Assignment 2

Yulei Sui

University of Technology Sydney, Australia

Assignment 2: Control-Dependence

Context-Sensitive ICFG Traversal

- You will be using what you have learned about ICFG and context-sensitive graph traversal.
- **Goal**: implement a context-sensitive graph traversal on ICFG and print **feasible** paths from a source node to a sink node on the graph

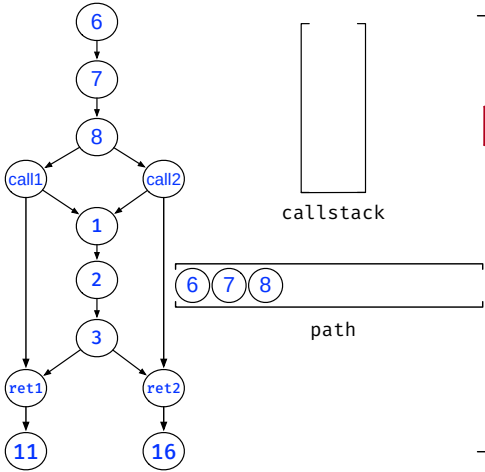
Assignment 2: Control-Dependence

Context-Sensitive ICFG Traversal

- You will be using what you have learned about ICFG and context-sensitive graph traversal.
- **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/SVF-Teaching/wiki/Assignment-2`
- **SVF CPP API**
  `https://github.com/SVF-tools/SVF-Teaching/wiki/SVF-CPP-API`

Context-Sensitive Control-Dependence

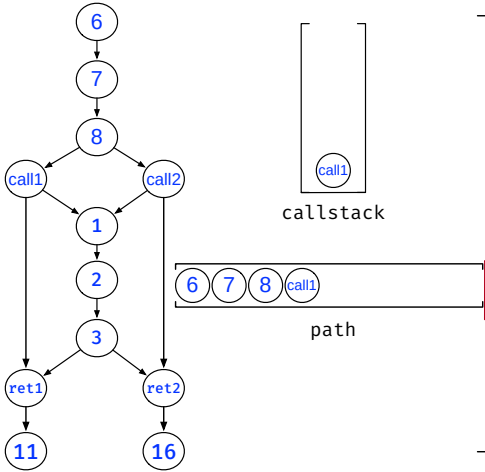
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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

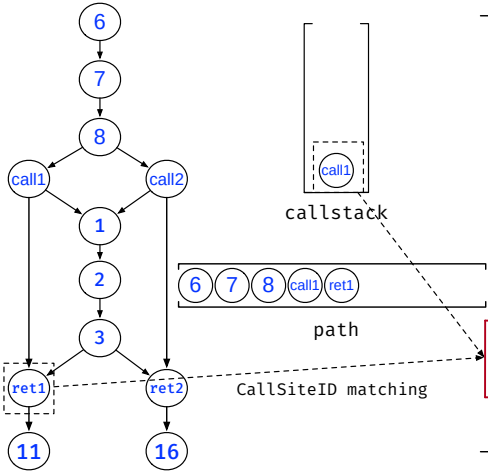
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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

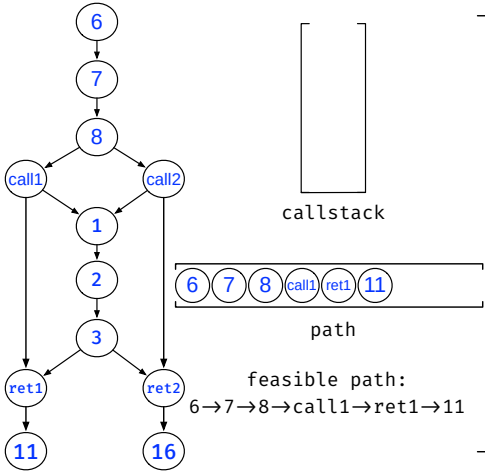
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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

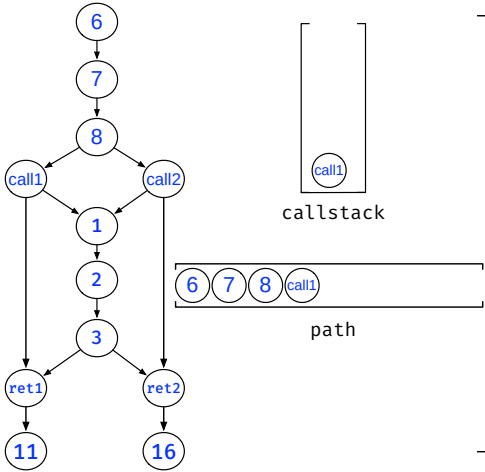
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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

