

Control Flow Analysis

COMP 621 – Program Analysis and Transformations

*These slides have been adapted from
<http://cs.gmu.edu/~white/CS640/Slides/CS640-2-02.ppt>
by Professor Liz White.*

How to represent the structure of the program?

- Based on the compositional structure ...
i.e. the AST ...

- As a graph – which we discover from a
sequential representation of low-level IR
statements.

Program Control Flow

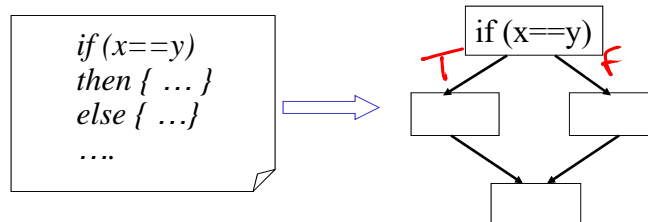
- Control flow
 - Sequence of operations
 - Representations
 - Control flow graph
 - Control dependence
 - Call graph
- Control flow analysis
 - Analyzing program to discover its control structure
 - Today's topic: CFG-based analysis

Control Flow Analysis

3

Control Flow Graph

- CFG models flow of control in the program (procedure)
- $G = (N, E)$ as a directed graph
 - Node $n \in N$: basic blocks
 - A basic block is a maximal sequence of stmts with a single entry point, single exit point, and no internal branches
 - For simplicity, we assume a unique entry node n_0 and a unique exit node n_f in later discussions
 - Edge $e = (n_i, n_j) \in E$: possible transfer of control from block n_i to block n_j



Control Flow Analysis

4

Basic Blocks

- Definition
 - A basic block is a maximal sequence of consecutive statements with a single entry point, a single exit point, and no internal branches
- Basic unit in control flow analysis

Control Flow Analysis

5

Basic Blocks

- Local level of code optimizations
 - Redundancy elimination
 - Register-allocation
- Easy to do flow analysis because there are no alternative control flow paths.

$x = 2$

$y = x + 1$

$z = z + 1$

Control Flow Analysis

6

Basic Block Example

```
(1) i := m - 1
(2) j := n
(3) t1 := 4 * n
(4) v := a[t1]
(5) i := i + 1
(6) t2 := 4 * i
(7) t3 := a[t2]
(8) if t3 < v goto (5)
(9) j := j - 1
(10) t4 := 4 * j
(11) t5 := a[t4]
(12) if t5 > v goto (9)
(13) if i >= j goto (23)
(14) t6 := 4 * i
(15) x := a[t6]
...
```

(23) =

- How many basic blocks in this code fragment?
- What are they?

Control Flow Analysis

7

Basic Block Example

```
(1) i := m - 1
(2) j := n
(3) t1 := 4 * n
(4) v := a[t1]
(5) i := i + 1
(6) t2 := 4 * i
(7) t3 := a[t2]
(8) if t3 < v goto (5)
(9) j := j - 1
(10) t4 := 4 * j
(11) t5 := a[t4]
(12) if t5 > v goto (9)
(13) if i >= j goto (23)
(14) t6 := 4 * i
(15) x := a[t6]
...
```

- How many basic blocks in this code fragment?
- What are they?

Control Flow Analysis

8

Identify Basic Blocks

Input: A sequence of intermediate code statements

1. Determine the *leaders*, the first statements of basic blocks
 - The first statement in the sequence (entry point) is a leader
 - Any statement that is the target of a branch (conditional or unconditional) is a leader
 - Any statement immediately following a branch (conditional or unconditional) or a return is a leader
2. For each leader, its basic block is the leader and all statements up to, but not including, the next leader or the end of the program

Control Flow Analysis

9

Example

```
(1) i := m - 1
(2) j := n
(3) t1 := 4 * n
(4) v := a[t1]
(5) i := i + 1
(6) t2 := 4 * i
(7) t3 := a[t2]
(8) if t3 < v goto (5)
(9) j := j - 1
(10) t4 := 4 * j
(11) t5 := a[t4]
(12) if t5 > v goto (9)
(13) if i >= j goto (23)
(14) t6 := 4 * i
(15) x := a[t6]
(16) t7 := 4 * i
(17) t8 := 4 * j
(18) t9 := a[t8]
(19) a[t7] := t9
(20) t10 := 4 * j
(21) a[t10] := x
(22) goto (5)
(23) t11 := 4 * i
(24) x := a[t11]
(25) t12 := 4 * i
(26) t13 := 4 * n
(27) t14 := a[t13]
(28) a[t12] := t14
(29) t15 := 4 * n
(30) a[t15] := x
```

Control Flow Analysis

10

Example: Leaders

| | |
|--|---------------------------------------|
| (1) $i := m - 1$ | (16) $t7 := 4 * i$ |
| (2) $j := n$ | (17) $t8 := 4 * j$ |
| (3) $t1 := 4 * n$ | (18) $t9 := a[t8]$ |
| (4) $v := a[t1]$ | (19) $a[t7] := t9$ |
| (5) $i := i + 1$ | (20) $t10 := 4 * j$ |
| (6) $t2 := 4 * i$ | (21) $a[t10] := x$ |
| (7) $t3 := a[t2]$ | (22) goto (5) |
| (8) if $t3 < v$ goto (5) | (23) $t11 := 4 * i$ |
| (9) $j := j - 1$ | (24) $x := a[t11]$ |
| (10) $t4 := 4 * j$ | (25) $t12 := 4 * i$ |
| (11) $t5 := a[t4]$ | (26) $t13 := 4 * n$ |
| (12) If $t5 > v$ goto (9) | (27) $t14 := a[t13]$ |
| (13) if $i \geq j$ goto (23) | (28) $a[t12] := t14$ |
| (14) $t6 := 4 * i$ | (29) $t15 := 4 * n$ |
| (15) $x := a[t6]$ | (30) $a[t15] := x$ |

