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

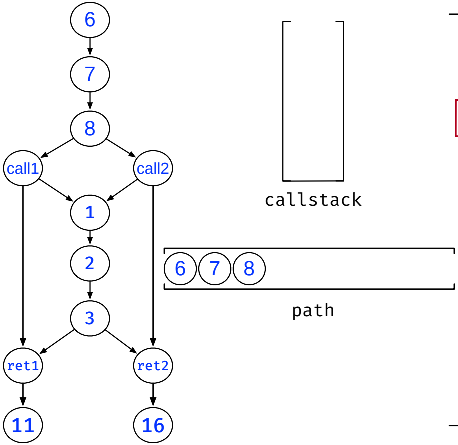
Assignment 2: Control-Dependence

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

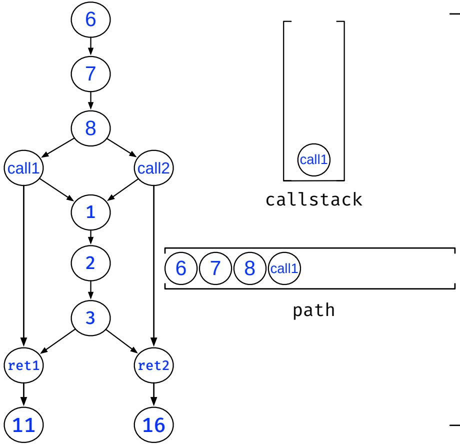
Obtaining a path from node 6 to node 11 on ICFG



```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
visited.inser(src)
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
  if e.dst ∉ visited then
     if e.isIntraCFGEdge() then
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isCallCFGEdge() then
        callstack.push(e.getCallsite())
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isRetCFGEdge() then
         if !callstack.empty() && callstack.top()==e.getCallsite() then
            callstack.pop()
            DFS(visited, path, callstack, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context-Sensitive Control-Dependence

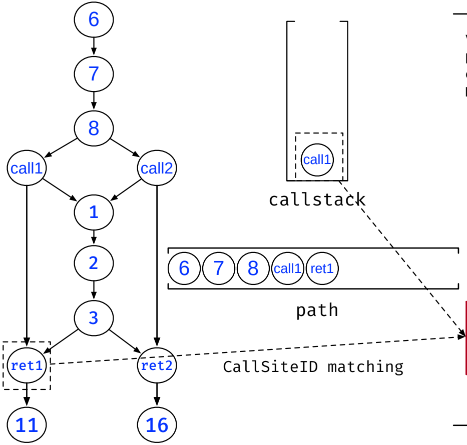
Obtaining a path from node 6 to node 11 on ICFG



```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
visited.inser(src)
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
  if e.dst ∉ visited then
     if e.isIntraCFGEdge() then
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isCallCFGEdge() then
        callstack.push(e.getCallsite())
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isRetCFGEdge() then
         if !callstack.empty() && callstack.top()==e.getCallsite() then
            callstack.pop()
            DFS(visited, path, callstack, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context-Sensitive Control-Dependence

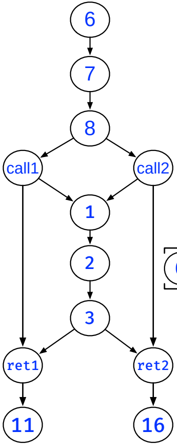
Obtaining a path from node 6 to node 11 on ICFG

