Assignment 2

Yulei Sui

University of Technology Sydney, Australia

Assignment 2: Quizzes + A Coding Task

- Two sets of quizzes (10 ponts)
  - LLVM compiler and its intermediate representation
  - Code graphs (including ICFG and PAG)
- One coding task (10 ponts)
  - **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 ponts)
  - LLVM compiler and its intermediate representation
  - Code graphs (including ICFG and PAG)
- One coding task (10 ponts)
  - **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 1 Context sensitive control-flow reachability

```
Input : src : ICFGNode   dst : ICFGNode
        path : vector⟨ICFGNode⟩   visited : set⟨ICFGNode⟩;
1  dfs(path, src, dst)
2    visited.insert(src)
3    path.push_back(src)
4    if src == dst then
5      print path
6    foreach edge ∈ src.getOutEdges() do
7      if edge.dst ∉ visited then
8        if edge.isIntraCFGEdge() then
9          if handleIntra(edge) then
10           dfs(path, edge.dst, dst)
11       else if edge.isCallCFGEdge() then
12         if handleCall(edge) then
13           dfs(path, edge.dst, dst)
14       else if edge.isRetCFGEdge() then
15         if handleRet(edge) then
16           dfs(path, edge.dst, dst)
17   visited.erase(src)
18   path.pop_back(src)
```

Algorithm 2 Handle intra ICFGEdge

```
1  handleIntra(intraEdge)
2    return true
```

Algorithm 3 Handle call ICFGEdge

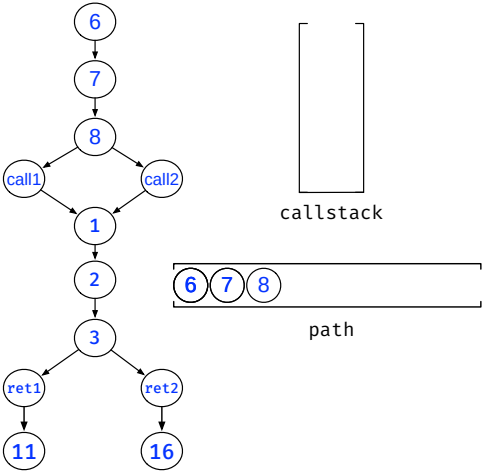
```
1  handleCall(callEdge)
2    callNode ← getSrcNode(callEdge)
3    callstack.push_back(callNode)
4    return true
```

Algorithm 4 Handle return ICFGEdge

```
1  handleRet(retEdge)
2    retNode ← getDstNode(retEdge)
3    if callstack ≠ ∅ then
4      if callstack.back() == getCallICFGNode(retNode) then
5        callstack.pop()
6        return true
7      else
8        return false
9    return true
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



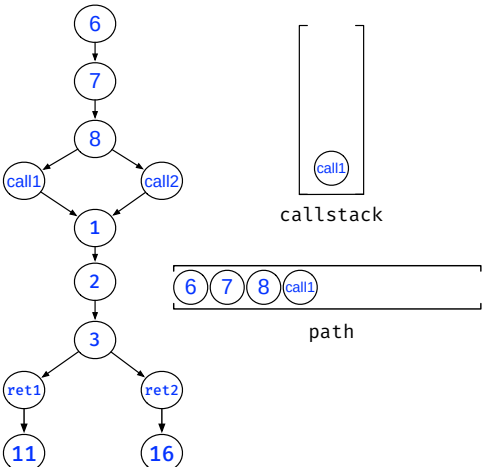
Algorithm 1 Context sensitive control-flow reachability

Input : src : ICFGNode dst : ICFGNode
path : vector⟨ICFGNode⟩ visited : set⟨ICFGNode⟩;

```
1  dfs(path, src, dst)
2  A   visited.insert(src)
3      path.push_back(src)
4  Bmybrown   if src == dst then
5      print path
6      foreach edge ∈ src.getOutEdges() do
7          if edge.dst ∉ visited then
8              if edge.isIntraCFGEdge() then
9                  if handleIntra(edge) then
10                     dfs(path, edge.dst, dst)
11             else if edge.isCallCFGEdge() then
12                 if handleCall(edge) then
13                     dfs(path, edge.dst, dst)
14             else if edge.isRetCFGEdge() then
15                 if handleRet(edge) then
16                     dfs(path, edge.dst, dst)
17     visited.erase(src)
18     path.pop_back(src)
```

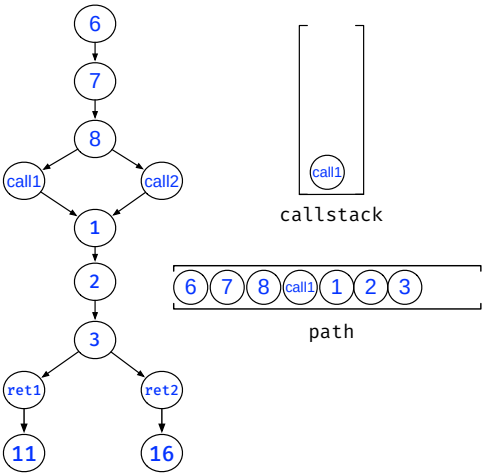
Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



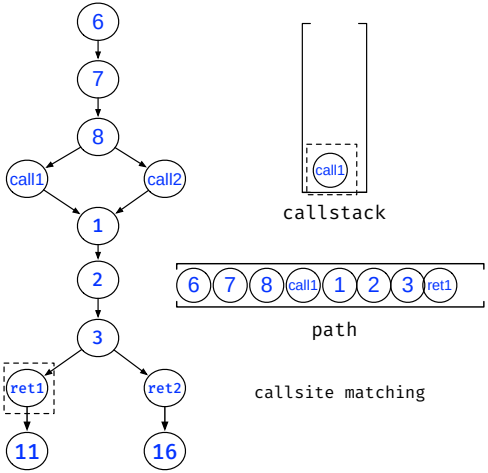
Algorithm 1 Context sensitive control-flow reachability

Input : src : ICFGNode dst : ICFGNode
path : vector⟨ICFGNode⟩ visited : set⟨ICFGNode⟩;

