Lecture 15
Control Dependence Graphs

Kenneth M. Anderson

CSCI 5828, Spring 2000

This lecture comes from...

A Compositional Approach to Interprocedural Control Dependence Analysis

Judith Stafford

Software Engineering Research Laboratory

Department of Computer Science

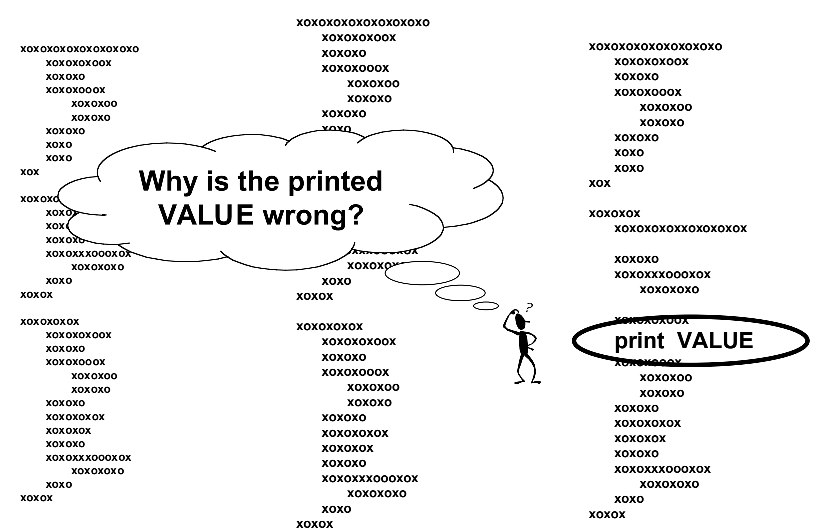
University of Colorado at Boulder

http://www.cs.colorado.edu/serl

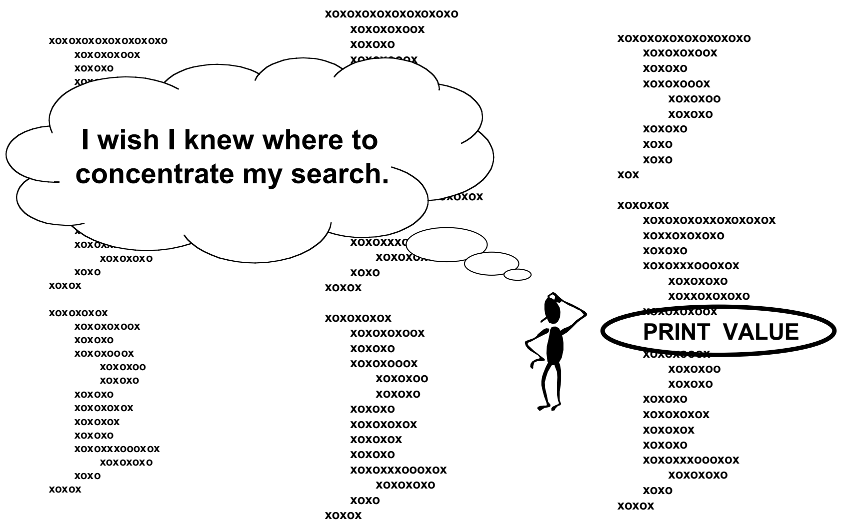
The Roadmap

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

My Big Program Doesn't Work



But Where To Look?



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?

