Class 10

- Review; questions
- Questions about project
- Arbitrary interprocedural control flow (cont'd)
- Pointers
- Assign (see Schedule for links)
  - Readings on symbolic execution
  - Problem Set 5: due 9/22/09
  - Project proposal
    - Initial: due by e-mail 9/22/09
    - Final: due (written, 2 pages) 9/29/09

Complicating Factors

A. Programs with more than one procedure
B. Programs with recursion
C. Programs with arbitrary control flow
D. Programs with pointers
E. Programs with complex data structures

Arbitrary Interprocedural CF

- Three ways in which intra-procedural control dependences can be inaccurate
  - Entry-dependence effect
  - Multiple-context effect
  - Return-dependence effect

Computation of Interprocedural CD

Identify potentially non-returning call sites

↓

Construct augmented control-flow graph

↓

Compute partial control dependences
Construct augmented control-dependence graph

↓

Construct interprocedural control-dependence graph
Propagate control dependences

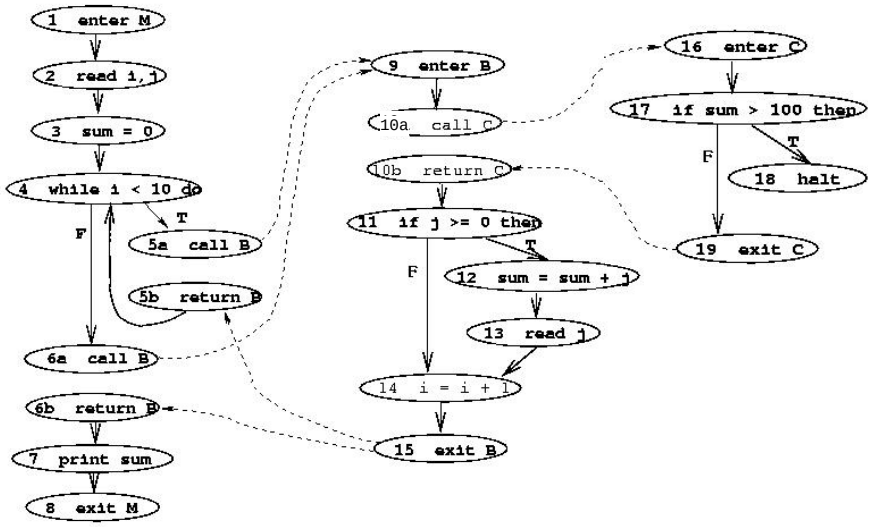
PNRC Analysis

- Step 1: Identifies three sets
  - DNRPList: Definitely non-returning procedures
  - UnreachList: Statically unreachable nodes
  - HNList: Halt statements reachable from entry
- Method
  - Build ICFG
  - Depth first traversal along realizable paths marking visited nodes
    - Unmarked nodes are unreachable
    - Unmarked exit nodes indicate DNRPs
    - Marked halt nodes indicate reachable halts

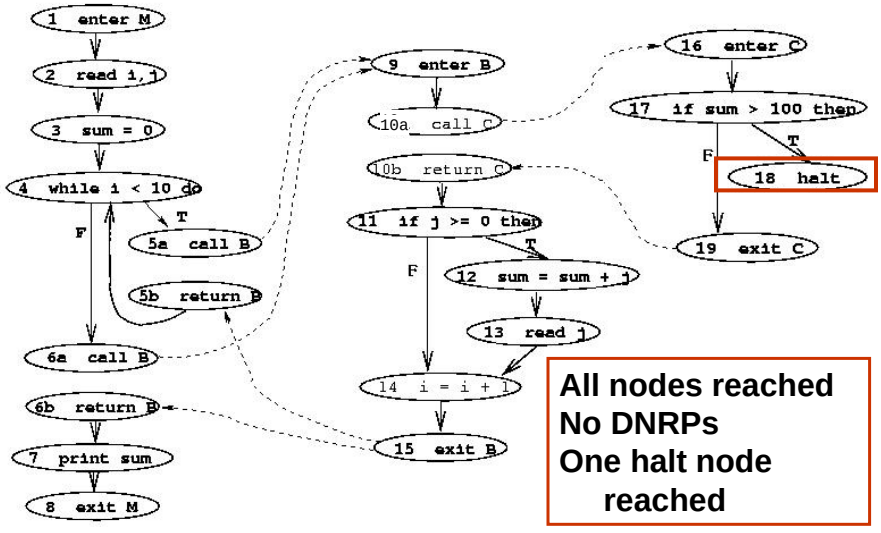
PNRC Analysis

- Step 1: Identifies three sets
  - DNRPList: Definitely non-returning procedures
  - UnreachList: Statically unreachable nodes
  - HNList: Halt statements reachable from entry
- Method
  - Build ICFG
  - Depth first traversal along realizable paths marking visited nodes
    - Unmarked nodes are unreachable
    - Unmarked exit nodes indicate DNRPs
    - Marked halt nodes indicate reachable halts