Control Flow Analysis

11

Example: Basic Blocks

| | |
|--|---------------------------------------|
| (1) $i := m - 1$ | (16) $t7 := 4 * i$ |
| (2) $j := n$ | (17) $t8 := 4 * j$ |
| (3) $t1 := 4 * n$ | (18) $t9 := a[t8]$ |
| (4) $v := a[t1]$ | (19) $a[t7] := t9$ |
| (5) $i := i + 1$ | (20) $t10 := 4 * j$ |
| (6) $t2 := 4 * i$ | (21) $a[t10] := x$ |
| (7) $t3 := a[t2]$ | (22) goto (5) |
| (8) if $t3 < v$ goto (5) | (23) $t11 := 4 * i$ |
| (9) $j := j - 1$ | (24) $x := a[t11]$ |
| (10) $t4 := 4 * j$ | (25) $t12 := 4 * i$ |
| (11) $t5 := a[t4]$ | (26) $t13 := 4 * n$ |
| (12) If $t5 > v$ goto (9) | (27) $t14 := a[t13]$ |
| (13) if $i \geq j$ goto (23) | (28) $a[t12] := t14$ |
| (14) $t6 := 4 * i$ | (29) $t15 := 4 * n$ |
| (15) $x := a[t6]$ | (30) $a[t15] := x$ |

Control Flow Analysis

12

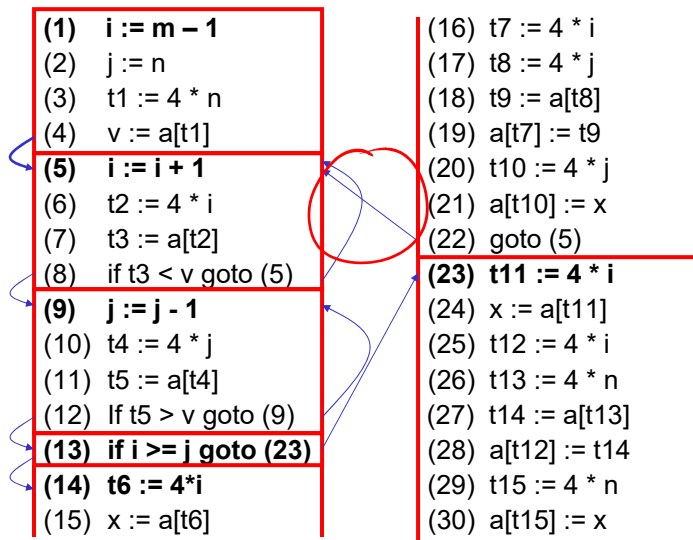
Generating CFGs

- Partition intermediate code into basic blocks
- Add edges corresponding to control flows between blocks
 - Unconditional goto
 - Conditional branch – multiple edges
 - Sequential flow – control passes to the next block (if no branch at the end)
- If no unique entry node n_0 or exit node n_f , add dummy nodes and insert necessary edges
 - Ideally no edges entering n_0 ; no edges exiting n_f
 - Simplify many analysis and transformation algorithms

Control Flow Analysis

13

Example: CFG

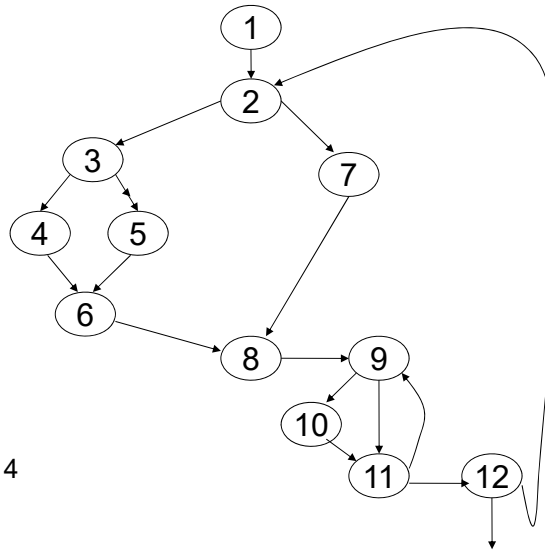


Control Flow Analysis

14

CFG and HL code

```
I = 1
J = 1
K = 1
L = 1
repeat
  if (P) then begin
    J = I
    if (Q) then L = 2
    else L = 3
    K = K + 1
  end
  else K = K + 2
  print (I,J,K,L)
  repeat
    if (R) then L = L + 4
  until (S)
  I = I + 6
until (T)
```



Control Flow Analysis

15

Complications in CFG Construction

- Function calls
 - Instruction scheduling may prefer function calls as basic block boundaries
 - Special functions as `setjmp()` and `longjmp()`
- Exception handling
- Ambiguous jump
 - **Jump r1** //target stored in register r1
 - Static analysis may generate edges that never occur at runtime
 - Record potential targets if possible
- Jumps target outside the current procedure
 - PASCAL, Algol: still restricted to lexically enclosing procedure

Control Flow Analysis

16

Nodes in CFG

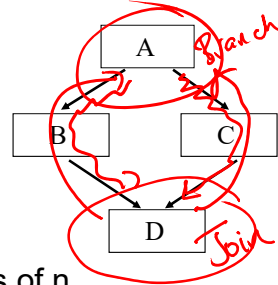
□ Given a CFG = $\langle N, E \rangle$

○ If there is an edge $n_i \rightarrow n_j \in E$

- n_i is a **predecessor** of n_j
- n_j is a **successor** of n_i

○ For any node $n \in N$

- **Pred(n)**: the set of predecessors of n
- **Succ(n)**: the set of successors of n
- A **branch node** is a node that has more than one successor
- A **join node** is a node that has more than one predecessor



Control Flow Analysis

17

Depth First Traversal

□ CFG is a rooted, directed graph

- Entry node as the root

□ Depth-first traversal (depth-first searching)

- Idea: start at the root and explore as far/deep as possible along each branch before backtracking
- Can build a spanning tree for the graph

□ Spanning tree of a directed graph G contains all nodes of G such that

- There is a path from the root to any node reachable in the original graph and
- There are no cycles