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


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
```

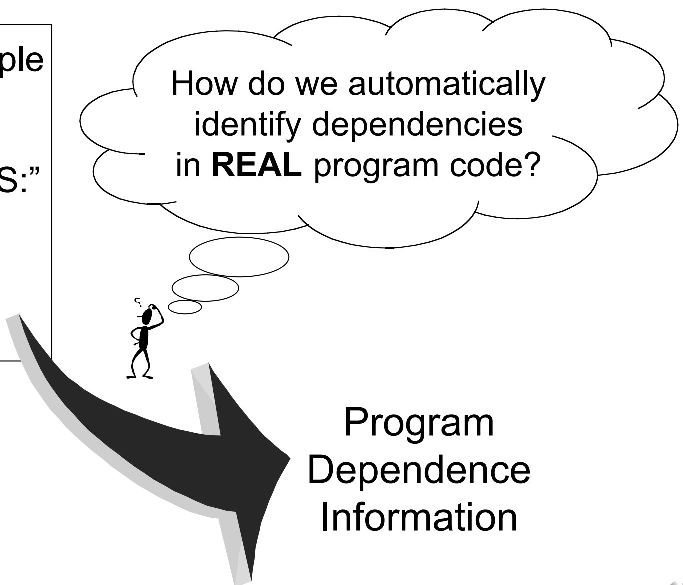

The Answer

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

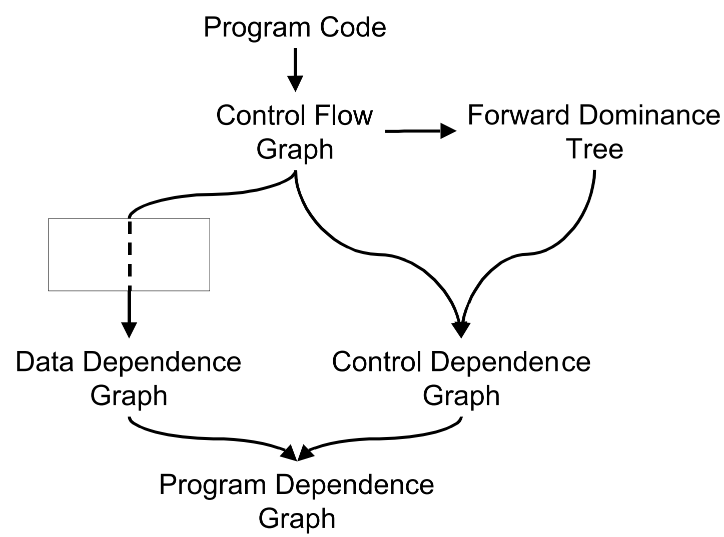
```
Program Simple
1: read i
2: if ( i == 1)
3:    print "POS:"
   else
4:    i = 1
5: print i
6: end
```


The Big Question

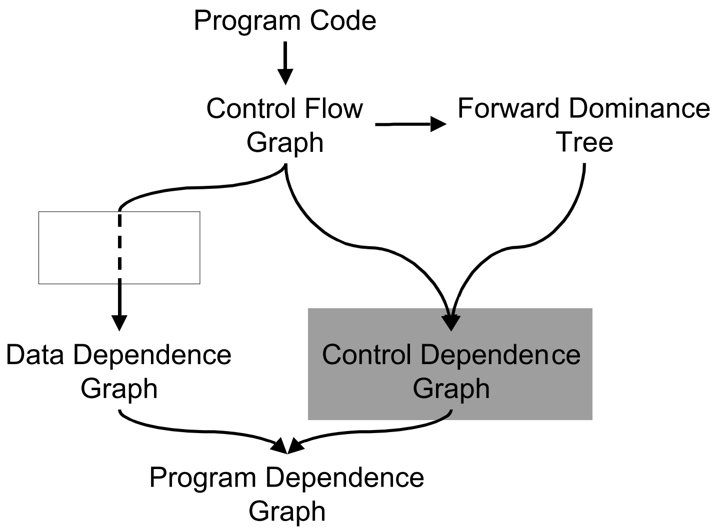
```
Program Simple
1: read i
2: if ( i == 1)
3:    print "POS:"
   else
4:    i = 1
5: print i
6: end
```



A Graph-Based Model



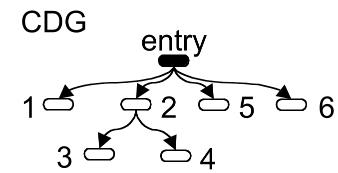
A Graph-Based Model



A Control Dependence Representation

- Represent control dependencies in a *control dependence graph, "CDG"*
  - Vertices represent executable statements
  - Arcs represent direct control dependence
  - A distinguished entry vertex representing the start of the program

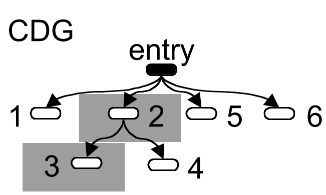
```
Program Simple
1: read i
2: if ( i == 1)
