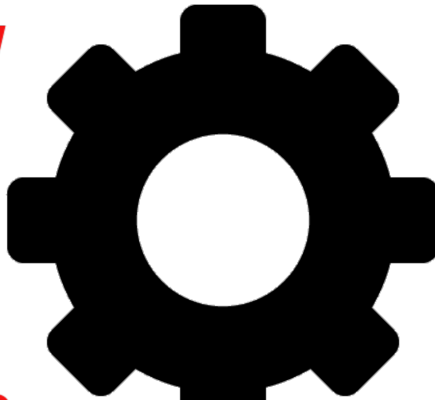
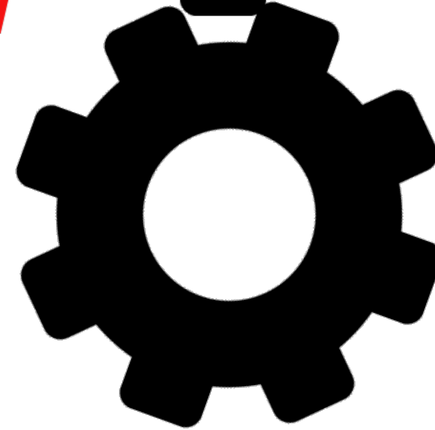
Advanced

T  pics

in

C  mpilers

Call Graph

Simone Campanoni
simone.campanoni@northwestern.edu

Call graph in NOELLE

- Sources:
  src/core/call_graph

- Main headers:
  install/noelle/core/CallGraph.hpp
  install/noelle/core/SCCCAG.hpp

- Examples of passes using the abstraction:
  examples/passes/callgraph
  examples/passes/scccag
  examples/passes/island

Outline

- Call graph (summary from 323)

- Call graph in NOELLE

- Other abstractions generated from call graph in NOELLE

Call graph

- First problem: how do we know what procedures are called from where?
  - Especially difficult in higher-order languages, languages where functions are values
  - What about C programs?
  - We’ll ignore this for now

```
void foo (int a, int (*p_to_f)(int v)){
  int l = (*p_to_f)(5);
  a = l + 1;
  return a;
}
```

- Let’s assume we have a (static) **call graph**
  - Indicates which procedures can call which other procedures, and from which program points

Call graph example

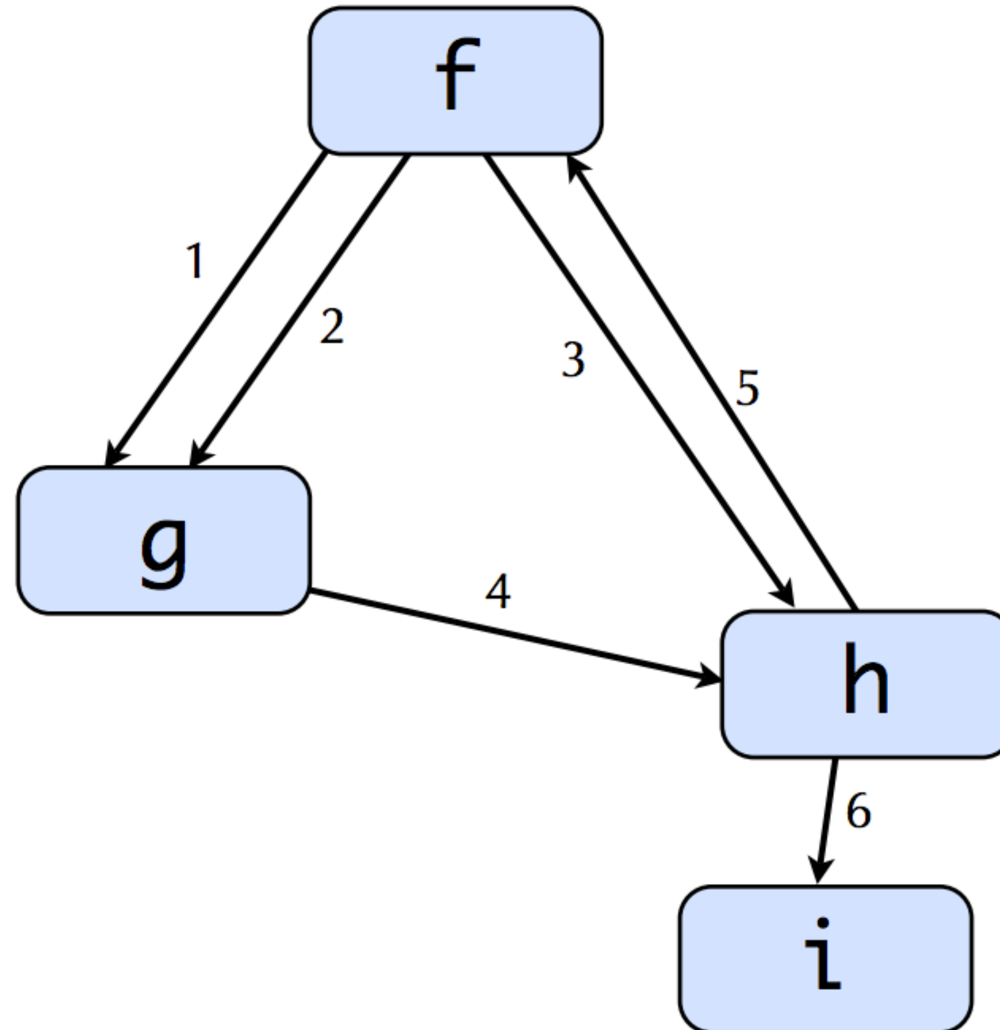
```
f() {
   1:  g();
   2:  g();
   3:  h();
}

g() {
   4: h();
}

h() {
   5: f();
   6: i();
}

i() { … }
```

