Assignment 2

Yulei Sui

University of Technology Sydney, Australia

Software Analysis   `https://github.com/SVF-tools/Teaching-Software-Analysis`

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
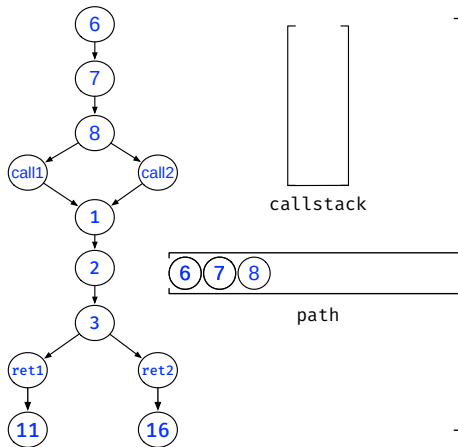
Assignment 2: Quizzes + A Coding Task

- 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-Analysis/wiki/Assignment-2`
  - **SVF CPP API** `https://github.com/SVF-tools/Teaching-Software-Analysis/wiki/SVF-CPP-API`

All the above quizzes and coding task is due by **13th September**. You are encouraged to finish the quizzes before starting your coding task.

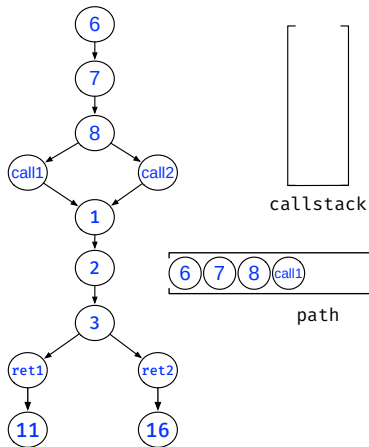# Context-Sensitive Control-Dependence
Obtaining a path from node 6 to node 11 on ICFG


```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.insert(src)
 2 path.push_back(src)
 3 if src == dst then
 4    Print path
 5 foreach edge e ∈ outEdges(src) do
 6    if e.dst ∉ visited then
 7       if e.isIntraCFGEdge() then
 8          DFS(visited, path, callstack, e.dst, dst)
 9       else if e.isCallCFGEdge() then
10          callstack.push(e.getCallsite())
11          DFS(visited, path, callstack, e.dst, dst)
12       else if e.isRetCFGEdge() then
13          if !callstack.empty() && callstack.top()==e.getCallsite() then
14             callstack.pop()
15             DFS(visited, path, callstack, e.dst, dst)
16 visited.erase(src);
17 path.pop_back();
```

Context-Sensitive Control-Dependence (Coding task)
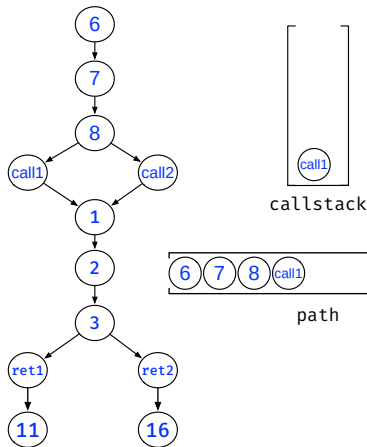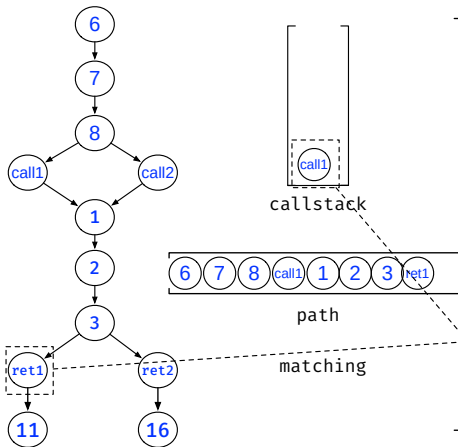
Obtaining a path from node 6 to node 11 on ICFG


```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.insert(src)
 2 path.push_back(src)
 3 if src == dst then
 4   Print path
 5 foreach edge e ∈ outEdges(src) do
 6   if e.dst ∉ visited then
 7     if e.isIntraCFGEdge() then
 8       DFS(visited, path, callstack, e.dst, dst)
 9     else if e.isCallCFGEdge() then
10       callstack.push(e.getCallsite())
11       DFS(visited, path, callstack, e.dst, dst)
12     else if e.isRetCFGEdge() then
13       if !callstack.empty() && callstack.top()==e.getCallsite() then
14         callstack.pop()
15         DFS(visited, path, callstack, e.dst, dst)
16 visited.erase(src);
17 path.pop_back();
```