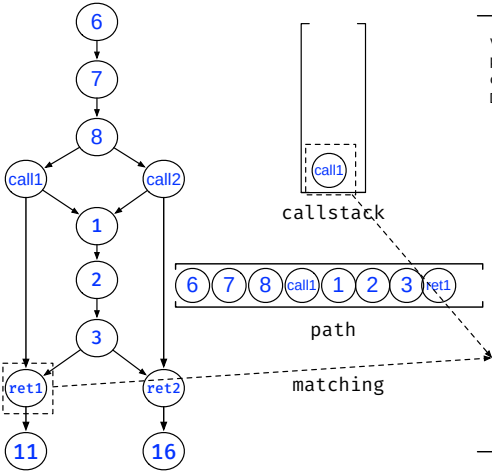
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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

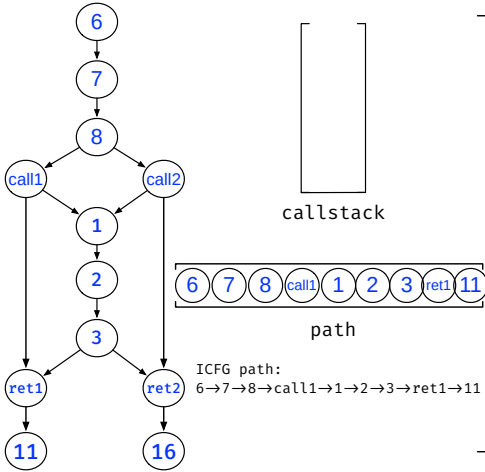
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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

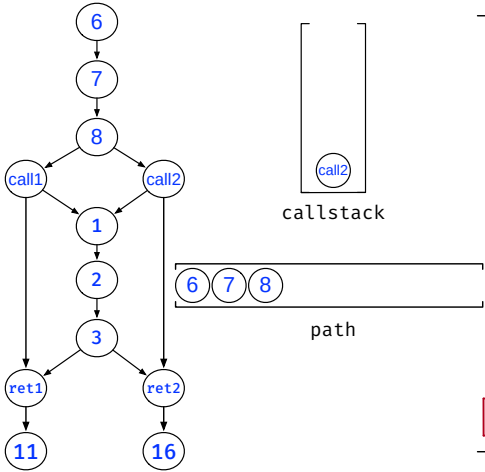
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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

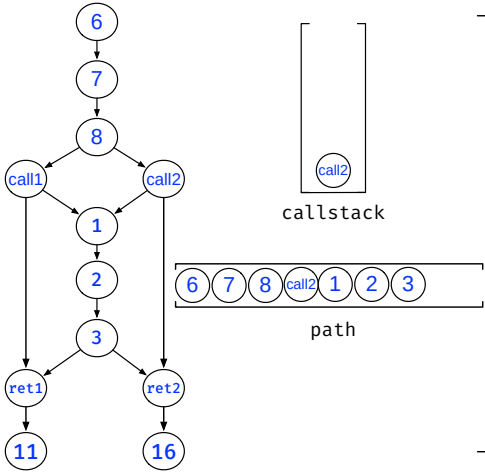
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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

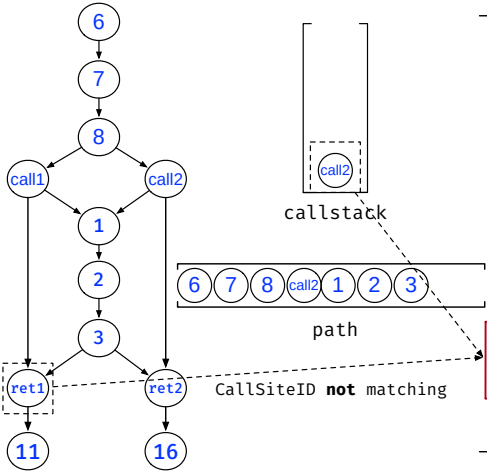
Obtaining a path from node 6 to node 11 on ICFG




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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




```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 1 visited.inser(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();
```