PNRC Analysis

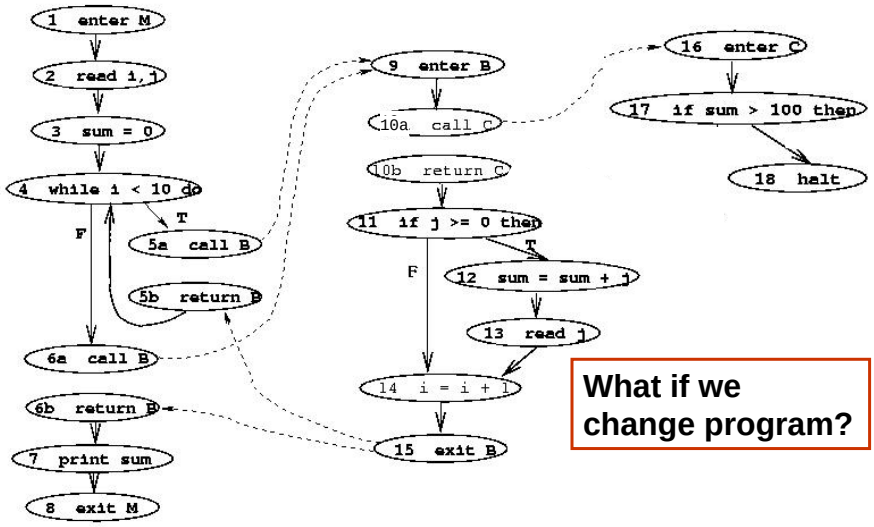


PNRC Analysis



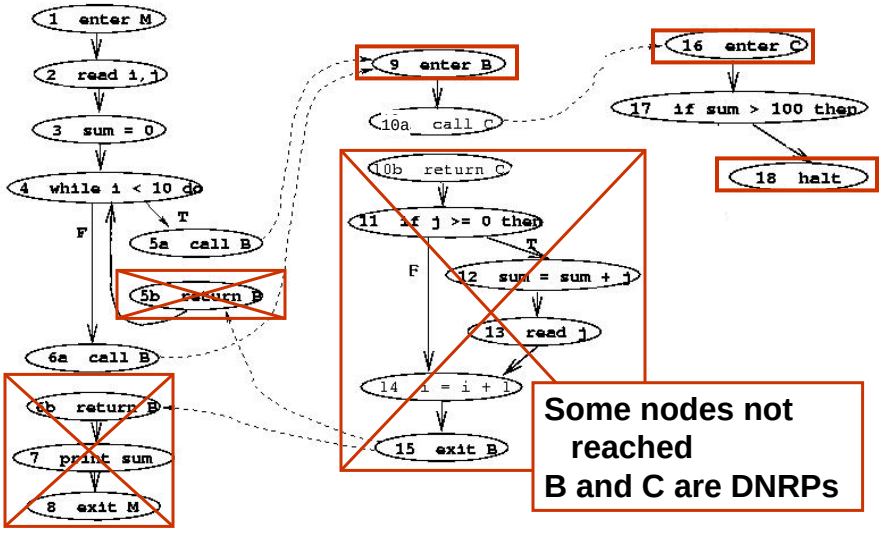
All nodes reached
No DNRPs
One halt node reached

PNRC Analysis



What if we change program?