3:    print "POS:"
   else
4:    i = 1
5: print i
6: end
```



A Control Dependence Representation

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

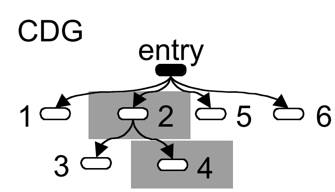
```
     Program Simple
     1: read i
X -> 2: if ( i == 1)
Y -> 3:    print "POS:"
        else
     4:    i = 1
     5: print i
     6: end
```



A Control Dependence Representation

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

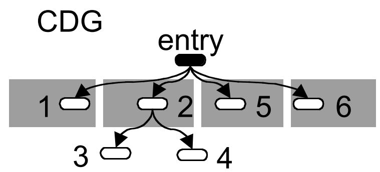
```
     Program Simple
     1: read i
X -> 2: if ( i == 1)
     3:    print "POS:"
        else
Y -> 4:    i = 1
     5: print i
     6: end
```



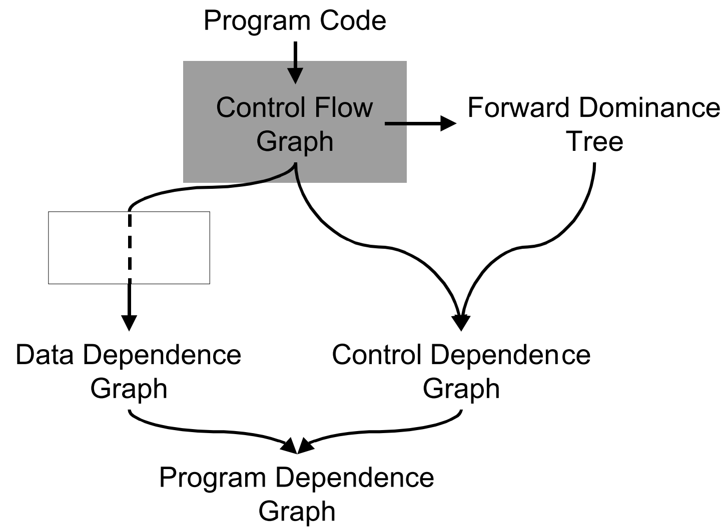
A Control Dependence Representation

- 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



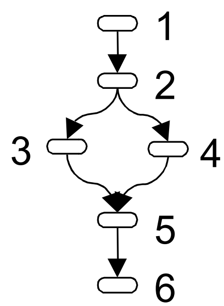
©2000 Judith A. Stafford
University of Colorado at Boulder SERL

A Graph Representation of Behavior

- The *control flow graph, "CFG"*

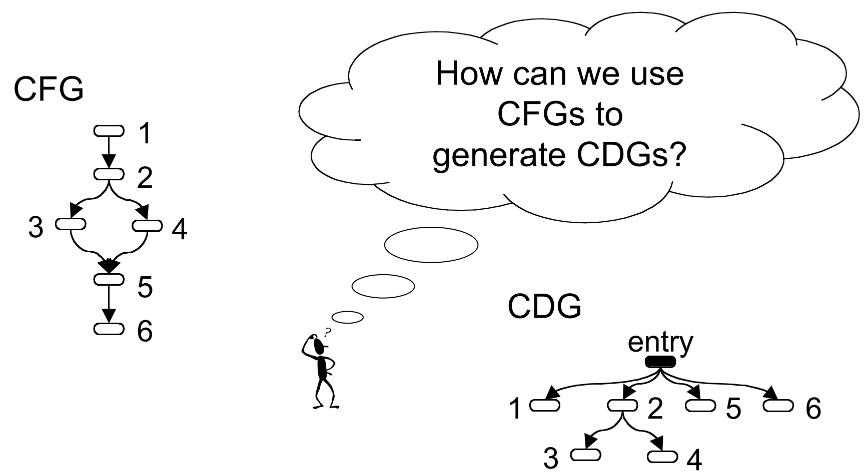
