Assignment 2

Yulei Sui

University of Technology Sydney, Australia

Assignment 2: Quizzes + A Coding Task

- Two sets of quizzes (10 points)
  - LLVM compiler and its intermediate representation
  - Code graphs (including ICFG and PAG)
- One coding task (10 points)
  - **Goal**: implement a context-sensitive graph traversal on ICFG and print **feasible** paths from a source node to a sink node on the graph

Assignment 2: Quizzes + A Coding Task

- Two sets of quizzes (10 points)
  - LLVM compiler and its intermediate representation
  - Code graphs (including ICFG and PAG)
- One coding task (10 points)
  - **Goal**: implement a context-sensitive graph traversal on ICFG and print **feasible** paths from a source node to a sink node on the graph
  - **Specification and code template**: `https://github.com/SVF-tools/Teaching-Software-Verification/wiki/Assignment-2`
  - **SVF CPP API** `https://github.com/SVF-tools/Teaching-Software-Verification/wiki/SVF-APIs`

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

Context-Sensitive Control-Dependence (Algorithm)

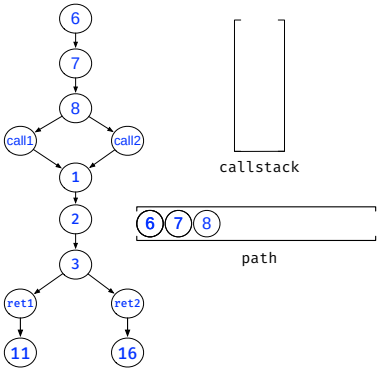
Algorithm 1 Context sensitive control-flow reachability

Input : curNode : ICFGNode snk : ICFGNode path : vector⟨ICFGNode⟩
callstack : vector⟨SVFInstruction⟩ visited : set⟨ICFGNode, callstack⟩;

```
1  dfs(curNode,dst)
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  |   printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10   |  if edge.isIntraCFGEdge() then
11   |  |  dfs(edge.dst, snk)
12   |  else if edge.isCallCFGEdge() then
13   |  |  callstack.push_back(edge.getCallSite())
14   |  |  dfs(edge.dst, snk)
15   |  |  callstack.pop_back()
16   |  else if edge.isRetCFGEdge() 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-Dependence

Obtaining a path from node 6 to node 11 on ICFG



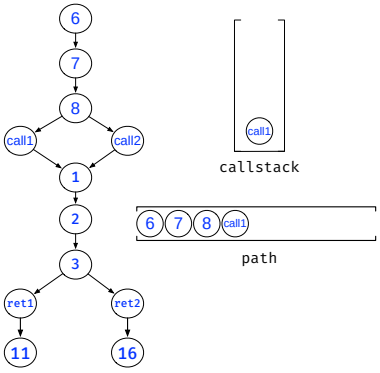
Algorithm 1 Context sensitive control-flow reachability

input : curNode : ICFGNode snk : ICFGNode path : vector⟨ICFGNode⟩
callstack : vector⟨SVFInstruction⟩ visited : set⟨ICFGNode, callstack⟩;

```
1  dfs(curNode, dst)
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      printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10     if edge.isIntraCFGEdge() then
11       dfs(edge.dst, snk)
12     else if edge.isCallCFGEdge() then
13       callstack.push_back(edge.getCallSite())
14       dfs(edge.dst, snk)
15       callstack.pop_back()
16     else if edge.isRetCFGEdge() 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-Dependence