PNRC Analysis

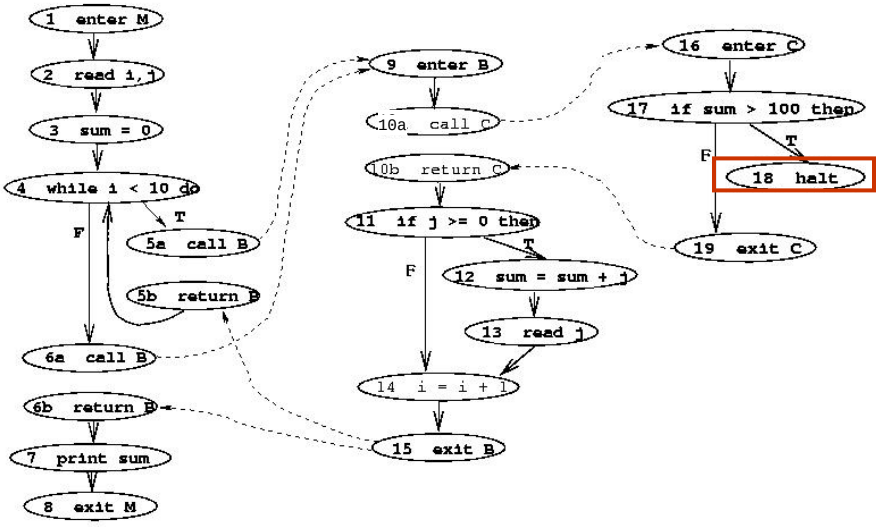


Some nodes not reached
B and C are DNRPs

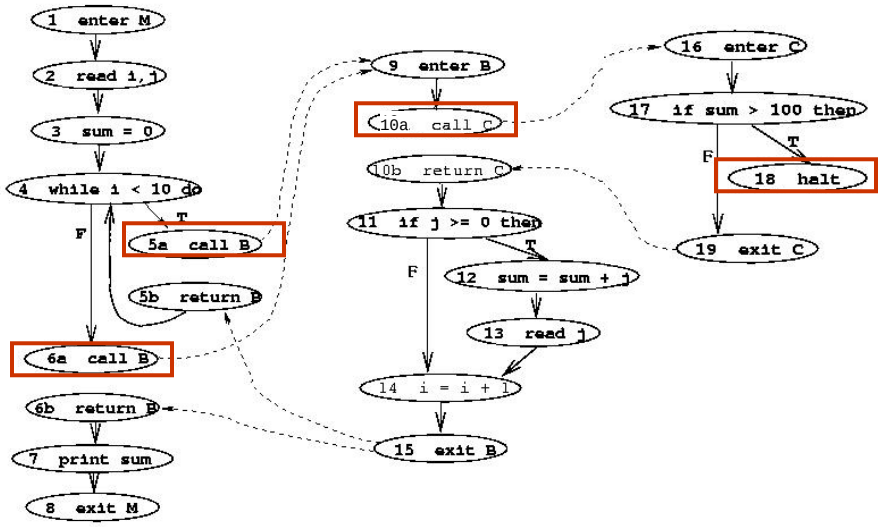
PNRC Analysis

- Step 2: Compute partial CD
  - Identify PNRCList: Possibly non-returning call-sites
  - Build ACFGs
- Method
  - Backward traversal of ICFG starting from (1) halt nodes and (2) calls to DNRPs
    - Ascending into callers, but not descending into callees (similar to SDG slicing)
  - Any call site reached is a PNRC

PNRC Analysis



PNRC Analysis



Computation of Interprocedural CD

Identify potentially non-returning call sites

↓

Construct augmented control-flow graph

↓

Compute partial control dependences
Construct augmented control-dependence graph

↓

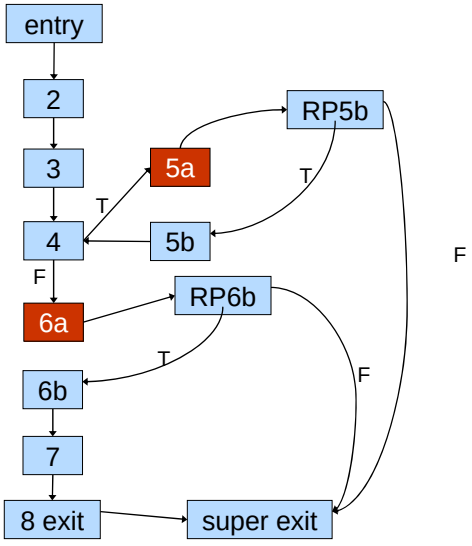
Construct interprocedural control-dependence graph
Propagate control dependences

Augmented Control-Flow Graph

For each procedure, starting from its CFG

- Create super-exit node
- For each potentially non-returning call site
  - create *return-predicate* node
  - Connect return-predicate node to potential return sites
  - Eliminate edge between call and return

Augmented Control-Flow Graph



For each procedure, starting from its CFG

- Create super-exit node
- For each potentially non-returning call site
  - Create *return-predicate* node
  - Connect return-predicate node to potential return sites
  - Eliminate edge between call and return