```
Program Simple
1: read i
2: if ( i == 1 )
3:    print "POS:"
   else
4:    i = 1
5: print i
6: end
```

- Vertices represent executable statements
- One entry and one exit
- Arcs represent potential control flow



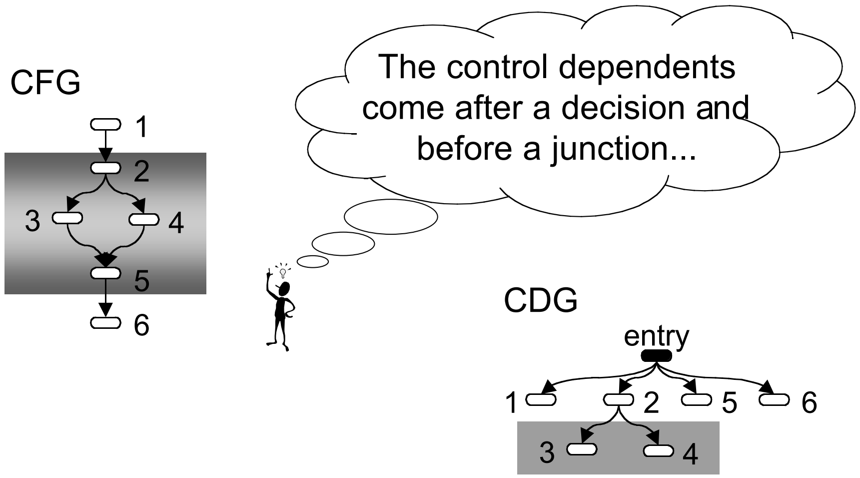
©2000 Judith A. Stafford
University of Colorado at Boulder SERL

Calculating Control Dependencies



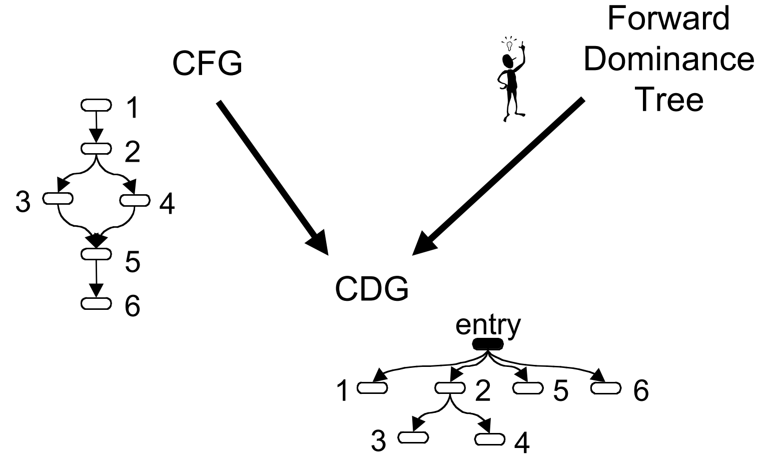
©2000 Judith A. Stafford
University of Colorado at Boulder SERL

Calculating Control Dependencies

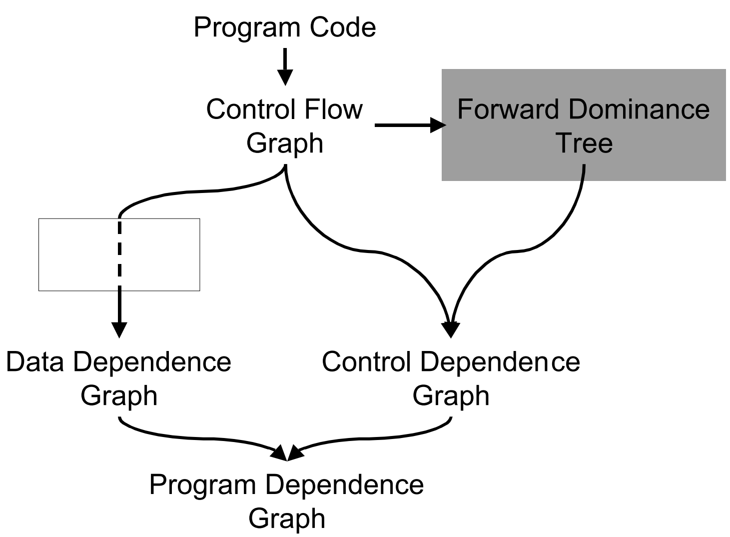


Calculating Control Dependencies

- Use the dominance tree (in reverse)!



A Graph-Based Model



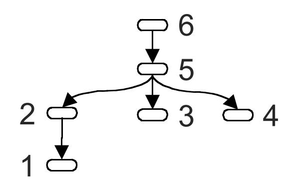
Forward Dominance (a.k.a. post dominance, inverse dominance)

- The *forward dominance tree, "FDT"*

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

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

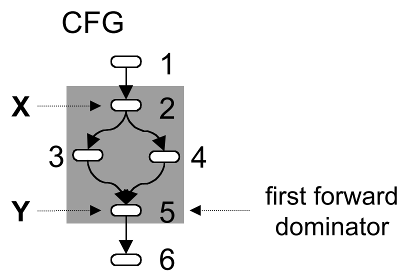
FDT



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