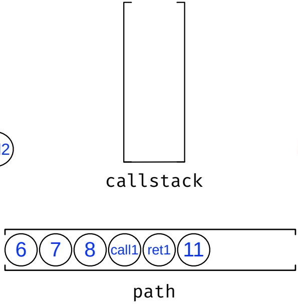


```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 visited.inser(src)
 path.push_back(src)
 if src == dst then
   Print path
 foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
      if e.isIntraCFGEdge() then
         DFS(visited, path, callstack, e.dst, dst)
      else if e.isCallCFGEdge() then
         callstack.push(e.getCallsite())
         DFS(visited, path, callstack, e.dst, dst)
      else if e.isRetCFGEdge() then
         if !callstack.empty() && callstack.top()==e.getCallsite() then
            callstack.pop()
            DFS(visited, path, callstack, e.dst, dst)
 visited.erase(src);
 path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



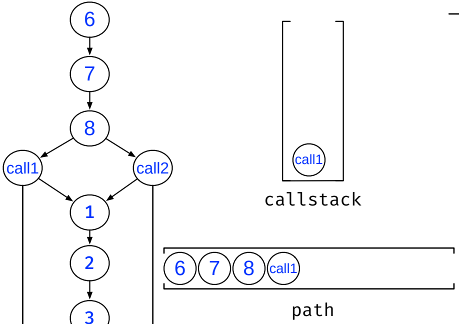


feasible path:
6→7→8→call1→ret1→11

```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
visited.inser(src)
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
  if e.dst ∉ visited then
     if e.isIntraCFGEdge() then
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isCallCFGEdge() then
        callstack.push(e.getCallsite())
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isRetCFGEdge() then
         if !callstack.empty() && callstack.top()==e.getCallsite() then
            callstack.pop()
            DFS(visited, path, callstack, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context-Sensitive Control-Dependence

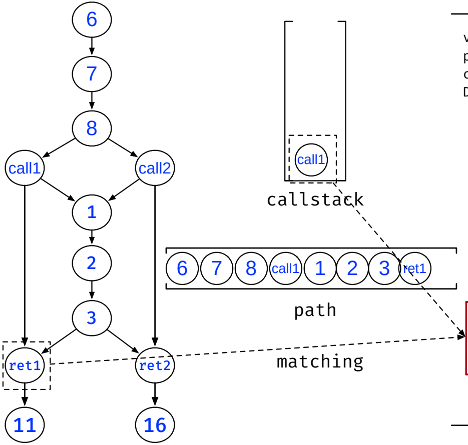
Obtaining a path from node 6 to node 11 on ICFG



```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
visited.inser(src)
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
  if e.dst ∉ visited then
     if e.isIntraCFGEdge() then
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isCallCFGEdge() then
        callstack.push(e.getCallsite())
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isRetCFGEdge() then
        if !callstack.empty() && callstack.top()==e.getCallsite() then
           callstack.pop()
           DFS(visited, path, callstack, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context-Sensitive Control-Dependence

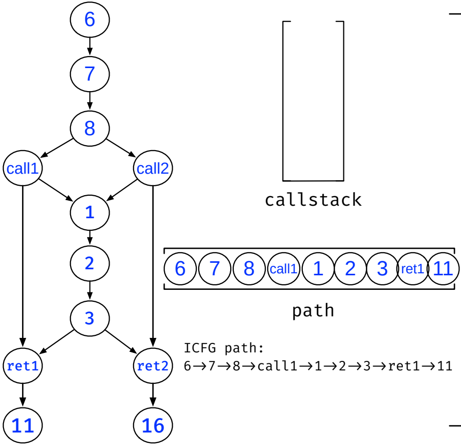
Obtaining a path from node 6 to node 11 on ICFG



```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
visited.inser(src)
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
  if e.dst ∉ visited then
     if e.isIntraCFGEdge() then
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isCallCFGEdge() then
        callstack.push(e.getCallsite())
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isRetCFGEdge() then
        if !callstack.empty() && callstack.top()==e.getCallsite() then
           callstack.pop()
           DFS(visited, path, callstack, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG

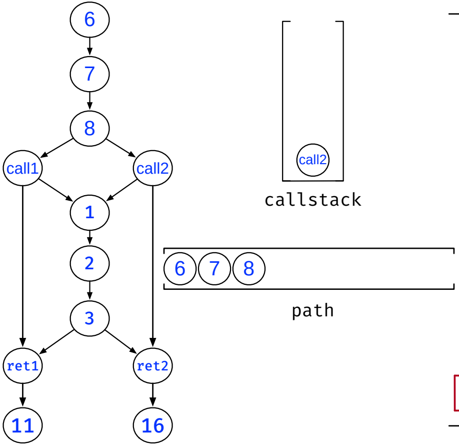


ICFG path:
6→7→8→call1→1→2→3→ret1→11

```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
 visited.inser(src)
 path.push_back(src)
 if src == dst then
   Print path
 foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
      if e.isIntraCFGEdge() then
         DFS(visited, path, callstack, e.dst, dst)
      else if e.isCallCFGEdge() then
         callstack.push(e.getCallsite())
         DFS(visited, path, callstack, e.dst, dst)
      else if e.isRetCFGEdge() then
        if !callstack.empty() && callstack.top()==e.getCallsite() then
           callstack.pop()
           DFS(visited, path, callstack, e.dst, dst)
 visited.erase(src);
 path.pop_back();
```

Context-Sensitive Control-Dependence

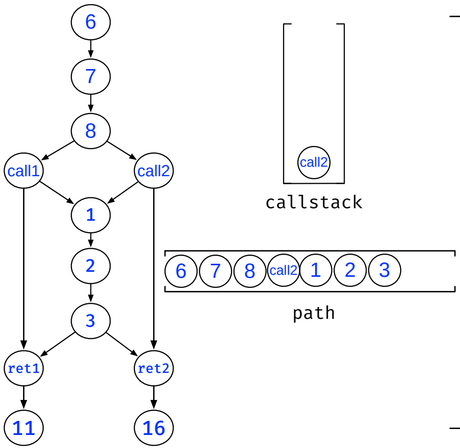
Obtaining a path from node 6 to node 11 on ICFG



```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
visited.inser(src)
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
  if e.dst ∉ visited then
     if e.isIntraCFGEdge() then
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isCallCFGEdge() then
        callstack.push(e.getCallsite())
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isRetCFGEdge() then
         if !callstack.empty() && callstack.top()==e.getCallsite() then
             callstack.pop()
             DFS(visited, path, callstack, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context-Sensitive Control-Dependence

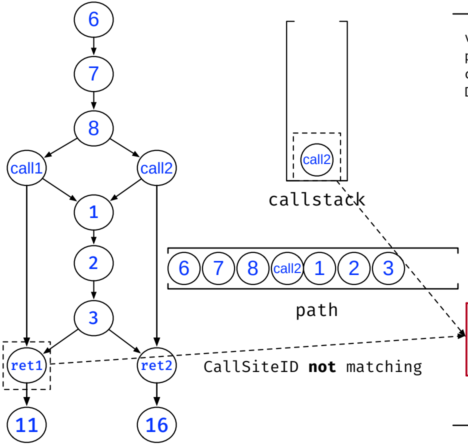
Obtaining a path from node 6 to node 11 on ICFG



```
visited: set<NodeID>
path: vector<NodeID>
callstack: stack<callsite> //A stack of LLVM call instructions
DFS(visited, path, callstack, src, dst)
visited.inser(src)
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
  if e.dst ∉ visited then
     if e.isIntraCFGEdge() then
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isCallCFGEdge() then
        callstack.push(e.getCallsite())
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isRetCFGEdge() then
        if !callstack.empty() && callstack.top()==e.getCallsite() then
           callstack.pop()
           DFS(visited, path, callstack, e.dst, dst)
visited.erase(src);
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)
visited.inser(src)
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
  if e.dst ∉ visited then
     if e.isIntraCFGEdge() then
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isCallCFGEdge() then
        callstack.push(e.getCallsite())
        DFS(visited, path, callstack, e.dst, dst)
     else if e.isRetCFGEdge() then
         if !callstack.empty() && callstack.top()==e.getCallsite() then
            callstack.pop()
            DFS(visited, path, callstack, e.dst, dst)
visited.erase(src);
path.pop_back();
```