Computation of Interprocedural CD

Identify potentially non-returning call sites

↓

Construct augmented control-flow graph

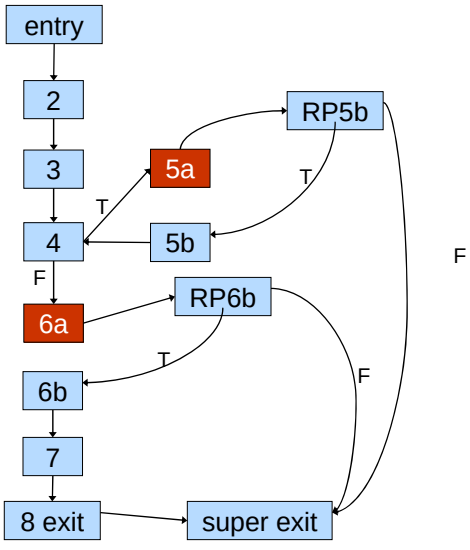
↓

Compute partial control dependences
Construct augmented control-dependence graph

↓

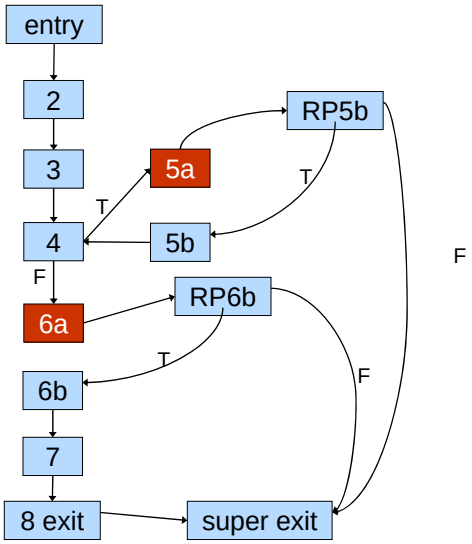
Construct interprocedural control-dependence graph
Propagate control dependences

Partial Control Dependences



	Partial CD
2,3	
4	
5a,6a	
5b	
6b,7,8	

Partial Control Dependences

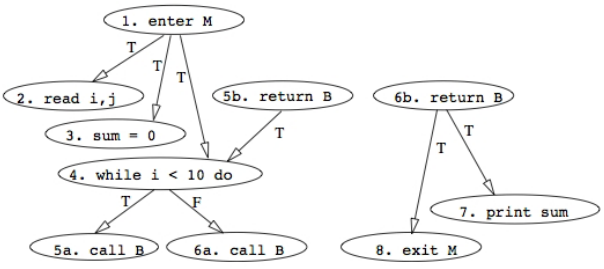


	Partial CD
2,3	entry
4	entry
	RP5b
5a,6a	4
5b	RP5b
6b,7,8	RP6b

Augmented CDG

Build ACDG

- CDG built from an ACFG
- Replace return-predicate nodes with corresponding return



	Partial CD
2,3	entry
4	entry
	RP5b
5a,6a	4
5b	RP5b
6b,7,8	RP6b

Computation of Interprocedural CD

Identify potentially non-returning call sites

↓

Construct augmented control-flow graph

↓

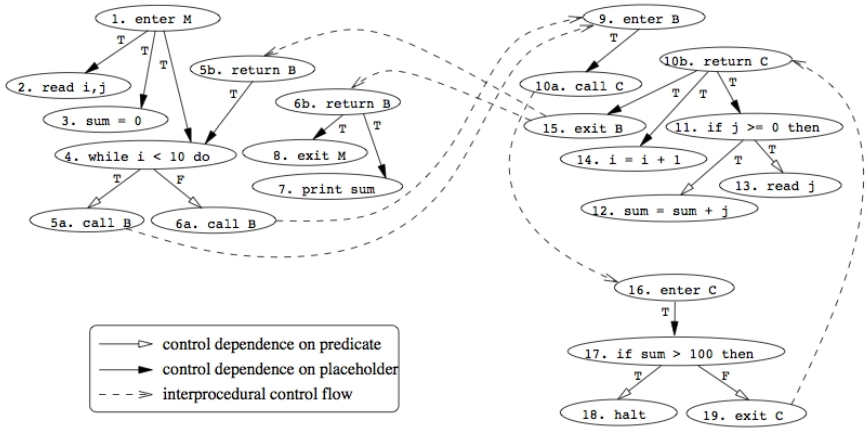
Compute partial control dependences
Construct augmented control-dependence graph

↓

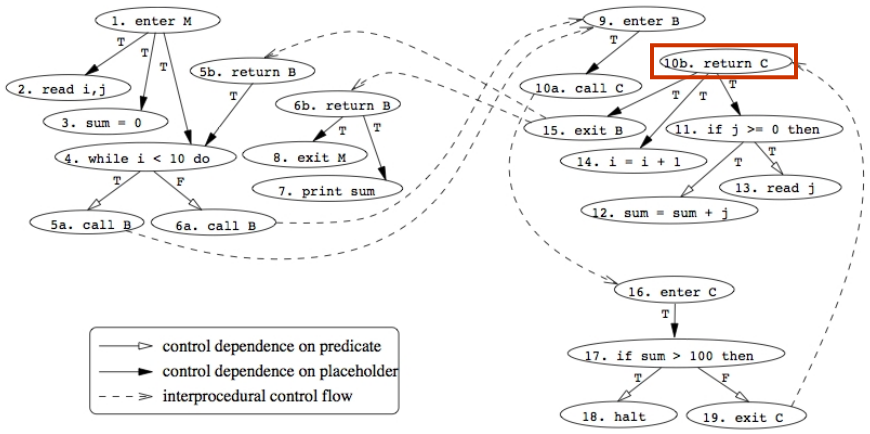
Construct interprocedural control-dependence graph
Propagate control dependences