Context-Sensitive Control-Dependence (Coding task)

Obtaining a path from node 6 to node 11 on ICFG

callstack

path

```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1  visited.insert(src)
 2  path.push_back(src)
 3  if src == dst then
 4    Print path
 5  foreach edge e ∈ outEdges(src) do
 6    if e.dst ∉ visited then
 7      if e.isIntraCFGEdge() then
 8        DFS(visited, path, callstack, e.dst, dst)
 9      else if e.isCallCFGEdge() then
10        callstack.push(e.getCallsite())
11        DFS(visited, path, callstack, e.dst, dst)
12      else if e.isRetCFGEdge() then
13        if !callstack.empty() && callstack.top()==e.getCallsite() then
14          callstack.pop()
15          DFS(visited, path, callstack, e.dst, dst)
16  visited.erase(src);
17  path.pop_back();
```
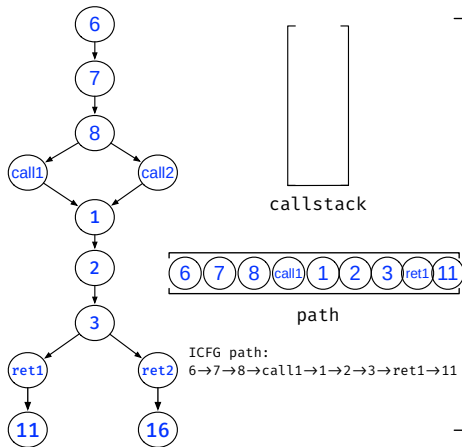
Context-Sensitive Control-Dependence
Obtaining a path from node 6 to node 11 on ICFG

callstack

path

matching

```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.insert(src)
 2 path.push_back(src)
 3 if src == dst then
 4    Print path
 5 foreach edge e ∈ outEdges(src) do
 6    if e.dst ∉ visited then
 7       if e.isIntraCFGEdge() then
 8          DFS(visited, path, callstack, e.dst, dst)
 9       else if e.isCallCFGEdge() then
10          callstack.push(e.getCallsite())
11          DFS(visited, path, callstack, e.dst, dst)
12       else if e.isRetCFGEdge() then
13          if !callstack.empty() && callstack.top()==e.getCallsite() then
14             callstack.pop()
15             DFS(visited, path, callstack, e.dst, dst)
16 visited.erase(src);
17 path.pop_back();
```
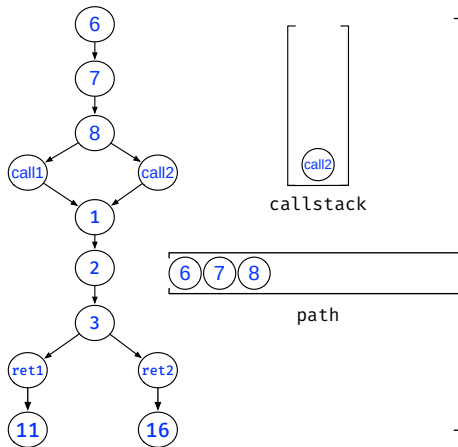
Context-Sensitive Control-Dependence (Coding task)