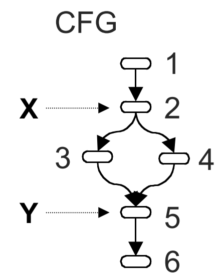


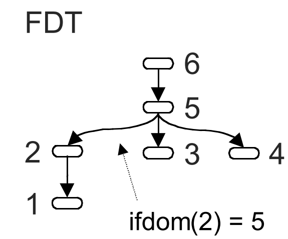
Notice that the control dependents, 3 and 4, don't forward dominate 2

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"

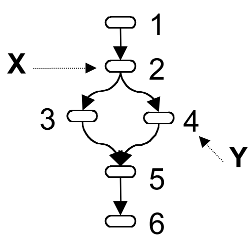
Program Simple
1: read i
2: if (i = 1)
3: print "POS:"
else
4: i = 1
5: print i
6: end

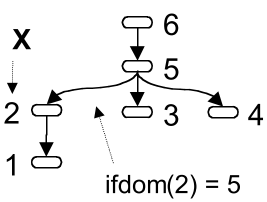


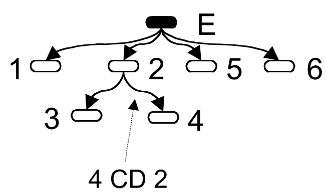


A Graph-Based Definition

- 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



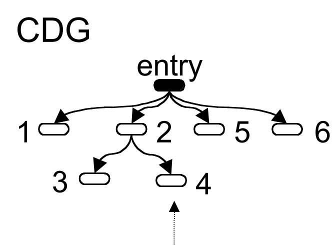




CFG + FDT = CDG

How Does This Help?

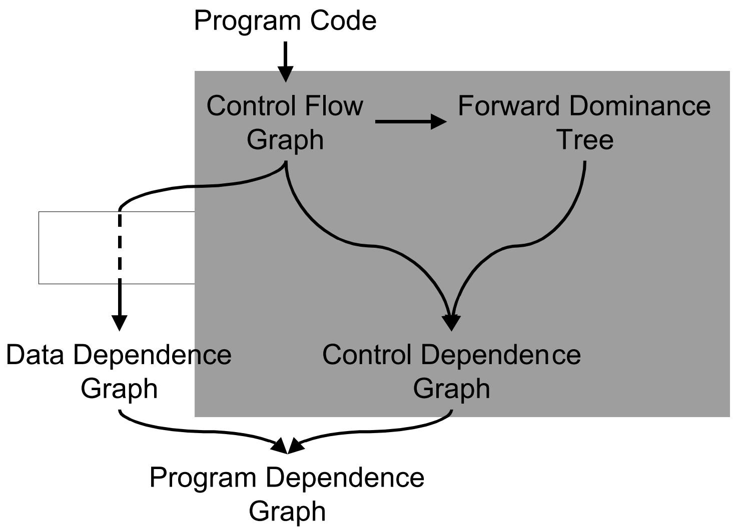
- Now we have half of the answer



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

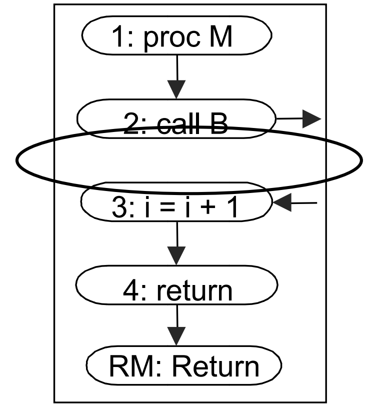
Program Simple
1: read i
2: if (i == 1)
3: print "POS:"
else
4: i = 1
5: print i
6: end

A Graph-Based Model



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

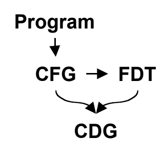


