Lecture 15
Control Dependence Graphs

Kenneth M. Anderson
CSCI 5828, Spring 2000
This lecture comes from...

The Roadmap

- Introduction to Dependence Analysis
- Current State of Affairs and Limitations
- Judy's Approach -- A Compositional Model
- Related Work

A Compositional Approach to Interprocedural Control Dependence Analysis

Judith Stafford
Software Engineering Research Laboratory
Department of Computer Science
University of Colorado at Boulder
http://www.cs.colorado.edu/serl

My Big Program Doesn't Work

Why is the printed VALUE wrong?

print VALUE
But Where To Look?

I wish I knew where to concentrate my search.

Simple Bug Tracking Example

Question: What statements of Simple could contain the bug that causes it to always print “1”?  
– Answer: 1, 2, or 4  
– How do we find the answer?

Getting Started

Best to start your search where the bad value is printed -- statement 5  
It looks like the value used at statement 5 comes from statement 4

Program Simple
1: read i
2: if ( i == 1)
3: print “POS:”
   else
4: i = 1
5: print i
6: end

Conditional Execution

But then you notice that statement 4 might not even be executed because it depends on the decision made at statement 2

Program Simple
1: read i
2: if ( i == 1)
3: print “POS:”
   else
4: i = 1
5: print i
6: end
**Variable Assignment**

- The decision made at statement 2 depends on what value is input at statement 1
- The value printed at 5 may come directly from statement 1

```
Program Simple
1: read i
2: if (i == 1)
3:   print “POS:”
   else
4:   i = 1
5: print i
6: end
```

**The Answer**

- So only statements 1, 2, and 4 could contain the bug…
- ✷ This is helpful

```
Program Simple
1: read i
2: if (i == 1)
3:   print “POS:”
   else
4:   i = 1
5: print i
6: end
```

**The Big Question**

How do we automatically identify dependencies in REAL program code?

```
Program Simple
1: read i
2: if (i == 1)
3:   print “POS:”
   else
4:   i = 1
5: print i
6: end
```

**A Graph-Based Model**

- Control Flow Graph
- Forward Dominance Tree
- Data Dependence Graph
- Control Dependence Graph
- Program Dependence Graph
A Graph-Based Model

Control Flow Graph ➔ Forward Dominance Tree

Data Dependence Graph ➔ Control Dependence Graph ➔ Program Dependence Graph

A Control Dependence Representation

- Represent control dependencies in a control dependence graph, “CDG”

Program Simple
1: read i
2: if (i == 1)
3:   print “POS:”
   else
4:   i = 1
5:   print i
6: end

CDG
entry
1 ➔ 2 ➔ 5 ➔ 6
3 ➔ 4

A Control Dependence Representation

- If statement X determines whether statement Y is executed, statement Y is control dependent on statement X

Program Simple

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: read i</td>
<td>2: if (i == 1)</td>
</tr>
<tr>
<td>3: print “POS:”</td>
<td>4: i = 1</td>
</tr>
<tr>
<td>else</td>
<td>5: print i</td>
</tr>
<tr>
<td>6: end</td>
<td>6: end</td>
</tr>
</tbody>
</table>

CDG
entry
1 ➔ 2 ➔ 5 ➔ 6
3 ➔ 4

A Control Dependence Representation

- If statement X determines whether statement Y is executed, statement Y is control dependent on statement X

Program Simple

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: read i</td>
<td>2: if (i == 1)</td>
</tr>
<tr>
<td>3: print “POS:”</td>
<td>4: i = 1</td>
</tr>
<tr>
<td>else</td>
<td>5: print i</td>
</tr>
<tr>
<td>6: end</td>
<td>6: end</td>
</tr>
</tbody>
</table>
A Control Dependence Representation

- Statements that are guaranteed to execute are control dependent on entry to the program

Program Simple

```
1: read i
2: if ( i == 1)
3:   print "POS:"
   else
4:    i = 1
5:    print i
6:   end
```

A Graph-Based Model

- The control flow graph, "CFG"
  - Vertices represent executable statements
  - One entry and one exit
  - Arcs represent potential control flow

```
Program Simple

1: read i
2: if (i == 1)
3:   print "POS:"
   else
4:    i = 1
5:    print i
6:   end
```

A Graph Representation of Behavior

```
1: read i
2: if (i == 1)
3:   print "POS:"
   else
4:    i = 1
5:    print i
6:   end
```

Calculating Control Dependencies

How can we use CFGs to generate CDGs?
Calculating Control Dependencies

The control dependents come after a decision and before a junction...

Use the dominance tree (in reverse)!

A Graph-Based Model

- Vertices represent executable statements
- Arcs represent immediate forward dominance
- The root of the tree is the exit of the CFG

Forward Dominance (a.k.a. post dominance, inverse dominance)

The forward dominance tree, “FDT”
Forward Dominance (a.k.a. post dominance, inverse dominance)

- **Y forward dominates X** if all paths from X include Y

Program Simple:
1: read i
2: if (i == 1)
3: print “POS:”
   else
4: i = 1
5: print i
6: end

Notice that the control dependents, 3 and 4, don’t forward dominate 2

Program Simple:
1: read i
2: if (i = 1)
3: print “POS:”
   else
4: i = 1
5: print i
6: end

Forward Dominance Tree

- The first forward dominator of X is called the **immediate forward dominator of X**, “ifdom(X)”
- Vertices between X and ifdom(X) are dependent on X
- Immediate forward dominators form a tree, “FDT”

Program Simple:
1: read i
2: if (i = 1)
3: print “POS:”
   else
4: i = 1
5: print i
6: end

A Graph-Based Definition

- **Y is control dependent on X** ⇔ there is a path in the CFG from X to Y that doesn’t contain the **immediate forward dominator** of X

![CFG and FDT diagrams]

How Does This Help?

