Control- and Data-Dependence

Yulei Sui

University of Technology Sydney, Australia

Control- and Data-Dependence

What are control- and data-dependence?

- **Control-dependence**
  - Execution order between two program statements/instructions.
  - Whether program point A reaches point B along the control-flow graph of a program?
  - Can be obtained through traversing on the ICFG of a program
- **Data-dependence**
  - Definition-use relation between two program variables.
  - Whether the definition of a variable X will be used and pass its value to another variable Y?
  - Can be obtained through analyzing the PAG of a program
  - Can also combine PAG with ICFG to yield more precise flow-sensitive and context-sensitive data-dependence.

Control- and Data-Dependence

Why learn control- and data-dependence?

A program dependence relation by its nature is the reachability property on a graph, particularly useful in compiler optimizations and bug detection.

Control- and Data-Dependence

Why learn control- and data-dependence?

A program dependence relation by its nature is the reachability property on a graph, particularly useful in compiler optimizations and bug detection.

- **Applications of control-dependence**
  - Dead code elimination: If a subgraph of an ICFG is not connected from the entry block of a program, that subgraph is possibly dead code.

Control- and Data-Dependence

Why learn control- and data-dependence?

A program dependence relation by its nature is the reachability property on a graph, particularly useful in compiler optimizations and bug detection.

- **Applications of control-dependence**
  - Dead code elimination: If a subgraph of an ICFG is not connected from the entry block of a program, that subgraph is possibly dead code.
  - Identify infinite loops: If the exit block is unreachable from the entry block, an infinite loop may exist.
  - . . .

Control- and Data-Dependence

Why learn control- and data-dependence?

A program dependence relation by its nature is the reachability property on a graph, particularly useful in compiler optimizations and bug detection.

- **Applications of control-dependence**
  - Dead code elimination: If a subgraph of an ICFG is not connected from the entry block of a program, that subgraph is possibly dead code.
  - Identify infinite loops: If the exit block is unreachable from the entry block, an infinite loop may exist.
  - . . .
- **Applications of data-dependence**
  - Pointer alias analysis: statically determine the possible runtime values of a pointer to detect memory errors, such as null pointers and use-after-frees.

Control- and Data-Dependence

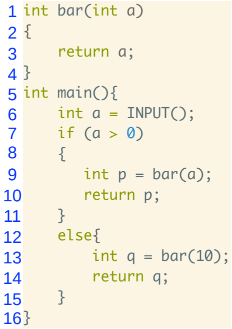
Why learn control- and data-dependence?

A program dependence relation by its nature is the reachability property on a graph, particularly useful in compiler optimizations and bug detection.

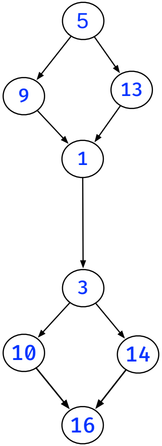
- **Applications of control-dependence**
  - Dead code elimination: If a subgraph of an ICFG is not connected from the entry block of a program, that subgraph is possibly dead code.
  - Identify infinite loops: If the exit block is unreachable from the entry block, an infinite loop may exist.
  - . . .
- **Applications of data-dependence**
  - Pointer alias analysis: statically determine the possible runtime values of a pointer to detect memory errors, such as null pointers and use-after-frees.
  - Taint analysis: if two program variables v1 and v2 are aliases (e.g., representing the same memory location), if v1 is tainted by user inputs, then v2 is also tainted.
  - . . .

Context-Insensitive Control-Dependence

Basic control-dependence traversal



```
 1 int bar(int a)
 2 {
 3     return a;
 4 }
 5 int main(){
 6     int a = INPUT();
 7     if (a > 0)
 8     {
 9        int p = bar(a);
10        return p;
11     }
12     else{
13         int q = bar(10);
14         return q;
15     }
16 }
```



```
visited: set<NodeID>
path: vector<NodeID>

DFS(visited, path, src, dst)
   visited ← visited ∪ {src};
   path.push_back(src);
   if src == dst then
     Print path;
   foreach edge e ∈ outEdges(src) do
      if (e.dst ∉ visited)
          DFS(visited, path, e.dst, dst);
   visited.erase(src);
   path.pop_back();
```