The Roadmap

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

Other Models

Uni-Procedure Model



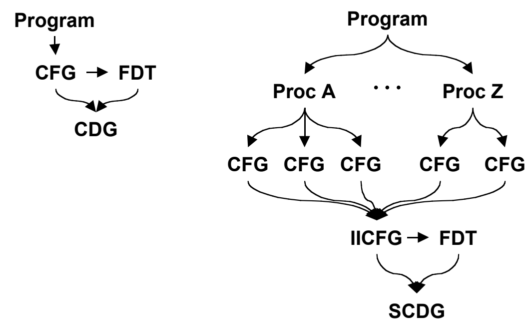
LIMITED

Multi-Procedure Approaches

Other Models

Uni-Procedure Model | **Multi-Procedure Approaches**

In-lined Approach

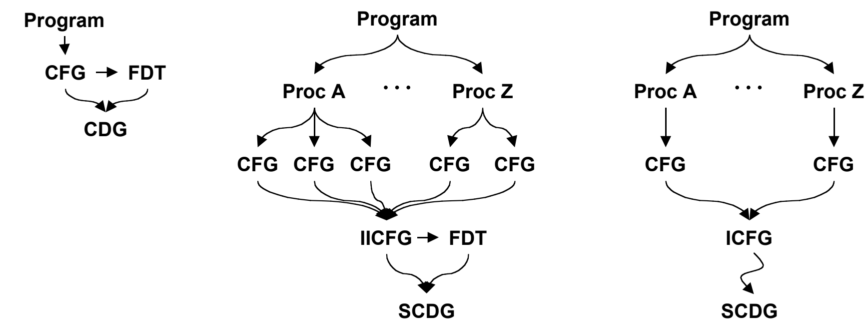


LIMITED | **IMPRACTICAL**

Other Models

Uni-Procedure Model | **Multi-Procedure Approaches**

In-lined Approach | **1-1 Graph Approach**



LIMITED | **IMPRACTICAL** | ***AD HOC***

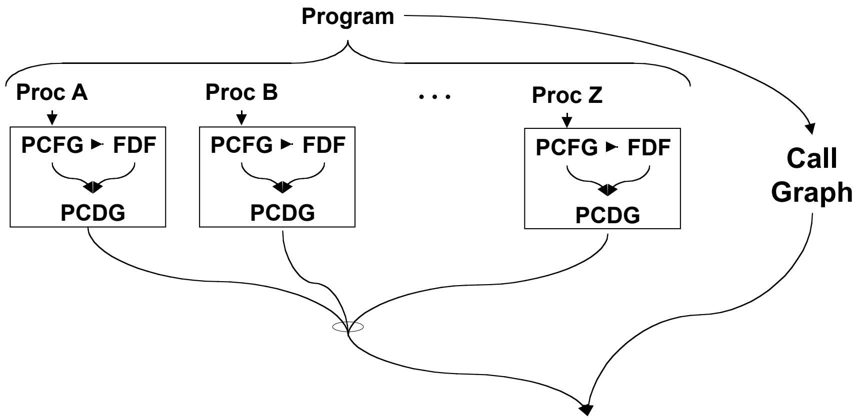
The Roadmap

- ✓ 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

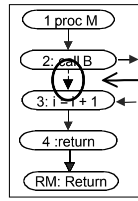
A New View - A Compositional Model



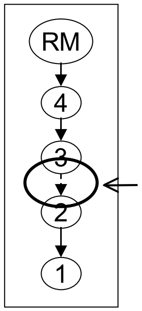
Procedure-level Structures

- Keep track of the potential indirect flows and forward dominances

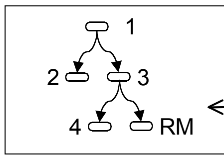
Procedure control flow graph ⇒



Forward Dominance Forest ⇒



⇓ Procedure Control Dependence Graph