Control Flow Analysis

18

DFS Spanning Tree Algorithm

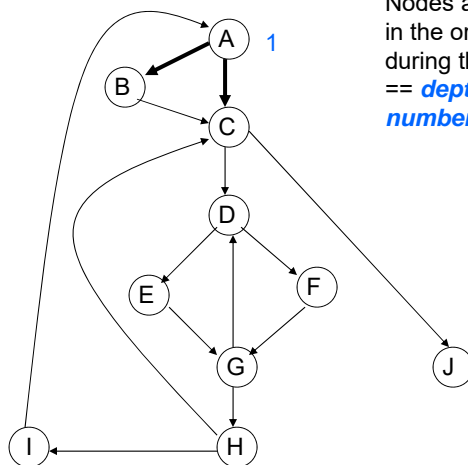
```
procedure span(v) /* v is a node in the
graph */
  InTree(v) = true
  For each w that is a successor of v do
    if (!InTree(w)) then
      Add edge v → w to spanning tree
      span(w)
  end span
```

□ Initial: $\text{span}(n_0)$

Control Flow Analysis

19

DFST Example

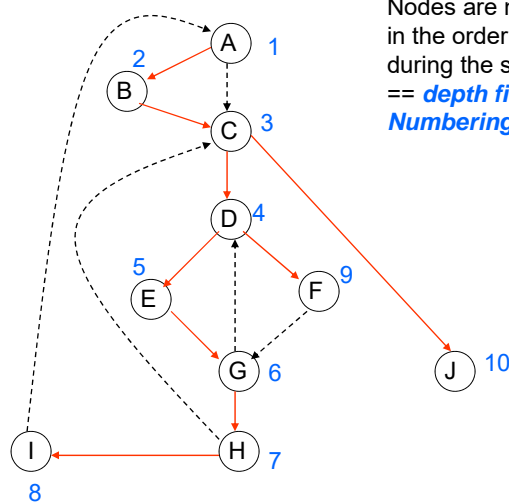


Nodes are numbered
in the order visited
during the search
== *depth first pre-order
numbering.*

Control Flow Analysis

20

DFST Example



Nodes are numbered
in the order visited
during the search
== *depth first pre-order
Numbering.*

Control Flow Analysis

21

CFG Edges Classification

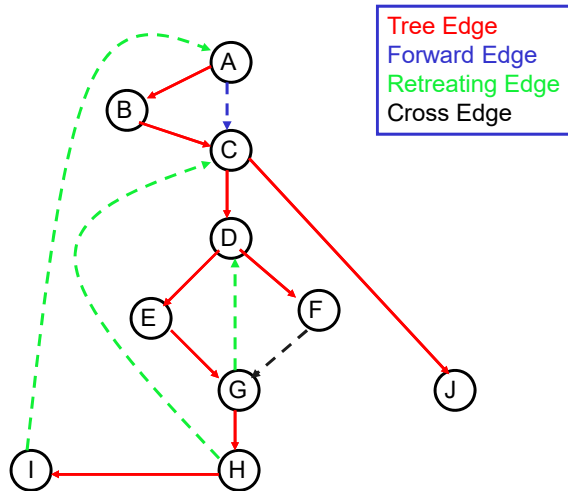
Edge $x \rightarrow y$ in a CFG is an

- *Advancing edge* – if x is an ancestor of y in the tree
 - *Tree edge* – if part of the spanning tree
 - *Forward edge* – if not part of the spanning tree and x is an ancestor of y in the tree
- *Retreating edge* – if not part of the spanning tree and y is an ancestor of x in the tree
- *Cross edge* – if not part of the spanning tree and neither is an ancestor of the other

Control Flow Analysis

22

DFST Example



Control Flow Analysis

23

Nodes Ordering wrt DFST

- Enhanced depth-first spanning tree algorithm:

```
time = 0;
procedure span(v) /* v is a node in the graph */
  InTree(v) = true; d[v] = ++time;
  For each w that is a successor of v do
    if (!InTree(w)) then
      Add edge v → w to spanning tree
      span(w)
  f[v] = ++time;
end span
```

- Associate two numbers to each node v in the graph
 - d[v]: discovery time of v in the spanning
 - f[v]: finish time of v in the spanning

Control Flow Analysis

24

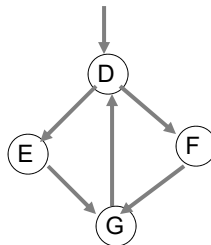
Nodes Ordering wrt DFST

- Pre-ordering
 - Ordering of vertices based on discovery time
- Post-ordering
 - Ordering of vertices based on finish time
- Reverse post-ordering
 - The reverse of a post-ordering, i.e. ordering of vertices in the opposite order of their finish time
 - Not the same as pre-ordering
 - Commonly used in forward data flow analysis
 - Backward data flow analysis: RPO on the reverse CFG

Control Flow Analysis

25

Ordering Example



- Pre-ordering: DEGF
- Post-ordering: GEFD
- Reverse post-ordering: DFEG

Control Flow Analysis

26

Big Picture

Why care about ordering / back edges?

- CFGs are commonly used to propagate information between nodes (basic blocks)
 - Data flow analysis
- The existence of back edges / cycles in flow graphs indicates that we may need to traverse the graph more than once
 - Iterative algorithms: when to stop? How quickly can we stop?
- Proper ordering of nodes during iterative algorithm assures number of passes limited by the number of “nested” back edges

Control Flow Analysis

27

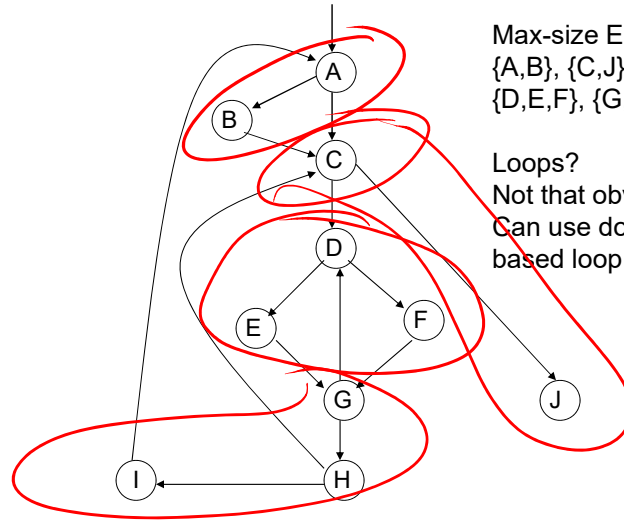
Regions in CFG – Bigger Blocks?

