Lab: Control-Dependence and Reachability

(Week 3)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Quiz-1 + Lab-Exercise-1 + Assignment-1

- A set of quizzes on WebCMS (5 points)
  - LLVM compiler and its intermediate representation
  - Code graphs (including ICFG and PAG)
- Lab-Exercise-1 (5 points)
  - Implement a graph traversal on a general graph
- Assignment-1 (20 points)
  - **Control-flow**: Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and print **feasible** paths from a source node to a sink node on the graph
  - **Data-flow**: Implement Andersen's inclusion-based constraint solving for points-to analysis
  - Implement a taint checker using control-flow analysis and data-flow analysis.

Quiz-1 + Lab-Exercise-1 + Assignment-1

- A set of quizzes on WebCMS (5 points)
  - LLVM compiler and its intermediate representation
  - Code graphs (including ICFG and PAG)
- Lab-Exercise-1 (5 points)
  - Implement a graph traversal on a general graph
- Assignment-1 (20 points)
  - **Control-flow**: Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and print **feasible** paths from a source node to a sink node on the graph
  - **Data-flow**: Implement Andersen's inclusion-based constraint solving for points-to analysis
  - Implement a taint checker using control-flow analysis and data-flow analysis.
  - **Specification and code template**: `https://github.com/SVF-tools/Software-Security-Analysis/wiki/Assignment-1`
  - **SVF APIs for control- and data-flow analysis** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API`

Context-Sensitive Control-Dependence

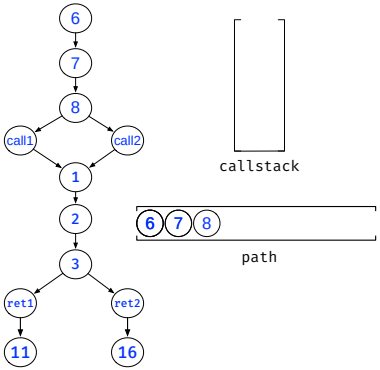
Algorithm 1: 1 Context sensitive control-flow reachability

Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

```
1  dfs(path, curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if ⟨edge.dst, callstack⟩ ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← edge.src;
13         callstack.push_back(callNode.getCallSite());
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19   visited.erase(curItem);
20   path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



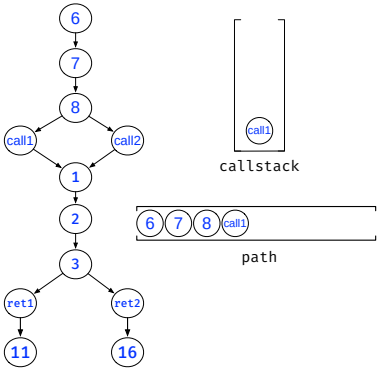
Algorithm 2: Context sensitive control-flow reachability

Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

```
1  dfs(curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if edge.dst ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← getSrcNode(edge);
13         callstack.push_back(callNode);
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19         else if callstack == ∅ then
20           dfs(path, edge, dst)
21   visited.erase(curItem);
22   path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



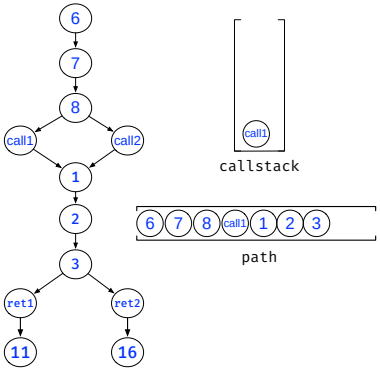
Algorithm 3: Context sensitive control-flow reachability

Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

```
1  dfs(curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if edge.dst ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← getSrcNode(edge);
13         callstack.push_back(callNode);
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19         else if callstack == ∅ then
20           dfs(path, edge, dst)
21   visited.erase(curItem);
22   path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



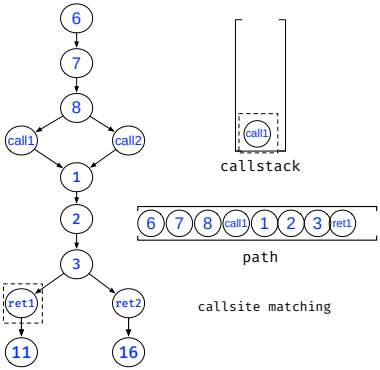
Algorithm 4: 1 Context sensitive control-flow reachability

Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

```
1  dfs(curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if edge.dst ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← getSrcNode(edge);
13         callstack.push_back(callNode);
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19         else if callstack == ∅ then
20           dfs(path, edge, dst)
21   visited.erase(curItem);
22   path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



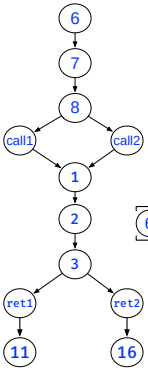
Algorithm 5: 1 Context sensitive control-flow reachability

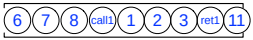
Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

```
1  dfs(curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if edge.dst ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← getSrcNode(edge);
13         callstack.push_back(callNode);
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19         else if callstack == ∅ then
20           dfs(path, edge, dst)
21   visited.erase(curItem);
22   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

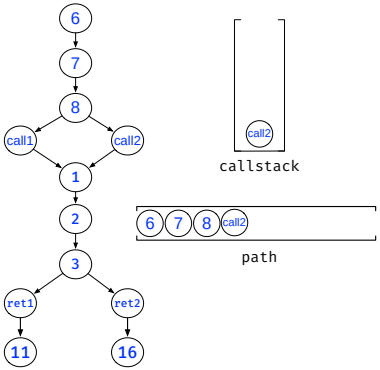
Algorithm 6: 1 Context sensitive control-flow reachability

Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