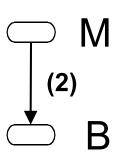
Vertices after a call depend upon return

A Graph Representation of Structure

- The *call graph*

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

- Vertices represent procedures in a program
- Arcs represent procedure call
- Arcs are annotated with ID of each call site

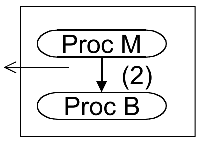


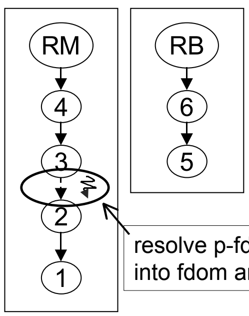
Program-level Structures

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

Program Call Graph ⇓

call arc labeled with call site vertex id



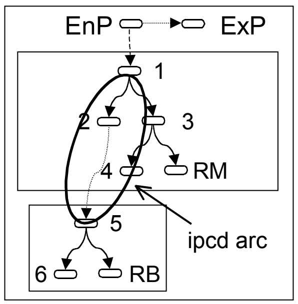


resolve p-fdom arc into fdom arc

⇐ Program Forward Dominance Forest

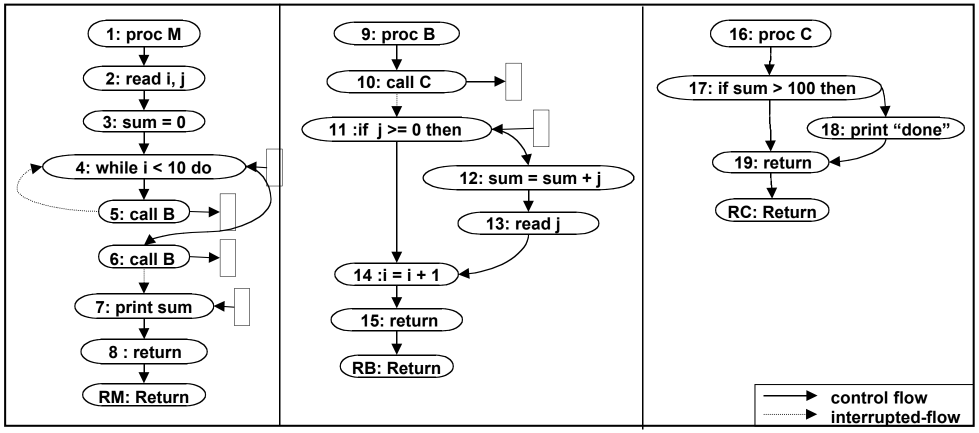
Compound Control Dependence ⇒

Procedure B and forward dominators of 2 **inherit control dependence** from 2

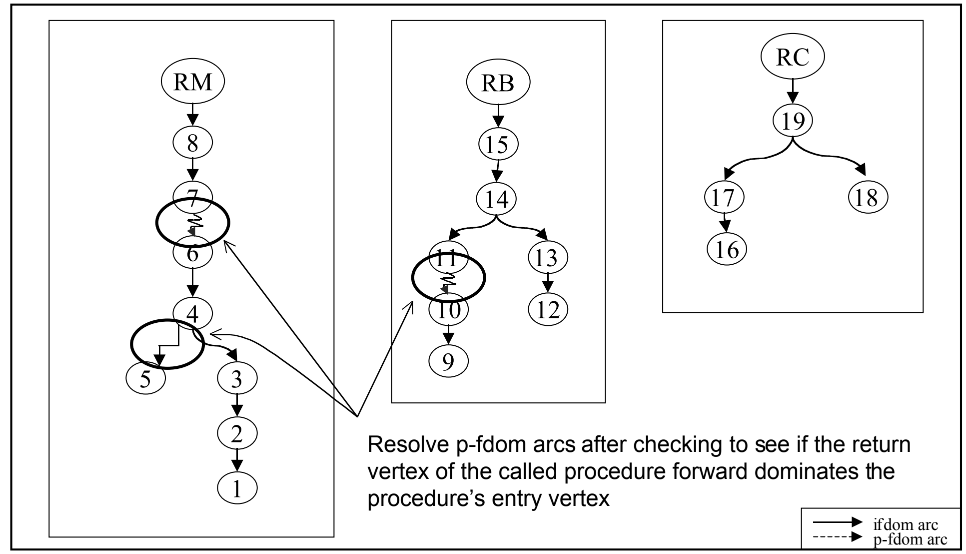


ipcd arc

Example -- Program Sum

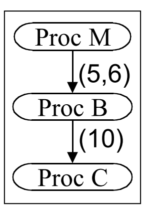


Forward Dominator Forest