Obtaining a path from node 6 to node 11 on ICFG

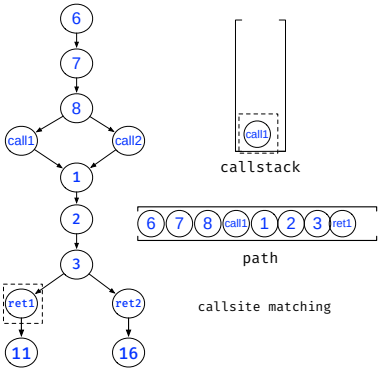


Algorithm 1 Context sensitive control-flow reachability

```
input : curNode : ICFGNode   snk : ICFGNode   path : vector⟨ICFGNode⟩
        callstack : vector⟨SVFInstruction⟩   visited : set⟨ICFGNode, callstack⟩;
1  dfs(curNode, dst)
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        printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10       if edge.isIntraCFGEdge() then
11           dfs(edge.dst, snk)
12       else if edge.isCallCFGEdge() then
13           callstack.push_back(edge.getCallSite())
14           dfs(edge.dst, snk)
15           callstack.pop_back()
16       else if edge.isRetCFGEdge() 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-Dependence

Obtaining a path from node 6 to node 11 on ICFG



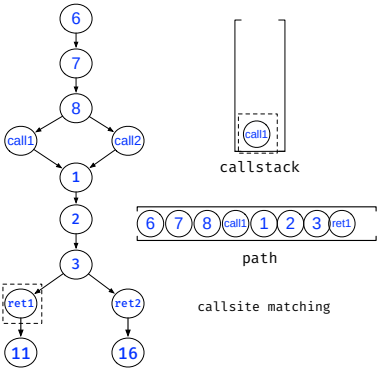
Algorithm 1 Context sensitive control-flow reachability

input : curNode : ICFGNode snk : ICFGNode path : vector⟨ICFGNode⟩
callstack : vector⟨SVFInstruction⟩ visited : set⟨ICFGNode, callstack⟩;

```
1  dfs(curNode, dst)
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      printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10     if edge.isIntraCFGEdge() then
11       dfs(edge.dst, snk)
12     else if edge.isCallCFGEdge() then
13       callstack.push_back(edge.getCallSite())
14       dfs(edge.dst, snk)
15       callstack.pop_back()
16     else if edge.isRetCFGEdge() 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-Dependence

Obtaining a path from node 6 to node 11 on ICFG



callstack

path

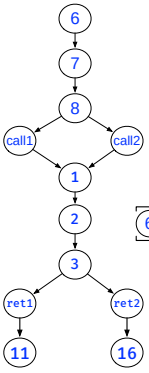
callsite matching

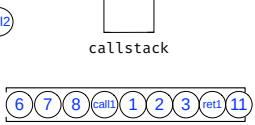
Algorithm 1 Context sensitive control-flow reachability

```
input : curNode : ICFGNode   snk : ICFGNode   path : vector⟨ICFGNode⟩
        callstack : vector⟨SVFInstruction⟩   visited : set⟨ICFGNode, callstack⟩;
1  dfs(curNode, dst)
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        printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10       if edge.isIntraCFGEdge() then
11           dfs(edge.dst, snk)
12       else if edge.isCallCFGEdge() then
13           callstack.push_back(edge.getCallSite())
14           dfs(edge.dst, snk)
15           callstack.pop_back()
16       else if edge.isRetCFGEdge() 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-Dependence

Obtaining a path from node 6 to node 11 on ICFG





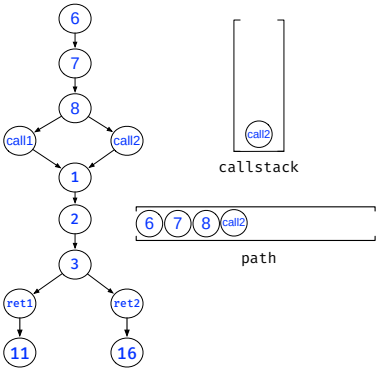
ICFG path:
6→7→8→call1→1→2→3→ret1→11

Algorithm 1 Context sensitive control-flow reachability

```
Input : curNode : ICFGNode   snk : ICFGNode   path : vector⟨ICFGNode⟩
        callstack : vector⟨SVFInstruction⟩   visited : set⟨ICFGNode, callstack⟩;
1  dfs(curNode, dst)
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        printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10       if edge.isIntraCFGEdge() then
11           dfs(edge.dst, snk)
12       else if edge.isCallCFGEdge() then
13           callstack.push_back(edge.getCallSite())
14           dfs(edge.dst, snk)
15           callstack.pop_back()
16       else if edge.isRetCFGEdge() 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-Dependence

Obtaining a path from node 6 to node 11 on ICFG



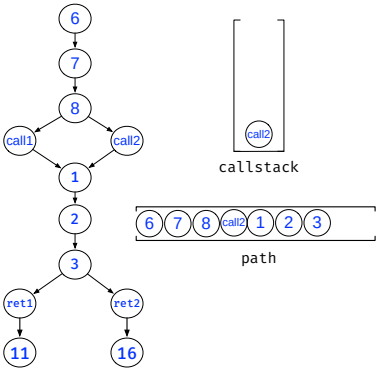
Algorithm 1 Context sensitive control-flow reachability

