Module 14: Approaches to Control Flow Analysis

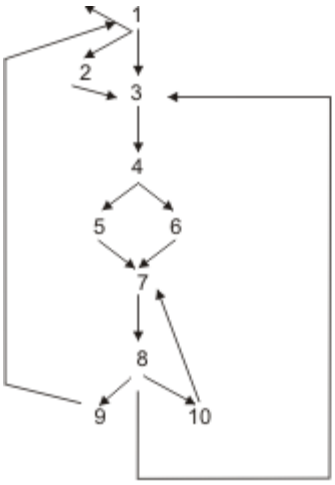
Lecture 27: Algorithm and Interval

The Lecture Contains:

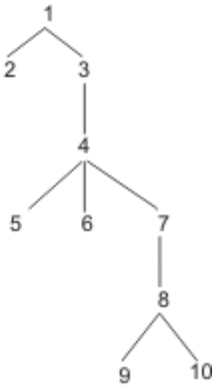
Previous Next

Algorithm to Find Dominators

D(n0) := {n0};
for n in N-{n0} do D(n) := N;
while changes to any D(n) occur do
for n in N-{n0} do
D(n) := {n} n D(p);



Control Flow Graph



Dominator Tree

Loop Detection

- Search for a back-edge such that a → b is an edge and b dom a.
- Given a back edge a → b
  - Find set of nodes that can reach node a without going through node b
  - And node b
- These nodes form a natural loop with b as header.
- Given back edge a → b natural loop is a sub graph which contains a, b and all the nodes which can reach a without passing through b

Previous Next

Algorithm to Detect Loops

```
stack := empty;
loop := {b};
insert(a);
while stack is not empty do begin
pop m of the stack;
for each predecessor p of m do insert(p)
end;
procedure insert(m);
if m is not in loop then begin
loop := loop    {m};
push m onto stack
end;
```

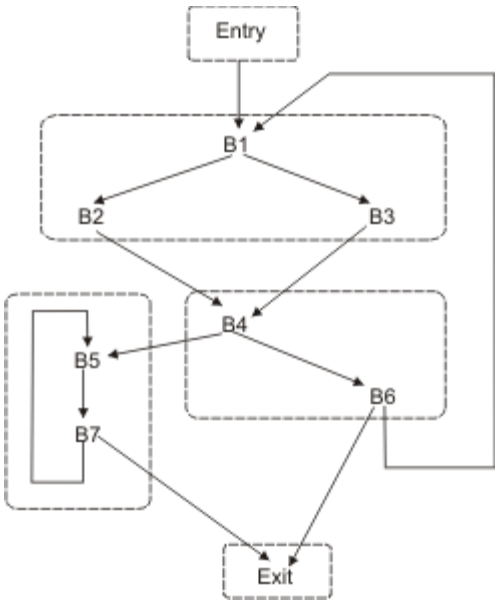
Approaches to Control Flow Analysis

- Approach 1; Use dominators to discover loops use loops in optimization do iterative data flow analysis
- Approach 2: Use interval analysis analyze overall structure of the program decompose it into nested regions
  the nesting structure forms a control tree
- Approach 3: Use structural analysis speeds up dataflow analysis also called elimination method
- Most compilers use the first approach
- It is easy to implement and provides most of the information for optimization
- It is inferior to the other two approaches
- Interval based approaches are faster
- Interval based approach can be used in incremental analysis
- Structural analysis makes control flow transformations easy

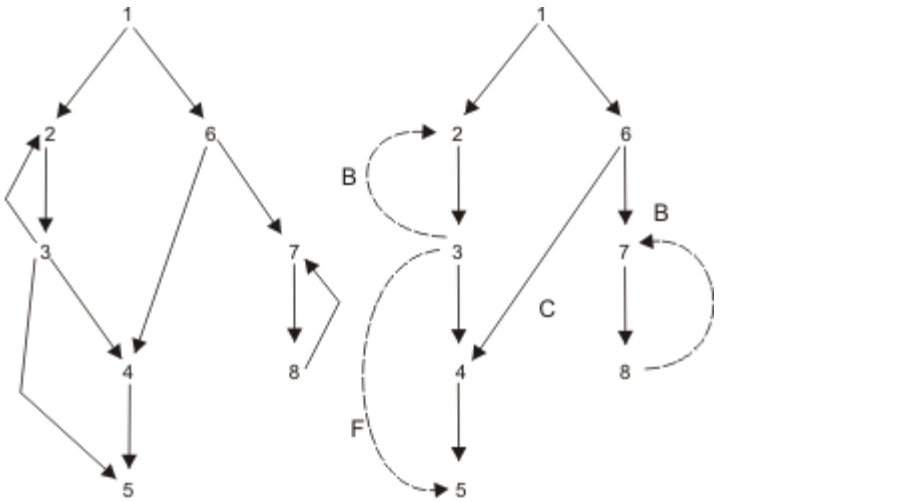
Previous Next

Extended Basic Block

- Maximal sequence of instructions begining with a header and no other join nodes
- Single entry multiple exits block