- Extended basic block (EBB)
 - EBB is a maximal set of nodes in a CFG that contains no join nodes other than the entry node
 - A single entry and possibly multiple exits
 - Some optimizations like value numbering and instruction scheduling are more effective if applied in EBBs

Control Flow Analysis

28

EBB Example



Max-size EBBs:
{A,B}, {C,J},
{D,E,F}, {G,H,I}

Loops?
Not that obvious...
Can use dominator-
based loop detection

Control Flow Analysis

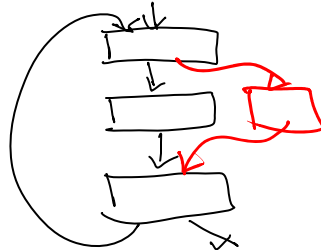
29

Loops

- explicit loops (AST)

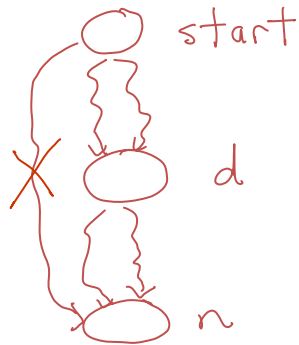
```
while (<expr>)  
<body>
```

- implicit loops in CFG



How to find loops

- need dominators



- a node d dominates a node n if every path of directed edges from start to n must go through d

- every node dominates itself

Immediate Dominators

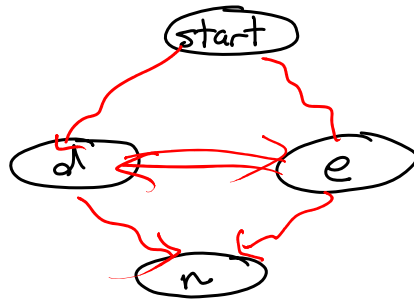
Every node n has no more than one immediate dominator, $\text{idom}(n)$, such that:

1. $\text{idom}(n)$ is not the same as n
2. $\text{idom}(n)$ dominates n
3. $\text{idom}(n)$ does not dominate any other dominator of n .



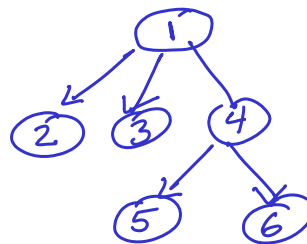
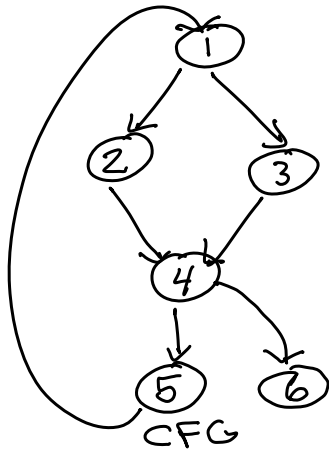
At most 1 immediate dominator

$$\begin{array}{ccc} d \text{ dominates } n & & d \text{ dominates } e \\ \neq & \Rightarrow & \text{or} \\ e \text{ dominates } n & & e \text{ dominates } d \end{array}$$



Dominator Tree

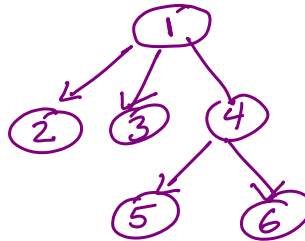
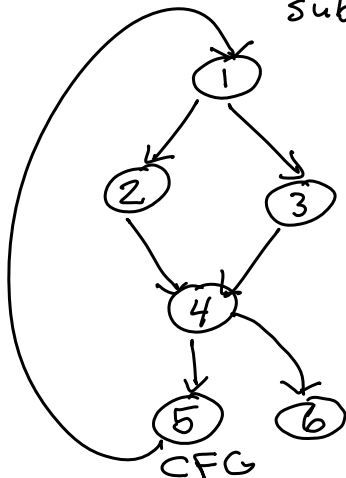
- node for each node in CFG
- edges for idom relationships



Dom Tree

Dominators & Loops

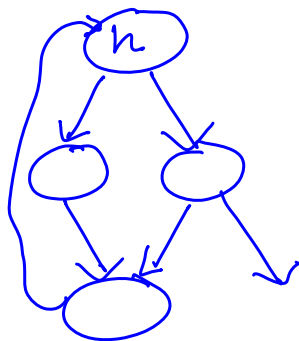
- cfg edge from n to h st h dominates n is a back edge
- for each back edge there is a sub-graph that is a loop.



Dom Tree

Finding the loop subgraph

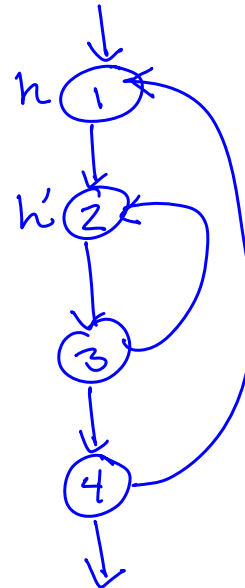
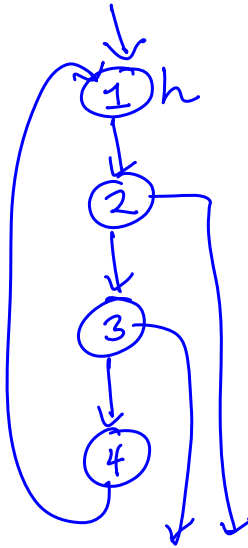
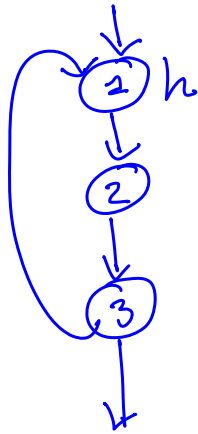
- a loop in a cfg is a set of nodes S , including a header node h with the following properties:



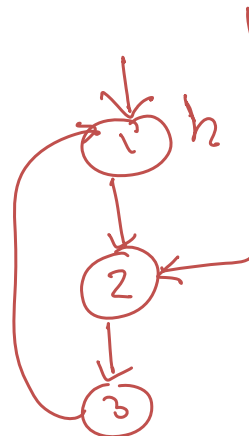
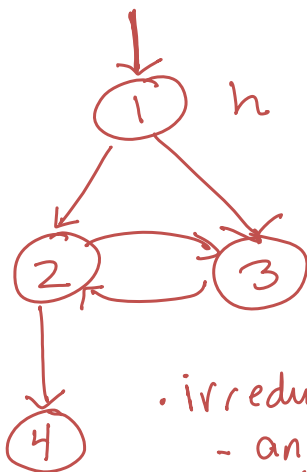
- from any node in S there is a path to h
- there is a path from h to any node in S

- there is no edge from any node outside S to any node inside S , except for h .