input : curNode : ICFGNode snk : ICFGNode path : vector⟨ICFGNode⟩
callstack : vector⟨SVFInstruction⟩ visited : set⟨ICFGNode, callstack⟩;

```
1  dfs(curNode, dst)
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      printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10     if edge.isIntraCFGEdge() then
11       dfs(edge.dst, snk)
12     else if edge.isCallCFGEdge() then
13       callstack.push_back(edge.getCallSite())
14       dfs(edge.dst, snk)
15       callstack.pop_back()
16     else if edge.isRetCFGEdge() 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-Dependence

Obtaining a path from node 6 to node 11 on ICFG

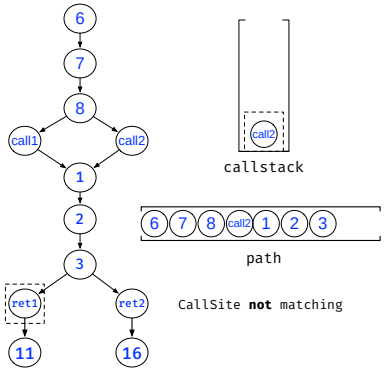


Algorithm 1 Context sensitive control-flow reachability

```
input : curNode : ICFGNode   snk : ICFGNode   path : vector⟨ICFGNode⟩
        callstack : vector⟨SVFInstruction⟩   visited : set⟨ICFGNode, callstack⟩;
1  dfs(curNode, dst)
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        printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10       if edge.isIntraCFGEdge() then
11           dfs(edge.dst, snk)
12       else if edge.isCallCFGEdge() then
13           callstack.push_back(edge.getCallSite())
14           dfs(edge.dst, snk)
15           callstack.pop_back()
16       else if edge.isRetCFGEdge() 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-Dependence

Obtaining a path from node 6 to node 11 on ICFG

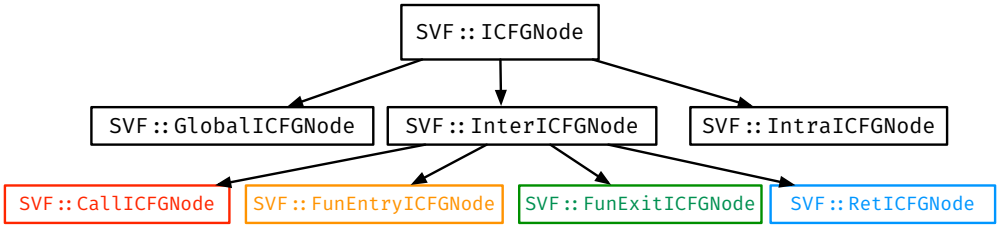


Algorithm 1 Context sensitive control-flow reachability

input : curNode : ICFGNode snk : ICFGNode path : vector⟨ICFGNode⟩
callstack : vector⟨SVFInstruction⟩ visited : set⟨ICFGNode, callstack⟩;

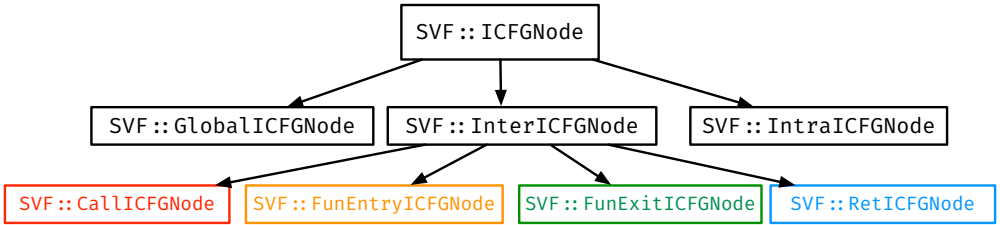
```
1  dfs(curNode, dst)
2    pair = ⟨curNode, callstack⟩
3    if pair ∈ visited then
4      return
5    visited.insert(pair)
6    path.push_back(curNode)
7    if src == snk then
8      printICFGPath(path)
9    foreach edge ∈ curNode.getOutEdges() do
10     if edge.isIntraCFGEdge() then
11       dfs(edge.dst, snk)
12     else if edge.isCallCFGEdge() then
13       callstack.push_back(edge.getCallSite())
14       dfs(edge.dst, snk)
15       callstack.pop_back()
16     else if edge.isRetCFGEdge() 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()
```

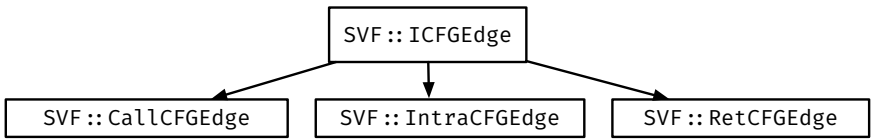
ICFG Node and Edge Classes



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/ICFGEdge.h

`cast` and `dyn_cast`

- C++ Inheritance: see slides in Week 2.
- 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);`
  - `CallCFGEdge* callEdge = SVFUtil::dyn_cast<CallCFGEdge>(ICFGEdge);`