```
1  dfs(path, src, dst)
2  A   visited.insert(src)
3      path.push_back(src)
4  Bmybrown   if src == dst then
5      print path
6      foreach edge ∈ src.getOutEdges() do
7          if edge.dst ∉ visited then
8              if edge.isIntraCFGEdge() then
9                  if handleIntra(edge) then
10                     dfs(path, edge.dst, dst)
11             else if edge.isCallCFGEdge() then
12                 if handleCall(edge) then
13                     dfs(path, edge.dst, dst)
14             else if edge.isRetCFGEdge() then
15                 if handleRet(edge) then
16                     dfs(path, edge.dst, dst)
17     visited.erase(src)
18     path.pop_back(src)
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG

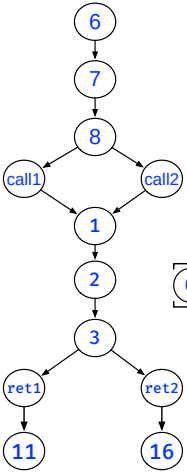


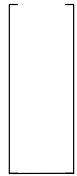
Algorithm 4 Handle return ICFGEdge

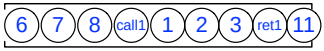
```
1 handleRet(retEdge)
2   retNode ← getDstNode(retEdge)
3   if callstack ≠ ∅ then
4 A   if callstack.back() == getCallICFGNode(retNode)
      then
5       callstack.pop()
6       return true
    B mybrown else
8       return false
9   return true
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG







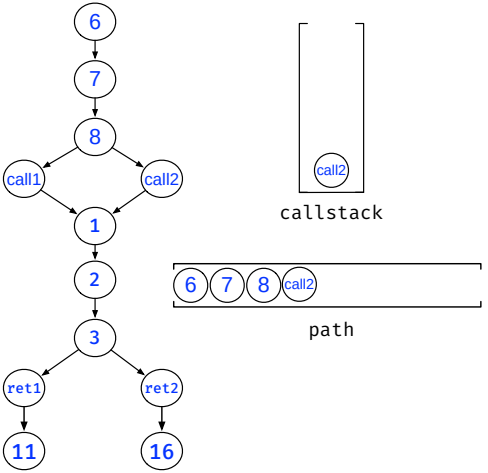
ICFG path:
6→7→8→call1→1→2→3→ret1→11

Algorithm 1 Context sensitive control-flow reachability

```
Input : src : ICFGNode   dst : ICFGNode
        path : vector⟨ICFGNode⟩   visited : set⟨ICFGNode⟩;
1  dfs(path, src, dst)
2    visited.insert(src)
3    path.push_back(src)
4  A   if src == dst then
5        print path
6  Bmybrown   foreach edge ∈ src.getOutEdges() do
7      if edge.dst ∉ visited then
8          if edge.isIntraCFGEdge() then
9              if handleIntra(edge) then
10                 dfs(path, edge.dst, dst)
11         else if edge.isCallCFGEdge() then
12             if handleCall(edge) then
13                 dfs(path, edge.dst, dst)
14         else if edge.isRetCFGEdge() then
15             if handleRet(edge) then
16                 dfs(path, edge.dst, dst)
17   visited.erase(src)
18   path.pop_back(src)
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



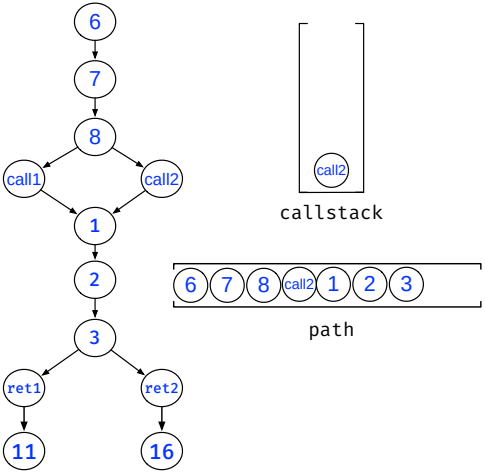
Algorithm 1 Context sensitive control-flow reachability