Obtaining a path from node 6 to node 11 on ICFG

callstack

path

ICFG path:
6→7→8→call1→1→2→3→ret1→11

```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.insert(src)
 2 path.push_back(src)
 3 if src == dst then
 4   Print path
 5 foreach edge e ∈ outEdges(src) do
 6   if e.dst ∉ visited then
 7     if e.isIntraCFGEdge() then
 8       DFS(visited, path, callstack, e.dst, dst)
 9     else if e.isCallCFGEdge() then
10       callstack.push(e.getCallsite())
11       DFS(visited, path, callstack, e.dst, dst)
12     else if e.isRetCFGEdge() then
13       if !callstack.empty() && callstack.top()==e.getCallsite() then
14         callstack.pop()
15         DFS(visited, path, callstack, e.dst, dst)
16 visited.erase(src);
17 path.pop_back();
```

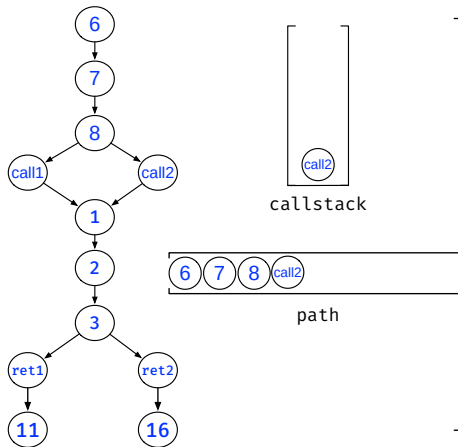# Context-Sensitive Control-Dependence (Coding task)

Obtaining a path from node 6 to node 11 on ICFG

callstack

path

```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.insert(src)
 2 path.push_back(src)
 3 if src == dst then
 4   Print path
 5 foreach edge e ∈ outEdges(src) do
 6   if e.dst ∉ visited then
 7     if e.isIntraCFGEdge() then
 8       DFS(visited, path, callstack, e.dst, dst)
 9     else if e.isCallCFGEdge() then
10       callstack.push(e.getCallsite())
11       DFS(visited, path, callstack, e.dst, dst)
12     else if e.isRetCFGEdge() then
13       if !callstack.empty() && callstack.top()==e.getCallsite() then
14         callstack.pop()
15         DFS(visited, path, callstack, e.dst, dst)
16 visited.erase(src);
17 path.pop_back();
```

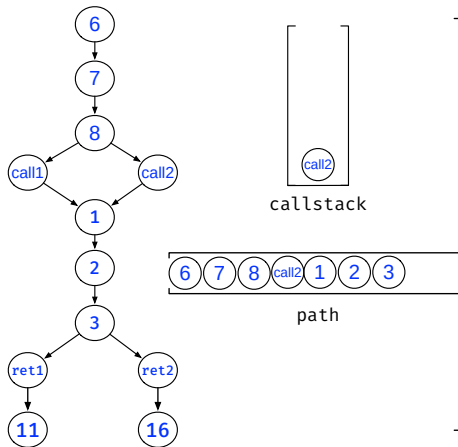# Context-Sensitive Control-Dependence (Coding task)

Obtaining a path from node 6 to node 11 on ICFG

callstack

path

```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.insert(src)
 2 path.push_back(src)
 3 if src == dst then
 4   Print path
 5 foreach edge e ∈ outEdges(src) do
 6   if e.dst ∉ visited then
 7     if e.isIntraCFGEdge() then
 8       DFS(visited, path, callstack, e.dst, dst)
 9     else if e.isCallCFGEdge() then
10       callstack.push(e.getCallsite())
11       DFS(visited, path, callstack, e.dst, dst)
12     else if e.isRetCFGEdge() then
13       if !callstack.empty() && callstack.top()==e.getCallsite() then
14         callstack.pop()
15         DFS(visited, path, callstack, e.dst, dst)
16 visited.erase(src);
17 path.pop_back();
```

