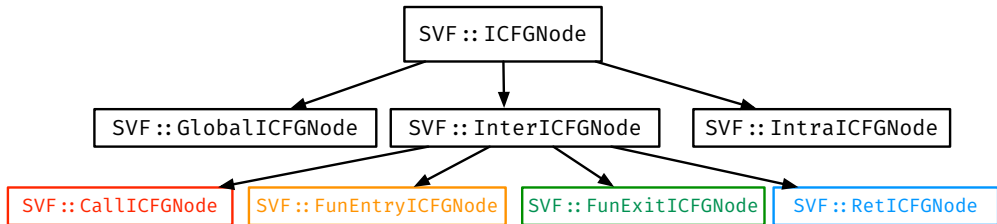
A feasible path from node 6 to node 11 on ICFG



Algorithm 26: 1 Context sensitive control-flow reachability

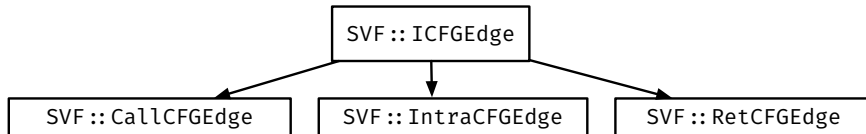
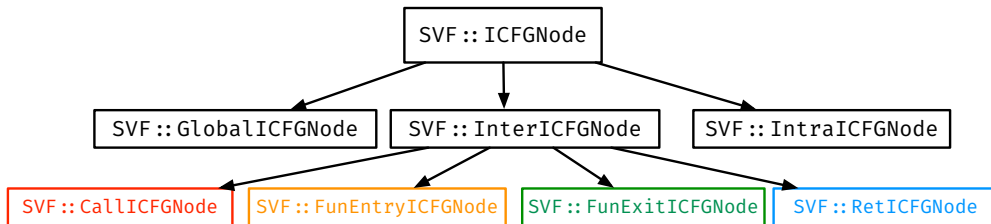
```
Input : curNode : ICFGNode  snk : ICFGNode  path : vector<ICFGNode>  
        callstack : vector<SVFInstruction>  visited : set<ICFGNode, callstack>;  
1  dfs(curNode, snk)  
2  pair = (curNode, callstack);  
3  if pair ∈ visited then  
4  | return;  
5  visited.insert(pair);  
6  path.push_back(curNode);  
7  if src == snk then  
8  | collectICFGPath(path);  
9  foreach edge ∈ curNode.getOutEdges() do  
10 | if edge.isIntraCFGEde() then  
11 | | dfs(edge.dst, snk);  
12 | else if edge.isCallCFGEde() then  
13 | | callstack.push_back(edge.getCallSite());  
14 | | dfs(edge.dst, snk);  
15 | | callstack.pop_back();  
16 | else if edge.isRetCFGEde() then  
17 | | if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then  
18 | | | callstack.pop_back();  
19 | | | dfs(edge.dst, snk);  
20 | | | callstack.push_back(edge.getCallSite());  
21 | | else if callstack == ∅ then  
22 | | | dfs(edge.dst, snk);  
23  visited.erase(pair);  
24  path.pop_back();
```

ICFG Node and Edge Classes



<https://github.com/SVF-tools/SVF/blob/master/include/Graphs/ICFGNode.h>

ICFG Node and Edge Classes



<https://github.com/SVF-tools/SVF/blob/master/include/Graphs/ICFGEde.h>

SVFUtil::cast and SVFUtil::dyn_cast

- C++ Inheritance: see slides in Week 1 Lab.
- Casting a **parent** class pointer to pointer of a **Child** type:
 - `SVFUtil::cast`
 - Casts a pointer or reference to an instance of a specified class. This cast fails and aborts the program if the object or reference is not the specified class at runtime.
 - `SVFUtil::dyn_cast`
 - "Checked cast" operation. Checks to see if the operand is of the specified type, and if so, returns a pointer to it (this operator does not work with references). If the operand is not of the correct type, a null pointer is returned.
 - Works very much like the `dynamic_cast<>` operator in C++, and should be used in the same circumstances.
- Example: accessing the attributes of the child class via casting.
 - `RetBlockNode* retNode = SVFUtil::cast<RetBlockNode>(ICFGNode);`
 - `CallCFGEde* callEdge = SVFUtil::dyn_cast<CallCFGEde>(ICFGEde);`