Examples

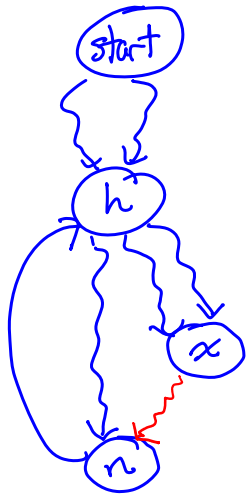


Loops?



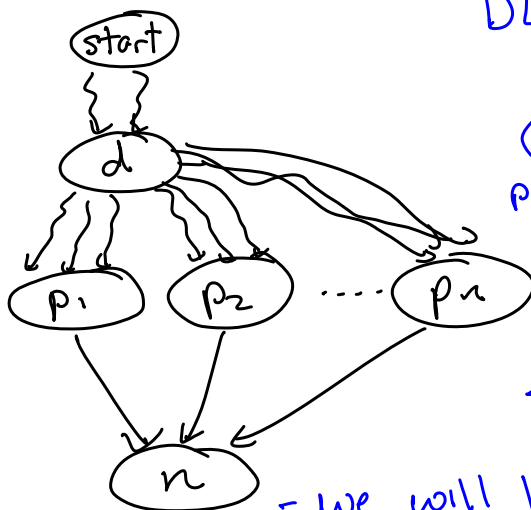
• irreducible
 - any cfg with this pattern
 or one that can collapse
 to this pattern.

Natural Loop



- edge $n \rightarrow h$
- h dominates n
- all x st
 $h \text{ dom } x \neq$
 $\exists \text{ path } x \rightsquigarrow n$
 which does not
 contain h .

Computing Dominators



$$D[n] = \{n\} \cup \left(\bigcap_{p \in \text{pred}(n)} D[p] \right)$$

• solve this as a fixed point.

• we will learn this next

Algorithm: Computing DOM

- An iterative fixed-point calculation

N is the set of nodes in the CFG

$DOM(n_0) = \{n_0\}$ (n_0 is the entry)

For all nodes $x \neq n_0$

$DOM(x) = N$

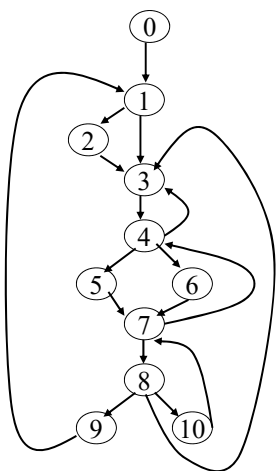
Until no more changes to dominator sets

for all nodes $x \neq n_0$

$DOM(x) = \{x\} + (\cap DOM(P))$ for all predecessors P of x

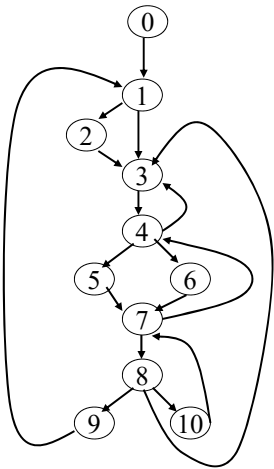
- At termination, node d in $DOM(n)$ iff d dominates n

Dominator Example



| | initial | iteration1 |
|----|---------|---|
| 0 | {0} | {0} |
| 1 | N | {1} + (Dom(0) ∩ Dom(9)) = {0,1} |
| 2 | N | {2} + Dom(1) = {0,1,2} |
| 3 | N | {3} + (Dom(1) ∩ Dom(2) ∩ Dom(8) ∩ Dom(4)) = {0,1,3} |
| 4 | N | {4} + (Dom(3) ∩ Dom(7)) = {0,1,3,4} |
| 5 | N | {5} + Dom(4) = {0,1,3,4,5} |
| 6 | N | {6} + Dom(4) = {0,1,3,4,6} |
| 7 | N | {7} + (Dom(5) ∩ Dom(6) ∩ Dom(10)) = {0,1,3,4,7} |
| 8 | N | {8} + Dom(7) = {0,1,3,4,7,8} |
| 9 | N | {9} + Dom(8) = {0,1,3,4,7,8,9} |
| 10 | N | {10} + Dom(8) = {0,1,3,4,7,8,10} |

Dominator Example



Control Flow Analysis

| Block | Dom | | |
|-------|---------|------------------|------------------|
| | initial | iteration1 | iteration2 |
| 0 | {0} | {0} | {0} |
| 1 | N | {0,1} | {0,1} |
| 2 | N | {0,1,2} | {0,1,2} |
| 3 | N | {0,1,3} | {0,1,3} |
| 4 | N | {0,1,3,4} | {0,1,3,4} |
| 5 | N | {0,1,3,4,5} | {0,1,3,4,5} |
| 6 | N | {0,1,3,4,6} | {0,1,3,4,6} |
| 7 | N | {0,1,3,4,7} | {0,1,3,4,7} |
| 8 | N | {0,1,3,4,7,8} | {0,1,3,4,7,8} |
| 9 | N | {0,1,3,4,7,8,9} | {0,1,3,4,7,8,9} |
| 10 | N | {0,1,3,4,7,8,10} | {0,1,3,4,7,8,10} |