From now on we assume we have a static call graph



Using CallGraphWrappingPass

- Declaring your pass dependence

```
void getAnalysisUsage(AnalysisUsage &AU) const override {
  AU.addRequired< CallGraphWrapperPass >();
```

- Fetching the call graph

```
bool runOnModule(Module &M) override {
  errs() << "Module \"" << M.getName() << "\"\n";
  CallGraph &CG = getAnalysis<CallGraphWrapperPass>().getCallGraph();
```

Call graph

- how do we know
  what procedures are called from where?
  - Especially difficult in higher-order languages, languages where functions are values
  - What about C programs?

```
void foo (int a, int (*p_to_f)(int v)){
  int l = (*p_to_f)(5);
  a = l + 1;
  return a;
}
```

- Call graph generated by LLVM:
  - If the callee is unknown: no edge is generated
  - If there are N possible callees (N > 1): no edge is generated
  - In other words: the call graph of LLVM **is not** complete

Outline

- Call graph (summary from 323)

- Call graph in NOELLE

- Other abstractions generated from call graph in NOELLE

Call graph in NOELLE

- Called “Program Call Graph (PCG)”
- PCG is complete (and conservative)
- If there are N possible callees ($N > 1$): there are N outgoing edges
- It is a hierarchical graph

Let’s compute the PCG

Normalize the code

Code must be normalized before you use NOELLE

- noelle-norm MYIR.bc –o IR.bc
  or
- noelle-simplification MYIR.bc –o IR.bc

Fetching the program call graph (PCG)

```
/*
 * Fetch NOELLE
 */
auto& noelle = getAnalysis<Noelle>();
```

arcana::noelle::Noelle

```
auto fm = noelle.getFunctionsManager();
```

arcana::noelle::FunctionsManager *

```
auto pcf = fm->getProgramCallGraph();
```

arcana::noelle::CallGraph *

Using the PCG

llvm::Function *

```
/*
 * Fetch the next program's function.
 */
auto f = node->getFunction();
if (f->empty()){
  continue ;
}
```

arcana::noelle::CallGraphFunctionNode *

```
for (auto node : pcf->getFunctionNodes()){
) lines: Fetch the next program's function.-
}
```

```
/*
 * Fetch the outgoing edges.
 */
auto outEdges = pcf->getOutgoingEdges(node);
if (outEdges.size() == 0) {
  errs() << " The function \"" << f->getName() << "\" has no calls\n";
  continue;
}
```

arcana::noelle::CallGraphFunctionFunctionEdge *

arcana::noelle::CallGraphFunctionNode *

```
errs() << " The function \"" << f->getName() << "\"";
errs() << " invokes the following functions:\n";
for (auto callEdge : outEdges){
  auto calleeNode = callEdge->getCallee();
  auto calleeF = calleeNode->getFunction();
lines: errs() << "   [" ;-----------------------------
}
```

llvm::Function *

PCG: from function to node

arcana::noelle::CallGraphFunctionNode *

```
auto mainNode = pcf->getFunctionNode(mainF);
```

llvm::Function *

Edges in the PCG

- All PCG edges are either may or must
  - May:
    when the related call executes,
    the destination of the edge might be called
  - Must:
    when the related call executes,
    the destination of the edge will always execute

LLVM call graph edges

```
if (callEdge->isAMustCall()){
  errs() << "must";
} else {
  errs() << "may";
}
```