Input : src : ICFGNode dst : ICFGNode
path : vector⟨ICFGNode⟩ visited : set⟨ICFGNode⟩;

```
1  dfs(path, src, dst)
2    visited.insert(src)
3    path.push_back(src)
4    if src == dst then
5      print path
6    foreach edge ∈ src.getOutEdges() do
7      if edge.dst ∉ visited then
8        if edge.isIntraCFGEdge() then
9          if handleIntra(edge) then
10           dfs(path, edge.dst, dst)
11       A else if edge.isCallCFGEdge() then
12         if handleCall(edge) then
13           dfs(path, edge.dst, dst)
14       B mybrown else if edge.isRetCFGEdge() then
15         if handleRet(edge) then
16           dfs(path, edge.dst, dst)
17   visited.erase(src)
18   path.pop_back(src)
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



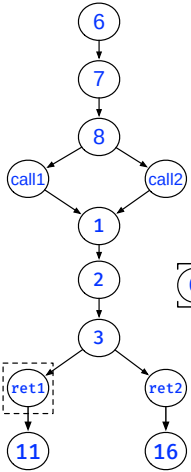
Algorithm 1 Context sensitive control-flow reachability

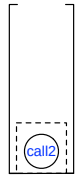
Input : `src : ICFGNode  dst : ICFGNode`
`path : vector⟨ICFGNode⟩  visited : set⟨ICFGNode⟩;`

```
1  dfs(path, src, dst)
2  A   visited.insert(src)
3    path.push_back(src)
4  Bmybrown   if src == dst then
5      print path
6    foreach edge ∈ src.getOutEdges() do
7      if edge.dst ∉ visited then
8          if edge.isIntraCFGEdge() then
9              if handleIntra(edge) then
10                 dfs(path, edge.dst, dst)
11         else if edge.isCallCFGEdge() then
12             if handleCall(edge) then
13                 dfs(path, edge.dst, dst)
14         else if edge.isRetCFGEdge() then
15             if handleRet(edge) then
16                 dfs(path, edge.dst, dst)
17   visited.erase(src)
18   path.pop_back(src)
```

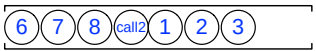
Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG





callstack



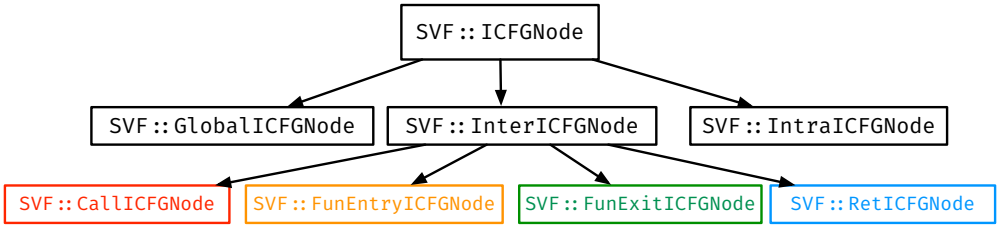
path

CallSite **not** matching

Algorithm 4 Handle return ICFGEdge

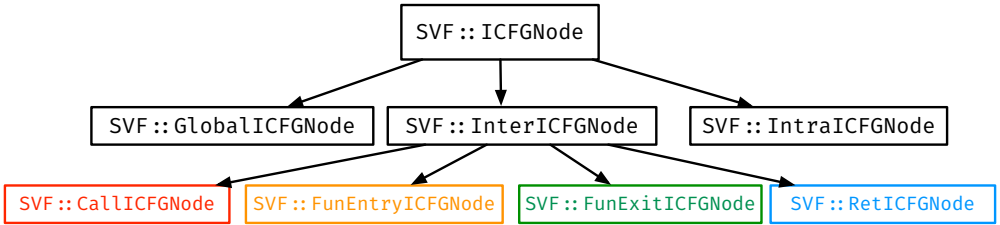
```
1 handleRet(retEdge)
2   retNode ← getDstNode(retEdge)
3   if callstack ≠ ∅ then
4     if callstack.back() == getCallICFGNode(retNode) then
5       callstack.pop()
6       return true
7     A else
8       return false
  B mybrown
9   return true
```

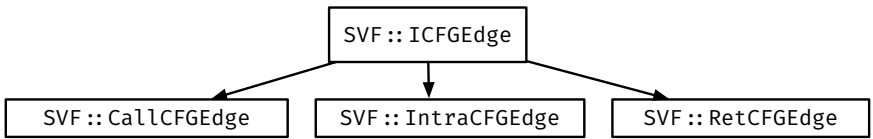
ICFG Node and Edge Classes



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);`