Build ICDG

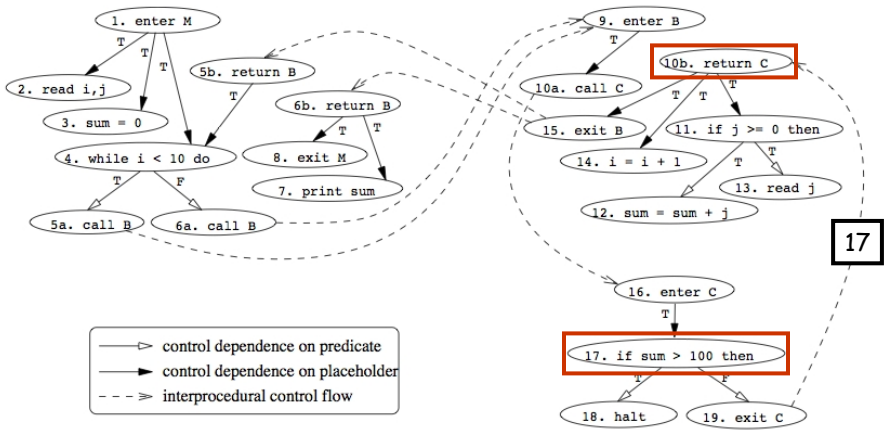
- Connect ACDGs with interprocedural control-flow edges



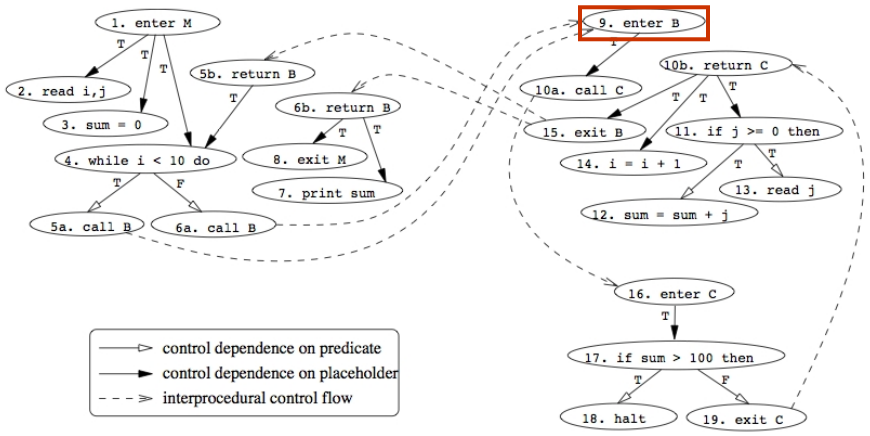
Build ICDG
- Connect ACDGs with interprocedural control-flow edges
- Replace all dependences to placeholder
  - Backward traversal from the placeholders to the first (non-placeholder) predicate node along each path
    => add control dependence



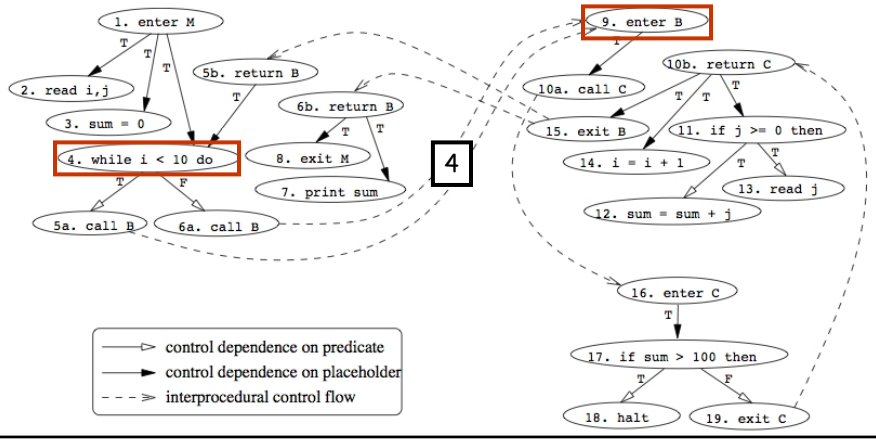
Build ICDG
- Connect ACDGs with interprocedural control-flow edges
- Replace all dependences to placeholder
  - Backward traversal from the placeholders to the first (non-placeholder) predicate node along each path
    => add control dependence