Basic DFS on ICFG: a → f

All possible paths:

a. 5→9→1→3→10→16

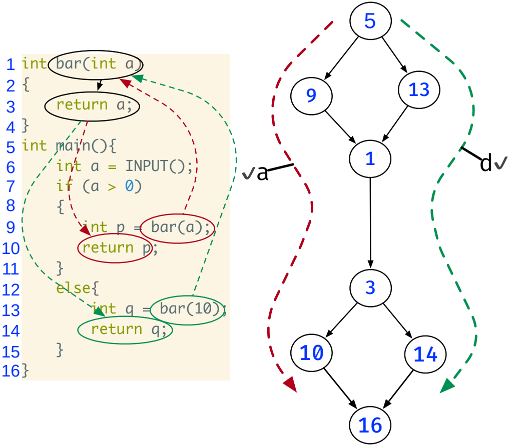
b. 5→9→1→3→14→16

c. 5→13→1→3→10→16

d. 5→13→1→3→14→16

Context Sensitive Control-Dependence

Spurious paths using context-insensitive control-dependence traversal



```
1  int bar(int a)
2  {
3      return a;
4  }
5  int main(){
6      int a = INPUT();
7      if (a > 0)
8      {
9          int p = bar(a);
10         return p;
11     }
12     else{
13         int q = bar(10);
14         return q;
15     }
16 }
```

```
visited: set<NodeID>
path: vector<NodeID>

DFS(visited, path, src, dst)
   visited ← visited ∪ {src};
   path.push_back(src);
   if src == dst then
     Print path;
   foreach edge e ∈ outEdges(src) do
      if (e.dst ∉ visited)
          DFS(visited, path, e.dst, dst);
   visited.erase(src);
   path.pop_back();
```

Basic DFS on ICFG: a → f

All possible paths:

a. 5→9→1→3→10→16 ✓

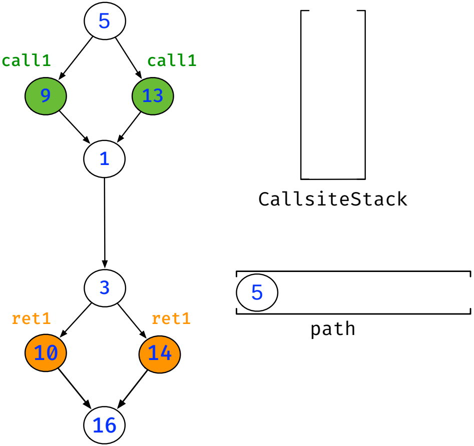
b. 5→9→1→3→14→16 ✗

c. 5→13→1→3→10→16 ✗

d. 5→13→1→3→14→16 ✓

Context Sensitive Control-Dependence

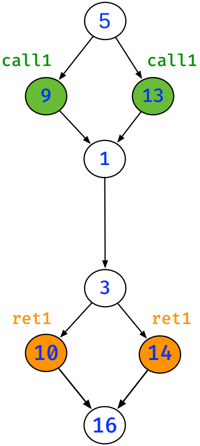
Obtaining a path from node 5 to node 16 on ICFG

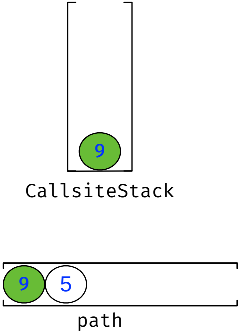


```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG

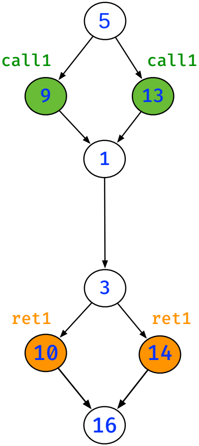


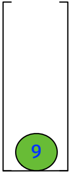


```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

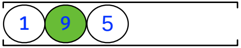
Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG





CallsiteStack

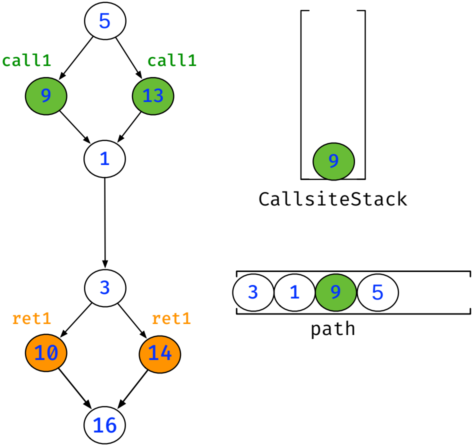


path

```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

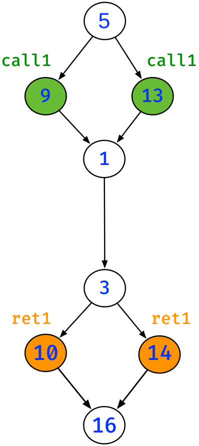
Obtaining a path from node 5 to node 16 on ICFG

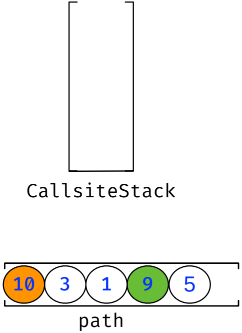