PCG of NOELLE is hierarchical

- If a function F invokes G N times,
  the PCG includes only one edge e from F to G
  - Source of e: F
  - Destination of e: G

arcana::noelle::CallGraphFunctionFunctionEdge *

- That edge includes N sub-edges
  - Source of a sub-edge: the specific **call instruction** of F
  - Destination of all sub-edges: **function** G

arcana::noelle::CallGraphInstructionFunctionEdge *

PCG of NOELLE is hierarchical

arcana::noelle::CallGraphFunctionFunctionEdge *

```
/*
 * Print the outgoing edges.
 */
errs() << " The function \"" << f->getName() << "\"";
errs() << " invokes the following functions:\n";
for (auto callEdge : outEdges) {
  auto calleerNode = callEdge->getCaller();
  auto calleeNode = callEdge->getCallee();
  auto calleeF = calleeNode->getFunction();
2 lines: errs() << "    [";-----------------------------
}
```

arcana::noelle::CallGraphFunctionNode *

arcana::noelle::CallGraphInstructionFunctionEdge *

arcana::noelle::CallGraphInstructionNode *

```
for (auto subEdge : callEdge->getSubEdges()){
  auto callerSubEdge = subEdge->getCaller();
  errs() << "      [";
  if (subEdge->isAMustCall()){
    errs() << "must";
  } else {
    errs() << "may";
  }
  errs() << "] " << *callerSubEdge->getInstruction() << "\n";
}
```

This code can be found in noelle/examples/passes/callgraph

Outline

- Call graph (summary from 323)
- Call graph in NOELLE
- Other abstractions generated from call graph in NOELLE

Islands

- Island: disconnected sub-graph of a graph
- Island in the PCG:
  set of functions that **cannot** reach
  from any other function of another island

```
auto islands = pcf->getIslands();
```

```
auto islandOfMain = islands[mainF];
```

This code can be found in
noelle/examples/passes/island

```
for (auto& F : M){
  auto islandOfF = islands[&F];
  if (islandOfF != islandOfMain){
    errs() << " Function " << F.getName() << " is not in the same island of main\n";
  }
}
```

Strongly Connected Component Call Acyclic Graph (SCCCAG)

```
/*
 * Fetch the entry point.
 */
auto fm = noelle.getFunctionsManager();

/*
 * Fetch the SCCDAG of the program call graph: SCCCAG
 */
auto sccCAG = fm->getSCCDAGOfProgramCallGraph();
```

This code can be found in noelle/examples/passes/scccag

Strongly Connected Component Call Acyclic Graph (SCCCAG)

```
/*
 * Print the nodes of the SCCCAG.
 */
errs() << "SCCCAG:   Nodes\n";
for (auto node : sccCAG->getNodes()) {

  /*
   * Print the node.
   */
  errs() << "SCCCAG:      " << node->getID() << ": ";
  if (node->isAnSCC()) {
    auto sccNode = static_cast<SCCCAGNode_SCC *>(node);
    errs() << "SCC\n";
    errs() << "SCCCAG:        Internal nodes:\n";
    for (auto internalNode : sccNode->getInternalNodes()) {
      auto f = internalNode->getFunction();
      errs() << "SCCCAG:          " << f->getName() << "\n";
    }

  } else {
    auto fNode = static_cast<SCCCAGNode_Function *>(node);
    errs() << "Function " << fNode->getNode()->getFunction()->getName()
           << "\n";
  }
}
```

arcana::noelle::SCCCAGNode *

arcana::noelle::CallGraphFunctionNode *

Strongly Connected Component Call Acyclic Graph (SCCCAG)

```
/*
 * Print the outgoing edges.
 */
errs() << "SCCCAG:   Edges\n";
for (auto node : sccCAG->getNodes()) {
  for (auto dstNodePair : sccCAG->getOutgoingEdges(node)) {
    auto edge = dstNodePair.second;
    auto dstNode = edge->getDst();
    errs() << "SCCCAG:     " << node->getID() << " -> " << dstNode->getID()
           << "\n";

    /*
     * Print the sub-edges.
     */
    errs()
        << "SCCCAG:       Because of the following edges in the call graph:\n";
    for (auto subEdge : edge->getSubEdges()) {
      auto callerNode = subEdge->getCaller();
      auto calleeNode = subEdge->getCallee();
      auto calleer = callerNode->getFunction();
      auto callee = calleeNode->getFunction();
      errs() << "SCCCAG:         \"" << calleer->getName()
             << "\" invokes \"" << callee->getName() << "\"\n";
    }
  }
}
```

arcana::noelle::SCCCAGNode *

arcana::noelle::SCCCAGEdge *

arcana::noelle::SCCCAGNode *

arcana::noelle::CallGraphFunctionFunctionEdge *

Always have faith in your ability

Success will come your way eventually

Best of luck!