```
1  dfs(curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if edge.dst ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← getSrcNode(edge);
13         callstack.push_back(callNode);
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19         else if callstack == ∅ then
20           dfs(path, edge, dst)
21   visited.erase(curItem);
22   path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



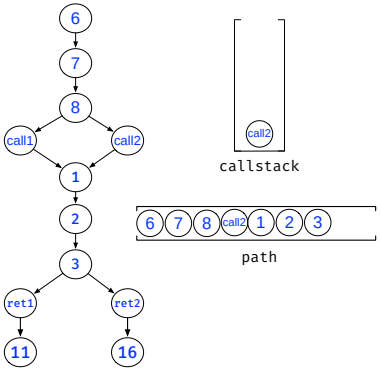
Algorithm 7: 1 Context sensitive control-flow reachability

Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

```
1  dfs(curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if edge.dst ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← getSrcNode(edge);
13         callstack.push_back(callNode);
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19         else if callstack == ∅ then
20           dfs(path, edge, dst)
21   visited.erase(curItem);
22   path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG



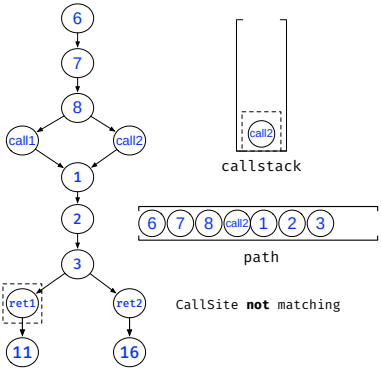
Algorithm 8: Context sensitive control-flow reachability

Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

```
1  dfs(curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if edge.dst ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← getSrcNode(edge);
13         callstack.push_back(callNode);
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19         else if callstack == ∅ then
20           dfs(path, edge, dst)
21   visited.erase(curItem);
22   path.pop_back();
```

Context-Sensitive Control-Dependence

Obtaining a path from node 6 to node 11 on ICFG

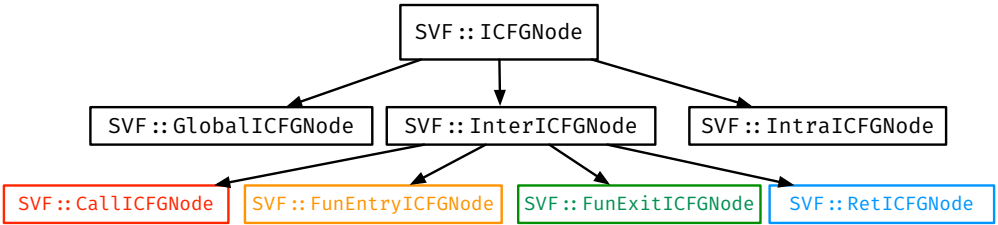


Algorithm 9: Context sensitive control-flow reachability

Input : curEdge : ICFGEdge dst : ICFGNode path : vector⟨ICFGEdge⟩
visited : set⟨ICFGEdge, callstack⟩;

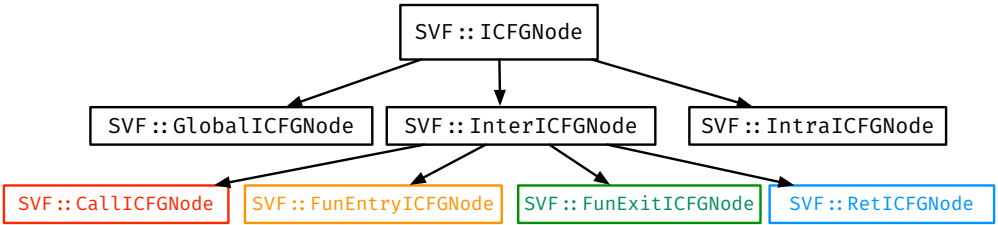
```
1  dfs(curEdge, dst)
2    curItem ← ⟨curEdge, callstack⟩;
3    visited.insert(curItem);
4    path.push_back(curEdge);
5    if src == dst then
6      printICFGPath(path);
7    foreach edge ∈ curEdge.dst.getOutEdges() do
8      if edge.dst ∉ visited then
9        if edge.isIntraCFGEdge() then
10         dfs(path, edge, dst)
11       else if edge.isCallCFGEdge() then
12         callNode ← getSrcNode(edge);
13         callstack.push_back(callNode);
14         dfs(path, edge, dst)
15       else if edge.isRetCFGEdge() then
16         if callstack ≠ ∅ && callstack.back() == edge.getCallSite() then
17           callstack.pop()
18           dfs(path, edge, dst)
19         else if callstack == ∅ then
20           dfs(path, edge, dst)
21   visited.erase(curItem);
22   path.pop_back();
```

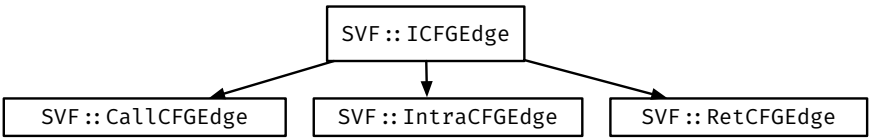
ICFG Node and Edge Classes



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);`