```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG

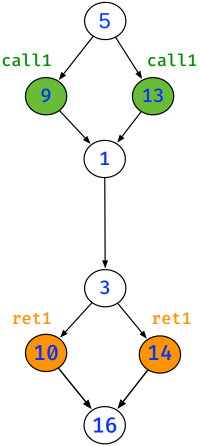


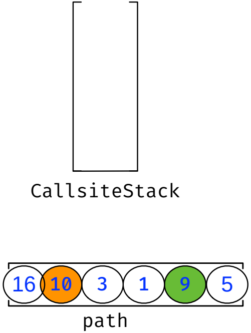


```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
  visited ← visited ∪ {src}
  path.push_back(src)
  if src == dst then
    Print path
  foreach edge e ∈ outEdges(src) do
     if e.dst ∉ visited then
         if e.isIntraEdge() then
               DFS(visited, path, callSiteStack, e.dst, dst)
         else if e.isCallEdge() then
               callsiteStack.push(e.dst.CallSiteID)
               DFS(visited, path, callsiteStack, e.dst, dst)
         else if e.isRetEdge() then
               if callsiteStack.top() == e.dst.CallSiteID then
                   callsiteStack.pop()
                   DFS(visited, path, callsiteStacke, e.dst, dst)
  visited.erase(src);
  path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG



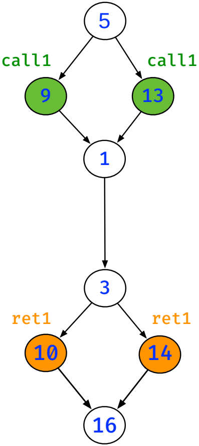


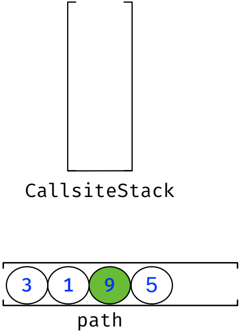
path: 5→9→1→3→10→16

```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG

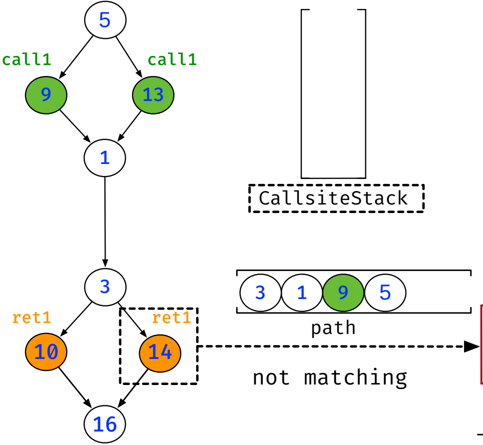




```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

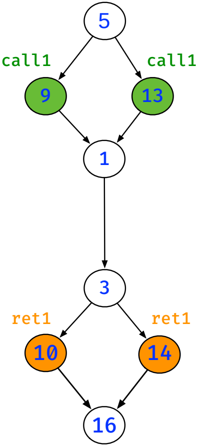
Obtaining a path from node 5 to node 16 on ICFG

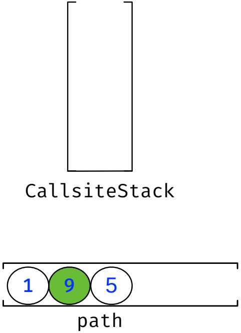


```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
  visited ← visited ∪ {src}
  path.push_back(src)
  if src == dst then
    Print path
  foreach edge e ∈ outEdges(src) do
     if e.dst ∉ visited then
         if e.isIntraEdge() then
               DFS(visited, path, callSiteStack, e.dst, dst)
         else if e.isCallEdge() then
               callsiteStack.push(e.dst.CallSiteID)
               DFS(visited, path, callsiteStack, e.dst, dst)
         else if e.isRetEdge() then
               if callsiteStack.top() == e.dst.CallSiteID then
                   callsiteStack.pop()
                   DFS(visited, path, callsiteStacke, e.dst, dst)
  visited.erase(src);
  path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG

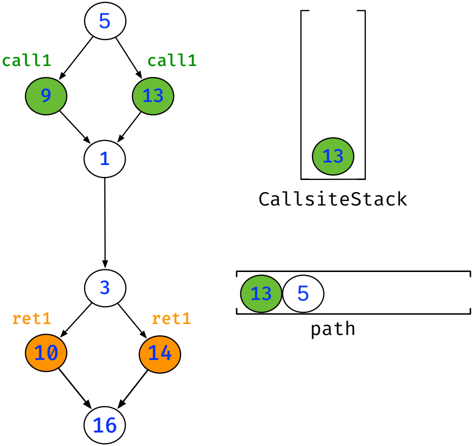




```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

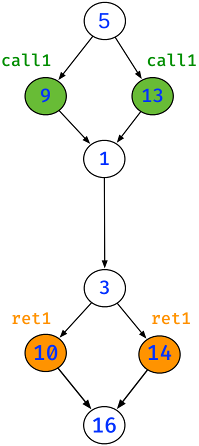
Obtaining a path from node 5 to node 16 on ICFG

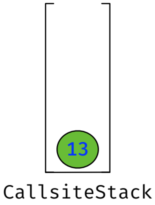


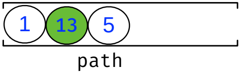
```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG



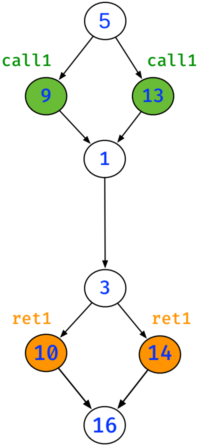


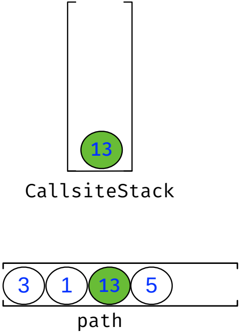


```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG

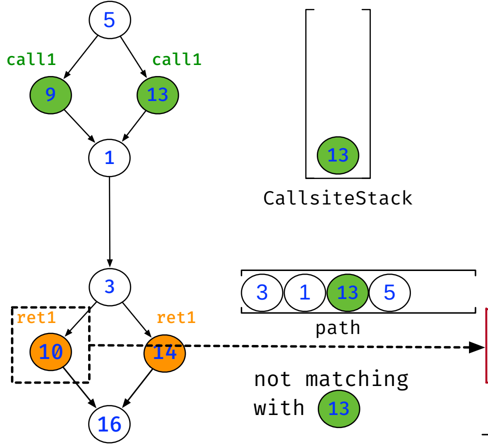




```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG

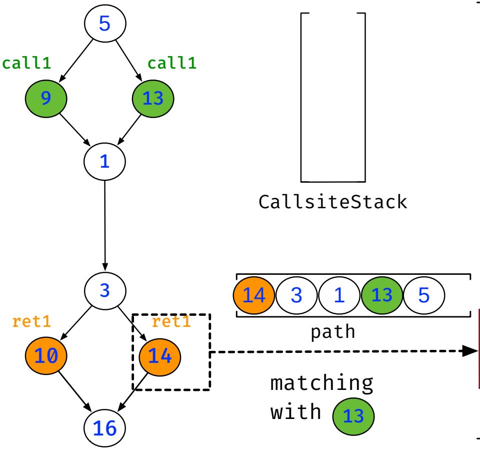


not matching with 13

```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```

Context Sensitive Control-Dependence

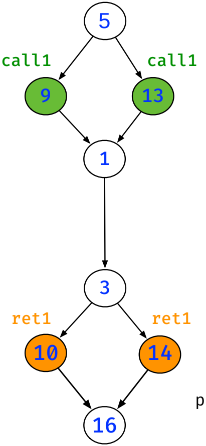
Obtaining a path from node 5 to node 16 on ICFG

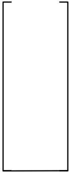


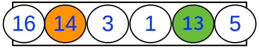
```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
  visited ← visited ∪ {src}
  path.push_back(src)
  if src == dst then
    Print path
  foreach edge e ∈ outEdges(src) do
     if e.dst ∉ visited then
         if e.isIntraEdge() then
               DFS(visited, path, callSiteStack, e.dst, dst)
         else if e.isCallEdge() then
               callsiteStack.push(e.dst.CallSiteID)
               DFS(visited, path, callsiteStack, e.dst, dst)
         else if e.isRetEdge() then
               if callsiteStack.top() == e.dst.CallSiteID then
                   callsiteStack.pop()
                   DFS(visited, path, callsiteStacke, e.dst, dst)
  visited.erase(src);
  path.pop_back();
```

Context Sensitive Control-Dependence

Obtaining a path from node 5 to node 16 on ICFG




CallsiteStack



path

path: 5→13→1→3→14→16

```
visited: set<NodeID>
path: vector<NodeID>
callsiteStack: stack<CallSiteID>
DFS(visited, path, callsiteStack, src, dst)
visited ← visited ∪ {src}
path.push_back(src)
if src == dst then
  Print path
foreach edge e ∈ outEdges(src) do
   if e.dst ∉ visited then
       if e.isIntraEdge() then
             DFS(visited, path, callSiteStack, e.dst, dst)
       else if e.isCallEdge() then
             callsiteStack.push(e.dst.CallSiteID)
             DFS(visited, path, callsiteStack, e.dst, dst)
       else if e.isRetEdge() then
             if callsiteStack.top() == e.dst.CallSiteID then
                 callsiteStack.pop()
                 DFS(visited, path, callsiteStacke, e.dst, dst)
visited.erase(src);
path.pop_back();
```