- Now we have half of the answer

CDG

“But then you notice that statement 4 might not even be executed because it depends on the decision made at statement 2”

![CDG diagram]
A Graph-Based Model

- Control Flow Graph
- Forward Dominance Tree
- Control Dependence Graph
- Data Dependence Graph
- Program Dependence Graph

Real Programs are More Complex

- CFG-based definitions and algorithms expect a connected graph
- Procedure-level control flow graphs are not connected because there is no direct flow from a call to the next statement

The Roadmap

- Introduction to Dependence Analysis
- Judy’s Approach -- A Compositional Model
- Related Work

Other Models

Uni-Procedure Model
- Program
- CFG → FDT
- CDG

Multi-Procedure Approaches

LIMITED
Other Models

Uni-Procedure Model

Multi-Procedure Approaches

In-lined Approach

Program

Proc A

Proc Z

CDG

CDG

CDG

CDG

CDG

IICFG ➔ FDT

SCDG

LIMITED

IMPRACTICAL

Other Models

Uni-Procedure Model

Multi-Procedure Approaches

In-lined Approach

Program

Proc A

Proc Z

CDG

CDG

CDG

CDG

CDG

IICFG ➔ FDT

SCDG

LIMITED

IMPRACTICAL

1-1 Graph Approach

Program

Proc A

Proc Z

CDG

CDG

CDG

CDG

CDG

IICFG ➔ FDT

SCDG

AD HOC

The Roadmap

✔ Introduction to Dependence Analysis
✔ Current State of Affairs and Limitations
➽ My Approach -- A Compositional Model
◆ Related Work

Guiding Principles

◆ Question…
  – Can I extend the forward dominator relation to create a practical and straight-forward model of control dependencies that addresses the pitfalls?

◆ Approach
  – Compositional
    » Reason about properties of procedures independently
    » Compose procedure-based representations to reflect program-wide properties
  – Language-independent
    » Modern programs are composed of parts written in different languages
  – Generalizable
    » Limitations and power are precisely defined
A New View - A Compositional Model

Program Multi
1. Proc M
2. call B
3. i = i + 1
4. return
1. Proc B
2. return

A Graph Representation of Structure

Procedure-level Structures

Keep track of the potential indirect flows and forward dominances

Procedure control flow graph ⇒
Forward Dominance Forest ⇒

Procedure Call Graph

Program-level Structures

Apply program call graph to resolve p-fdom arcs and identify interprocedural dependencies

Program Call Graph ⇒

Procedure B and forward dominators of 2 inherit control dependence from 2
Example -- Program Sum

1: proc M
2: read i, j
3: sum = 0
4: while i < 10 do
5: call B
6: call B
7: print sum
8: return
9: proc B
10: call C
11: if j >= 0 then
12: sum = sum + j
13: read j
14: i = i + 1
15: return
16: proc C
17: if sum > 100 then
18: print "done"
19: return

Control Dependencies in Program Sum

 Call Graph for Program Sum
 Sum’s Compound Control Dependence Graph

Forward Dominator Forest

Resolve p-fdom arcs after checking to see if the return vertex of the called procedure forward dominates the procedure’s entry vertex

Introducing an Embedded Halt

1: proc M
2: read i, j
3: sum = 0
4: while i < 10 do
5: call B
6: call B
7: print sum
8: return
9: proc B
10: call C
11: if j >= 0 then
12: sum = sum + j
13: read j
14: i = i + 1
15: return
16: proc C
17: if sum > 100 then
18: HALT

Procedure inherits control dependence of call
Forward Dominator Forest -- with EHalt

- Resolve p-fdom arcs after checking to see if the return vertex of the called procedure forward dominates the procedure's entry vertex.

Effect of Halt on Control Dependence

- Call Graph for Program Sum
- Sum's Compound Control Dependence Graph
  - Procedure inherits control dependence of call/return sites

Related Work

- Researchers have extended the CFG and generate the CDG in *ad hoc* ways to apply to complex programs

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uni-procedure</td>
<td>CFG + FDT = CDG</td>
<td>Podgurski+90, Ferrante+87</td>
</tr>
<tr>
<td>Multi-procedure</td>
<td>xFG ~ xCDG</td>
<td>Horwitz+90, Loyall+93, Harrold+98,99, Liao+99</td>
</tr>
<tr>
<td>Object-oriented</td>
<td>xFG ~ xCDG</td>
<td>Larsen+96, Zhao+96</td>
</tr>
<tr>
<td>Concurrent</td>
<td>xFG ~ xCDG</td>
<td>Zhao+96</td>
</tr>
<tr>
<td>Concurrent-OO</td>
<td>xFG ~ xCDG</td>
<td>Hatcliff+99, Zhao+99</td>
</tr>
<tr>
<td>Reactive</td>
<td>xFG ~ xCDG</td>
<td>Clarke+99, Stafford+98</td>
</tr>
</tbody>
</table>