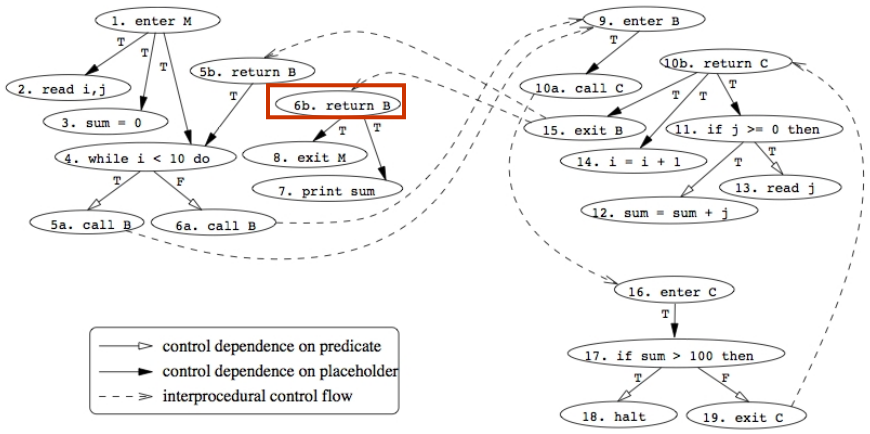
Build ICDG
- Connect ACDGs with interprocedural control-flow edges
- Replace all dependences to placeholder
  - Backward traversal from the placeholders to the first (non-placeholder) predicate node along each path
    => add control dependence



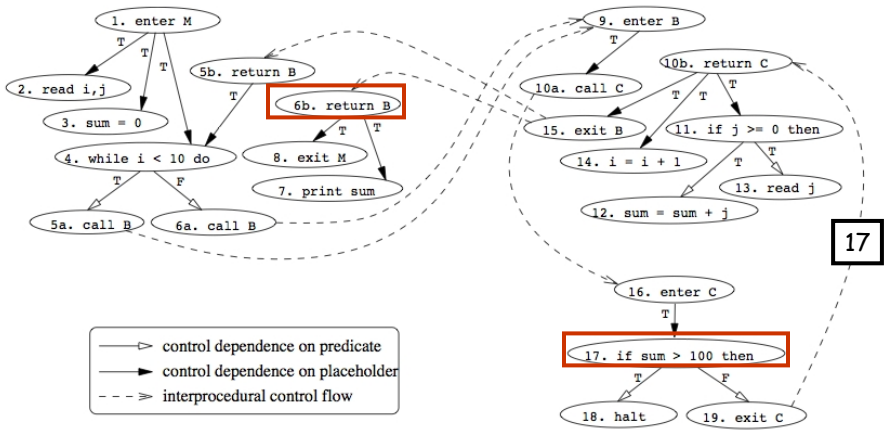
Build ICDG
- Connect ACDGs with interprocedural control-flow edges
- Replace all dependences to placeholder
  - Backward traversal from the placeholders to the first (non-placeholder) predicate node along each path
    => add control dependence



Build ICDG
- Connect ACDGs with interprocedural control-flow edges
- Replace all dependences to placeholder
  - Backward traversal from the placeholders to the first (non-placeholder) predicate node along each path
    => add control dependence

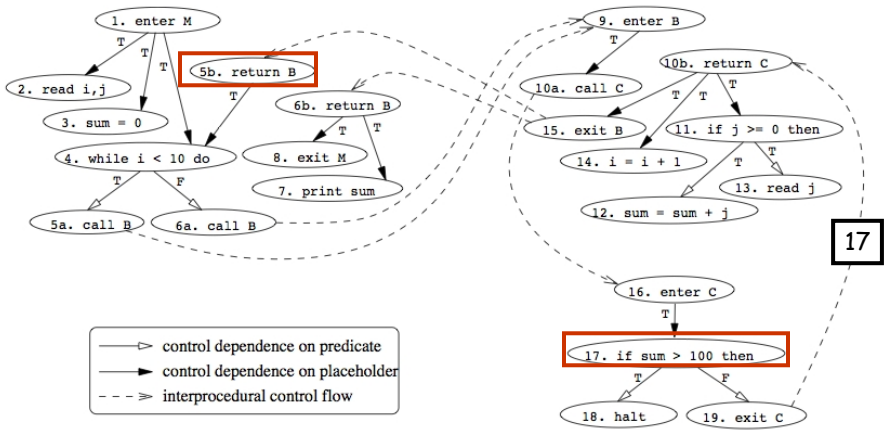


Build ICDG
- Connect ACDGs with interprocedural control-flow edges
- Replace all dependences to placeholder
  - Backward traversal from the placeholders to the first (non-placeholder) predicate node along each path
    => add control dependence



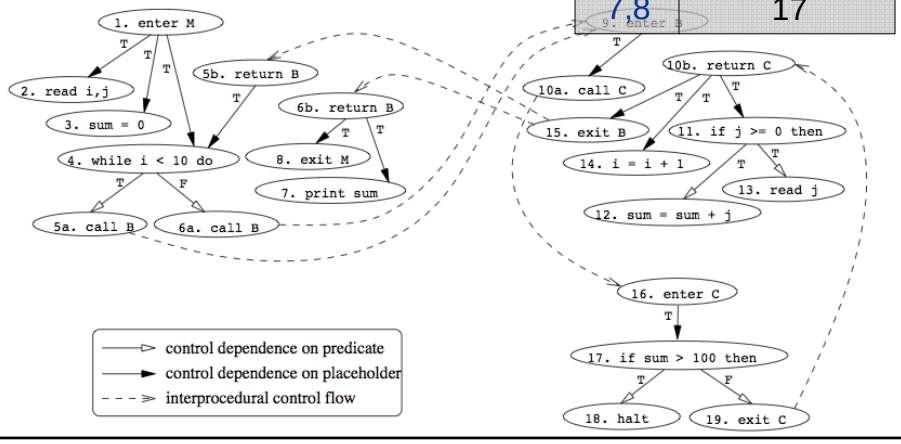
Build ICDG

- Connect ACDGs with interprocedural control-flow edges
- Replace all dependences to placeholder
  - Backward traversal from the placeholders to the first (non-placeholder) predicate node along each path
    => add control dependence