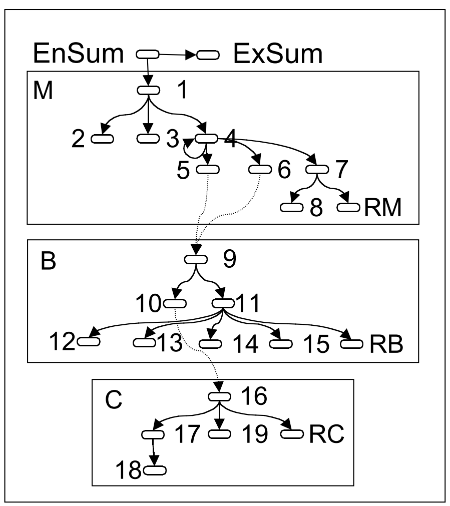
Control Dependencies in Program Sum

- Call Graph for Program Sum

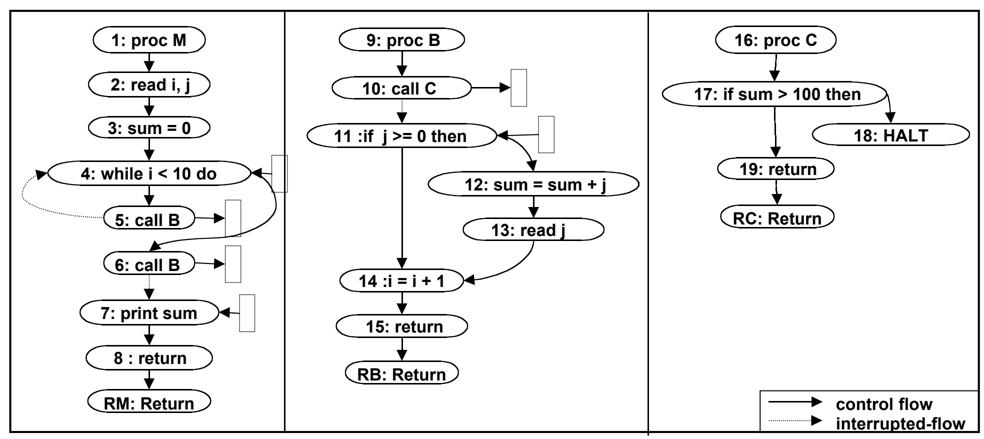


Procedure inherits control dependence of call

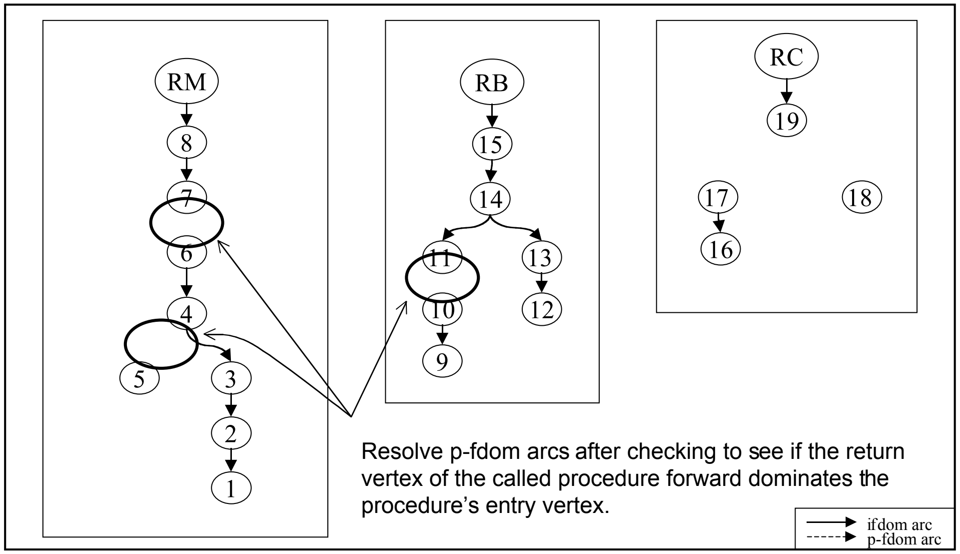
- Sum’s Compound Control Dependence Graph



Introducing an Embedded Halt

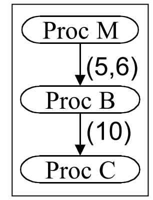


Forward Dominator Forest -- with EHalt



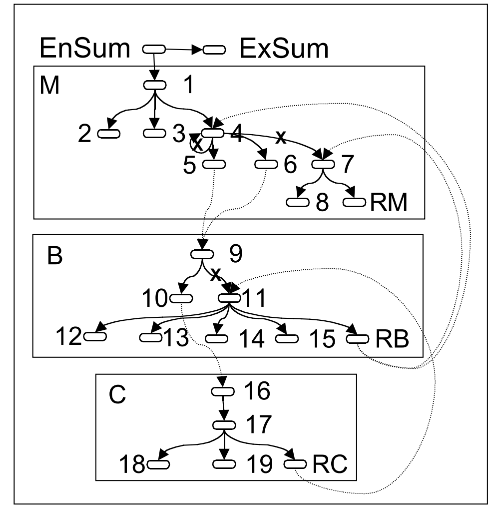
Effect of Halt on Control Dependence

- Call Graph for Program Sum



Procedure inherits control dependence of call/return sites

- Sum's Compound Control Dependence Graph



Related Work

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

CFG + FDT = CDG	Uni-procedure	Podgurski+'90 Ferrante+'87
xFG ⇝ xCDG	Multi-procedure	Horwitz+'90, Loyall+'93 Harrold+98,99,Liao+'99
xFG ⇝ xCDG	Object-oriented	Larsen+'96 Zhao+'96
xFG ⇝ xCDG	Concurrent	Zhao+'96
xFG ⇝ xCDG	Concurrent-OO	Hatcliff+'99, Zhao+'99
xFG ⇝ xCDG	Reactive	Clarke+'99, Stafford+'98