43

Computing IDOM from DOM

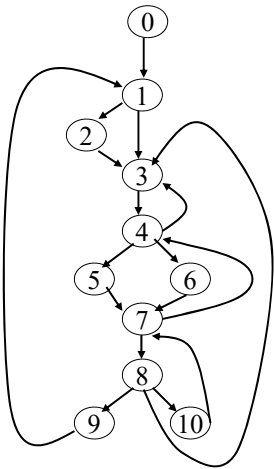
1. For each node n , initially set $IDOM(n) = DOM(n) - \{n\}$ (SDOM - strict dominators)
2. For each node p in $IDOM(n)$, see if p has dominators other than itself also included in $IDOM(n)$: if so, remove them from $IDOM(n)$

- The immediate dominator m of n is the strict dominator of n that is closest to n

Control Flow Analysis

44

I-Dominator Example



| Block | IDom | |
|-------|----------------|-----------------------------------|
| | initial (SDOM) | |
| 0 | {} | {} |
| 1 | {0} | {0} |
| 2 | {0,1} | {1} // 0 - 1's dominator |
| 3 | {0,1} | {1} // 0 - 1's dominator |
| 4 | {0,1,3} | {3} // 0,1 - 3's dominators |
| 5 | {0,1,3,4} | {4} // 0,1,3 - 4's dominators |
| 6 | {0,1,3,4} | {4} // 0,1,3 - 4's dominators |
| 7 | {0,1,3,4} | {4} // 0,1,3 - 4's dominators |
| 8 | {0,1,3,4,7} | {7} // 0,1,3,4 - 7's dominators |
| 9 | {0,1,3,4,7,8} | {8} // 0,1,3,4,7 - 8's dominators |
| 10 | {0,1,3,4,7,8} | {8} // 0,1,3,4,7 - 8's dominators |

Control Flow Analysis

45

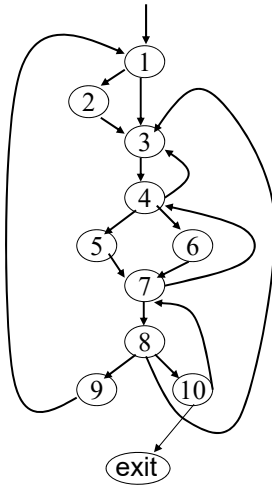
Post-Dominance

- Related concept
- Node d of a CFG *post-dominates* node n if every path from n to the exit node passes through d ($d \text{ pdom } n$)
 - $\text{Pdom}(n)$: the set of post-dominators of node n
 - Every node post-dominates itself: $n \in \text{Pdom}(n)$
- Each node n has a unique *immediate post dominator* m

Control Flow Analysis

46

Post-dominator Example

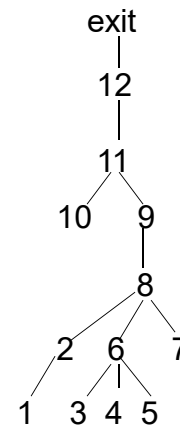
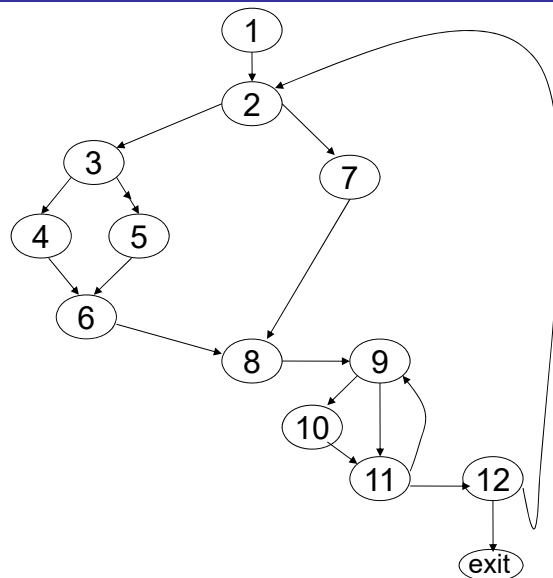


| Block | Pdom | IPdom |
|-------|---------------------|-------|
| 1 | {3,4,7,8,10,exit} | 3 |
| 2 | {2,3,4,7,8,10,exit} | 3 |
| 3 | {3,4,7,8,10,exit} | 4 |
| 4 | {4,7,8,10,exit} | 7 |
| 5 | {5,7,8,10,exit} | 7 |
| 6 | {6,7,8,10,exit} | 7 |
| 7 | {7,8,10,exit} | 8 |
| 8 | {8,10,exit} | 10 |
| 9 | {1,3,4,7,8,10,exit} | 1 |
| 10 | {10,exit} | exit |

Control Flow Analysis

47

CFG



Control Flow Analysis

48

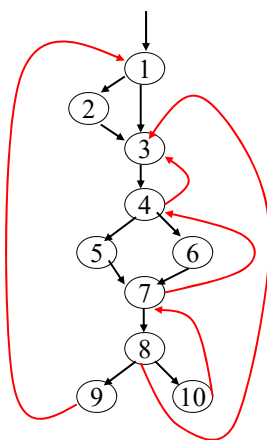
Natural Loops

- Natural loops that are suitable for improvement have two essential properties:
 - A loop must have a single entry point called *header*
 - There must be at least one way to iterate the loop, i.e., at least one path back to the header
- Identifying natural loops
 - Searching for **back edges** ($n \rightarrow d$) in CFG whose heads *dominate* their tails
 - For an edge $a \rightarrow b$, b is the *head* and a is the *tail*
 - A back edge flows from a node n to one of n 's dominators d
 - The natural loop for that edge is $\{d\}$ + the set of nodes that can reach n without going through d
 - d is the header of the loop

Control Flow Analysis

49

Back Edge Example



Back edges?

| Block | Dom | IDom |
|-------|--------------|------|
| 1 | 1 | — |
| 2 | 1,2 | 1 |
| 3 | 1,3 | 1 |
| 4 | 1,3,4 | 3 |
| 5 | 1,3,4,5 | 4 |
| 6 | 1,3,4,6 | 4 |
| 7 | 1,3,4,7 | 4 |
| 8 | 1,3,4,7,8 | 7 |
| 9 | 1,3,4,7,8,9 | 8 |
| 10 | 1,3,4,7,8,10 | 8 |