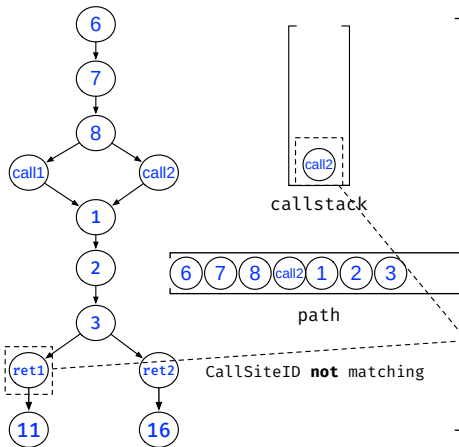# Context-Sensitive Control-Dependence (Coding task)

Obtaining a path from node 6 to node 11 on ICFG

callstack

path

```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.insert(src)
 2 path.push_back(src)
 3 if src == dst then
 4    Print path
 5 foreach edge e ∈ outEdges(src) do
 6    if e.dst ∉ visited then
 7       if e.isIntraCFGEdge() then
 8          DFS(visited, path, callstack, e.dst, dst)
 9       else if e.isCallCFGEdge() then
10          callstack.push(e.getCallsite())
11          DFS(visited, path, callstack, e.dst, dst)
12       else if e.isRetCFGEdge() then
13          if !callstack.empty() && callstack.top()==e.getCallsite() then
14             callstack.pop()
15             DFS(visited, path, callstack, e.dst, dst)
16 visited.erase(src);
17 path.pop_back();
```
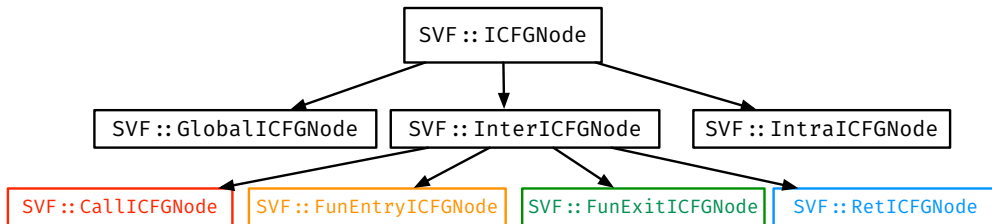
Context-Sensitive Control-Dependence (Coding task)

Obtaining a path from node 6 to node 11 on ICFG
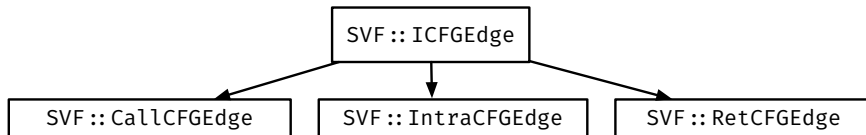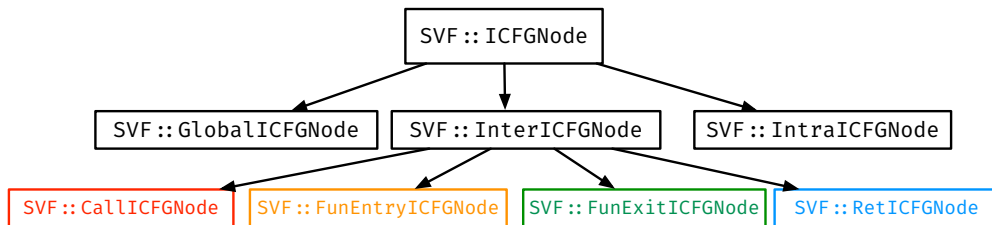


```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.insert(src)
 2 path.push_back(src)
 3 if src == dst then
 4   Print path
 5 foreach edge e ∈ outEdges(src) do
 6   if e.dst ∉ visited then
 7     if e.isIntraCFGEdge() then
 8       DFS(visited, path, callstack, e.dst, dst)
 9     else if e.isCallCFGEdge() then
10       callstack.push(e.getCallsite())
11       DFS(visited, path, callstack, e.dst, dst)
12     else if e.isRetCFGEdge() then
13       if !callstack.empty() && callstack.top()==e.getCallsite() then
14         callstack.pop()
15         DFS(visited, path, callstack, e.dst, dst)
16 visited.erase(src);
17 path.pop_back();
```

callstack

path

CallSiteID **not** matching

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);

14