Interprocedural CDG

	Partial CD
2,3	entry
4	entry
	17
5,6	4
7,8	17



Applications of Interprocedural CD

- Computing interprocedural slices
- Identifying conditions associated with statements/procedures
- Computing control coupling
- …

Complicating Factors

- Programs with more than one procedure
- Recursion
- Programs with pointers
- Programs with complex data structures
- Programs with arbitrary control flow

Complicating Factors (pointers)

- Aliasing: different names reference the same memory location

```
1 main() {
2  int *p, x, y;
3  x = 0;
4  p = &x;
5  *p = *p+1;
6  y = x;
7 }
```

*p is an alias for x
=> x = x+1;

- Alias information conveniently represented with points-to sets (e.g., *p -> {x})
- Typically, MAY information

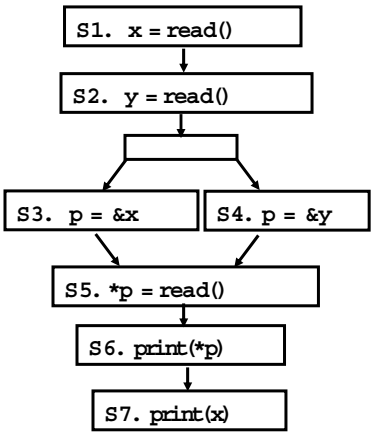
Complicating Factors (pointers)

Pointers complicate data-flow

Consider an example

What is Def(S5)?

Can we simply "plug-in" alias information?



Complicating Factors (pointers)

Extending def-use concepts

- DDEF: Definite Definition
- PDEF: Possible Definition
- DUSE: Definite Use
- PUSE: Possible Use

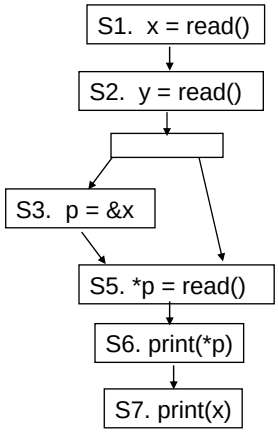
Extending algorithms

- Both possible and definite info in GEN
- Only definite info in KILL

Complicating Factors (pointers)

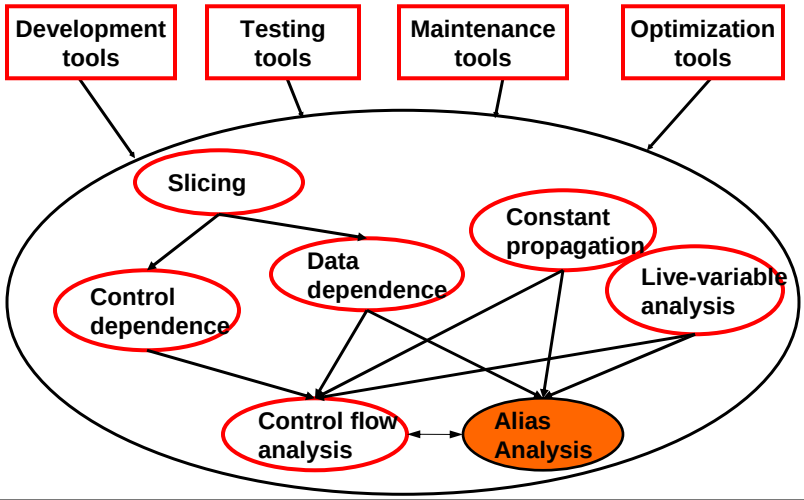
Pointers complicate
data-flow
Consider an example

Are we in better shape in this case?
(p* -> {x})



Pointer/Alias Analysis

- Goal: determine memory locations accessed through pointer dereferences Importance:



Alias Analysis (AA)

Must alias information indicates that the alias occurs on all paths in the CFG

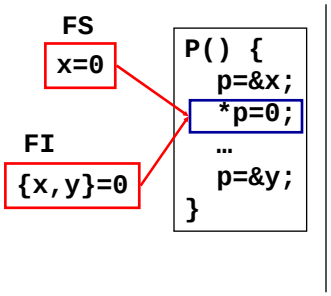
May alias information indicates that the alias occurs on some path in the CFG