Control Flow Analysis

50

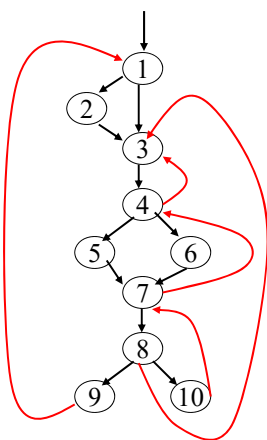
Identifying Natural Loops

- Given a back edge $n \rightarrow d$, the natural loop of the edge includes
 - Node d
 - Any node that can reach n without going through d
- Loop construction
 - Set $loop = \{d\}$
 - Add n into $loop$ if $n \neq d$
 - Consider each node $m \neq d$ that we know is in $loop$, make sure that m 's predecessors are also inserted in $loop$

Control Flow Analysis

51

Natural Loops Example



| Back edge | Natural loop |
|--------------------|-------------------------------------|
| $10 \rightarrow 7$ | $\{7, 10, 8\}$ |
| $7 \rightarrow 4$ | $\{4, 7, 5, 6, 10, 8\}$ |
| $4 \rightarrow 3$ | $\{3, 4, 7, 5, 6, 10, 8\}$ |
| $8 \rightarrow 3$ | |
| $9 \rightarrow 1$ | $\{1, 9, 8, 7, 5, 6, 10, 4, 3, 2\}$ |

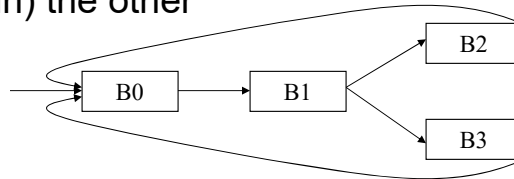
- Why neither $\{3, 4\}$ nor $\{4, 5, 6, 7\}$ is a natural loop?

Control Flow Analysis

52

Inner Loops

- A useful property of natural loops: unless two loops have the same header, they are either disjoint or one is entirely contained (nested within) the other



- An inner loop is a loop that contains no other loops
 - Good optimization candidate
 - The inner loop of the previous example: {7,8,10}

Control Flow Analysis

53

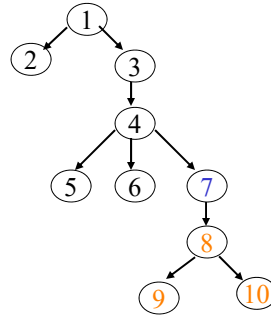
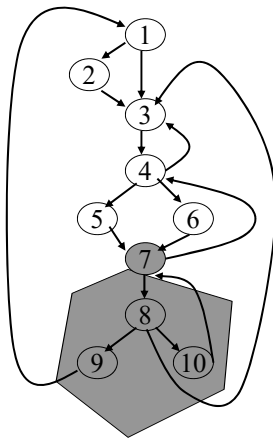
Dominance Frontiers

- For a node n in CFG, $DF(n)$ denotes the **dominance frontier** set of n
 - $DF(n)$ contains all nodes x s.t. n dominates an immediate predecessor of x but does not strictly dominate x
 - For this to happen, there is some path from node n to x , $n \rightarrow \dots \rightarrow y \rightarrow x$ where $(n \text{ DOM } y)$ but $!(n \text{ SDOM } x)$
 - Informally, $DF(n)$ contains the first nodes reachable from n that n does not strictly dominate, on each CFG path leaving n
- Used in SSA calculation and redundancy elimination

Control Flow Analysis

54

Dominance Frontier for Node 7



Paths of interest:

$7 \rightarrow 4$

$7 \rightarrow 8 \rightarrow 3$

$7 \rightarrow 8 \rightarrow 9 \rightarrow 1$

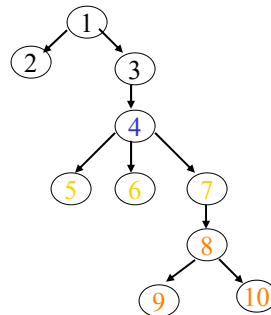
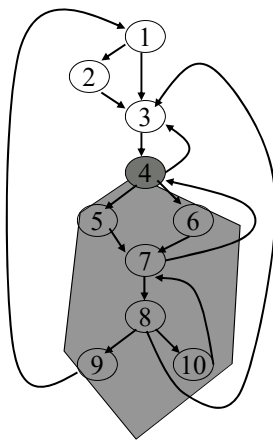
$7 \rightarrow 8 \rightarrow 10 \rightarrow 7$

$DF(7) = \{1, 3, 4, 7\}$

Control Flow Analysis

55

Dominance Frontier for Node 4



Paths of interest:

$DF(4) = \{1, 3, 4\}$

Control Flow Analysis

56

Computing Dominance Frontiers

□ Easiest way:

$DF(x) = \text{SUCC}(\text{DOM}^{-1}(x)) - \text{SDOM}^{-1}(x)$ where
SUCC(x) = set of successors of x in the CFG

- But not the most efficient

□ Observation

- Nodes in a DF must be join nodes
- The predecessor of any join node j must have j in its DF unless it dominates j
- The dominators of j 's predecessors must have j in their DF sets unless they also dominate j

Control Flow Analysis

57

Computing Dominance Frontiers

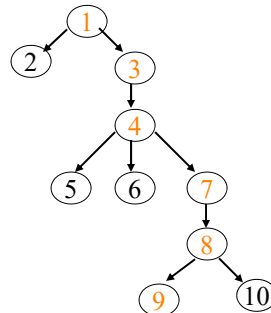
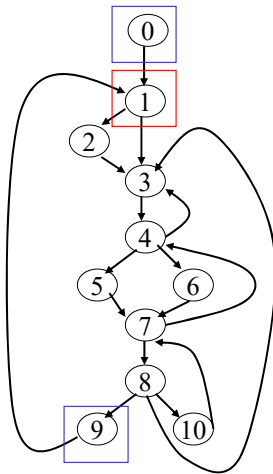
```
for all nodes n, initialize DF(n) = ∅
for all nodes n
  if n has multiple predecessors, then
    for each predecessor p of n
      runner = p
      while (runner ≠ IDom(n))
        DF(runner) = DF(runner) ∪ {n}
        runner = IDom(runner)
```

- First identify join nodes j in CFG
- Starting with j 's predecessors, walk up the dominator tree until we reach the immediate dominator of j
 - Node j should be included in the DF set of all the nodes we pass by except for j 's immediate dominator

Control Flow Analysis

58

Computing Dominance Frontier

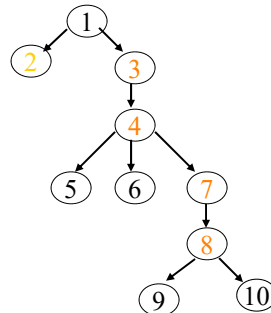
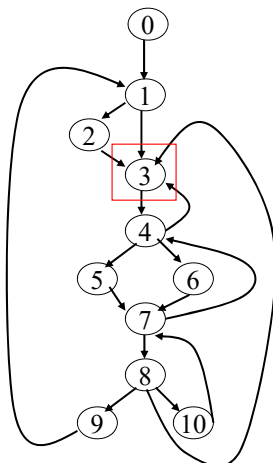


Join node 1:
 runner = 0 = IDom(1)
 runner = 9 : DF(9) += {1}
 runner = 8 : DF(8) += {1}
 runner = 7 : DF(7) += {1}
 runner = 4 : DF(4) += {1}
 runner = 3 : DF(3) += {1}
 runner = 1 : DF(1) += {1}
 runner = 0 = IDom(1)

Control Flow Analysis

59

Computing Dominance Frontier

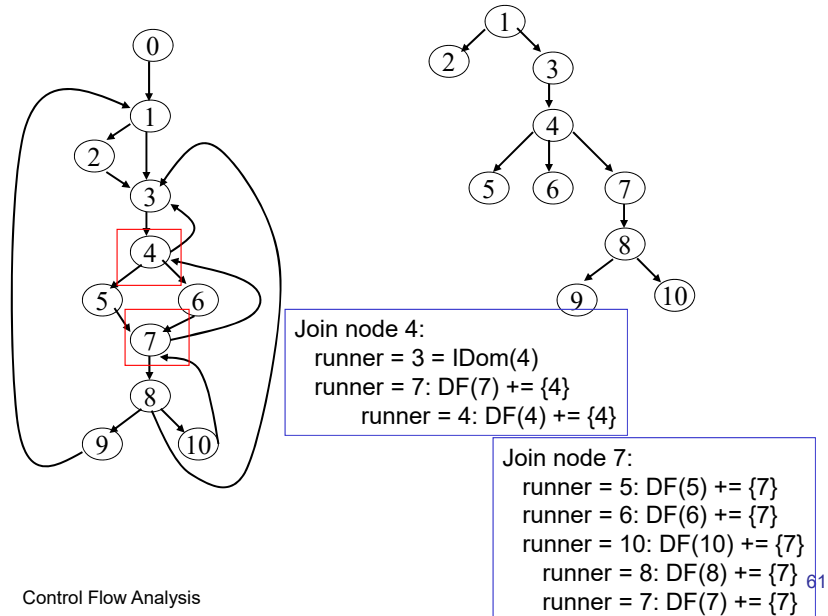


Join node 3:
 runner = 1 = IDom(3)
 runner = 2: DF(2) += {3}
 runner = 4: DF(4) += {3}
 runner = 3: DF(3) += {3}
 runner = 8 : DF(8) += {3}
 runner = 7 : DF(7) += {3}

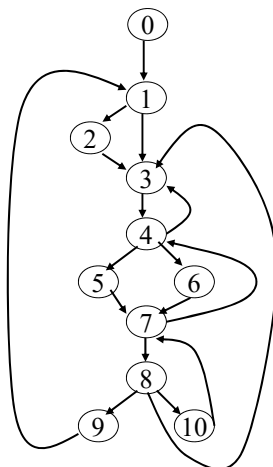
Control Flow Analysis

60

Computing Dominance Frontier



Dominance Frontier Example

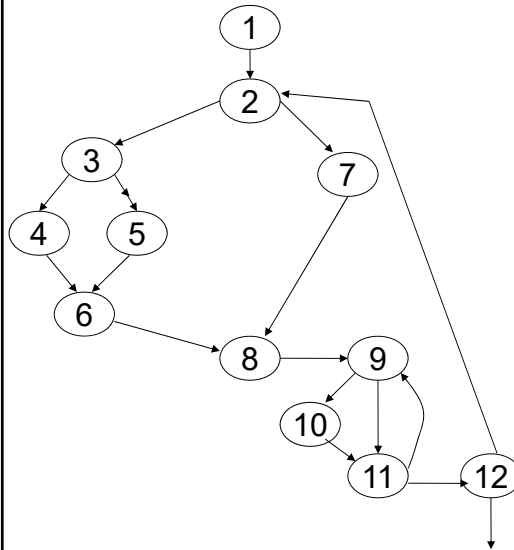


| Block | DF |
|-------|-----------|
| 1 | {1} |
| 2 | {3} |
| 3 | {1,3} |
| 4 | {1,3,4} |
| 5 | {7} |
| 6 | {7} |
| 7 | {1,3,4,7} |
| 8 | {1,3,7} |
| 9 | {1} |
| 10 | {7} |

Control Flow Analysis

62

Example 2



Control Flow Analysis

| Bloc k | DF |
|-----------|----|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |

63

Dominator-based Analysis

- Idea
 - Use dominators to discover loops for optimization
- Advantages
 - Sufficient for use by iterative data-flow analysis and optimizations
 - Least time-intensive to implement
 - Favored by most current optimizing compilers
- Alternative approach
 - Interval-based analysis/structural analysis

Control Flow Analysis

64

Summary

- CFG construction
 - Basic blocks identification
- CFG traversal
 - Depth-first spanning tree
 - Vertex ordering
- CFG analysis
 - Important regions: EBB and loop
 - Dominators
 - Dominance frontiers
- Additional references
 - Advanced compiler design and implementation, by S. Muchnick, Morgan Kaufmann