Flow-sensitive (flow-insensitive) aliasing information depends (does not depend) on the control flow in a procedure

Context-sensitive (context-insensitive) aliasing information obeys (does not obey) the calling context when propagating

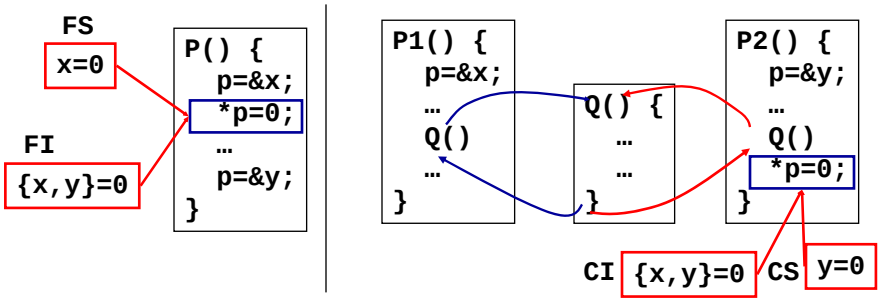
Introduction, Motivation

- Precise alias analysis is undecidable
- Approximation algorithms
  - Flow-sensitive (FS) vs flow-insensitive (FI)
  - Context-sensitive (CS) vs context-insensitive (CI)



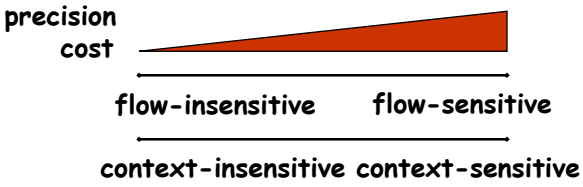
Introduction, Motivation

- Precise alias analysis is undecidable
- Approximation algorithms
  - Flow-sensitive (FS) vs flow-insensitive (FI)
  - Context-sensitive (CS) vs context-insensitive (CI)



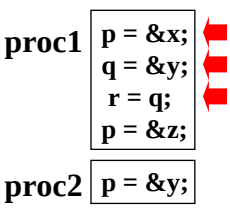
Introduction, Motivation

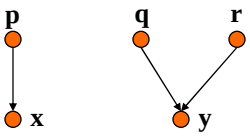
- Precise alias analysis is undecidable
- Approximation algorithms
  - Flow-sensitive (FS) vs flow-insensitive (FI)
  - Context-sensitive (CS) vs context-insensitive (CI)



Existing Approaches

Steendgaard's

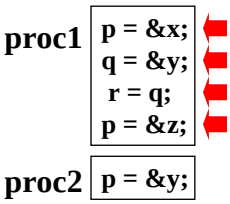


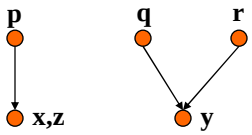


Program-specific points-to graph

Existing Approaches

Steendgaard's

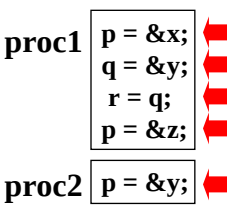


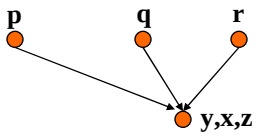


Program-specific points-to graph

Existing Approaches

Steendgaard's





Program-specific points-to graph

Existing Approaches

Landi and Ryder's

proc1

```
p = &x;
q = &y;
r = q;
p = &z;
```

proc2

```
p = &y;
```

Point-specific points-to graph

Existing Approaches

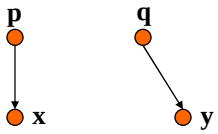
Landi and Ryder's

proc1

```
p = &x;
q = &y;
r = q;
p = &z;
```

proc2

```
p = &y;
```



Point-specific points-to graph

Existing Approaches

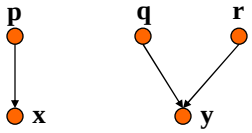
Landi and Ryder's

proc1

```
p = &x;
q = &y;
r = q;
p = &z;
```

proc2

```
p = &y;
```



Point-specific points-to graph

Existing Approaches

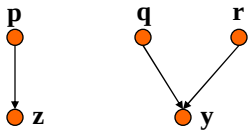
Landi and Ryder's

proc1

```
p = &x;
q = &y;
r = q;
p = &z;
```

proc2

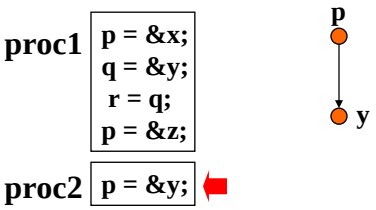
```
p = &y;
```



Point-specific points-to graph

Existing Approaches

Landi and Ryder's



Point-specific points-to graph

Program Analysis w/ Pointers

- Step 1: Perform alias analysis
- Step 2: Resolve pointer dereferences
- Step 3: Perform whole-program analysis