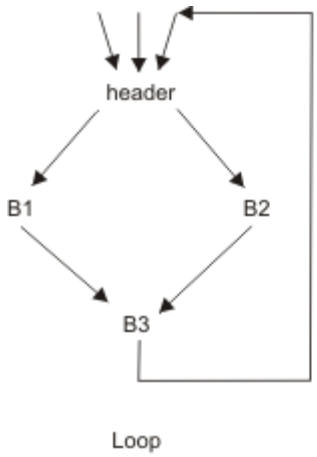


- Depth first traversal contains (i) all the nodes and (ii) edges which make depth first order. This is called depth first spanning tree.
- Forward edges: from a node to descendent
- Back edges: go from node to ancestor
- Cross edges: connects nodes such that neither is an ancestor in dfs

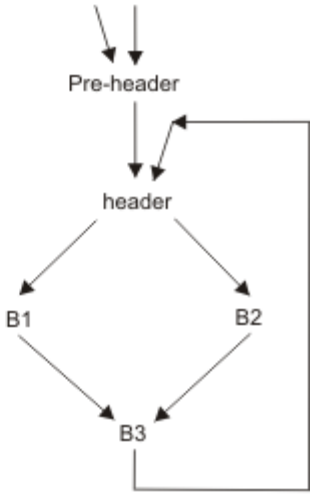


Pre-Header

- Many optimizations require code movement from inside a loop to just before it.
- Pre-header is a new empty block just before header.
  - All edges from outside to header go to pre-header
  - A new edge goes from pre-header to header



Loop



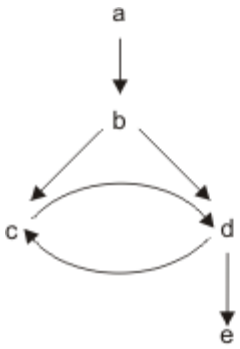
Loop with pre-header

Loops With Common Headers

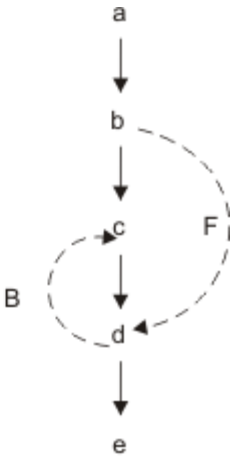
- Two loops with different headers are either disjoint or nested
- Two loops with the same header
  - Not clear if they are nested
  - Or are one loop
  - Can not be decided without looking at the source code

Previous Next

- A ntaural loop is a strongly connected region
- Each strongly connected region is not a loop



control flow graph

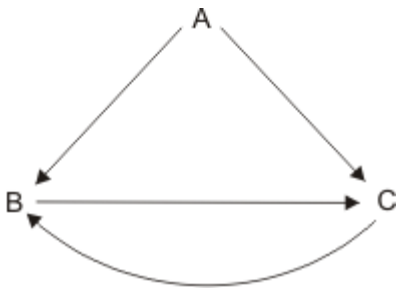


control flow graph

Reducible Flow Graphs

- A graph is reducible if applying a sequence of transformations reduce it to a single node
- A flow graph G=(N,E) is reducible if E can be partitioned into two disjoint groups Eb and Ef such that:
  - (N,Ef) forms an acyclic graph in which every node can be reached from entry
  - Eb are the backw edges, edges whose heads dominate their tails.

Example of a Non Reducible Flow Graph

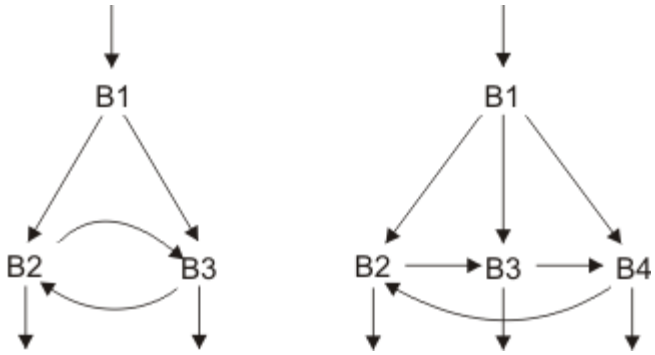


Previous Next

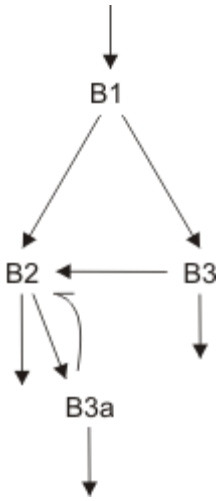
Module 14: Approaches to Control Flow Analysis

Lecture 27: Algorithm and Interval

- In a reducible graph
  - All the loops are natural loops
  - There are no jumps in the middle of a loop
- Improper regions are multiple entry strongly connected regions



- Programming languages do not allow irreducible flow graphs
- Fortran with loops of if and gotos construct irreducible flow graphs
- Most of the optimizations do not work for irreducible flow graphs
- Node splitting is a possible solution

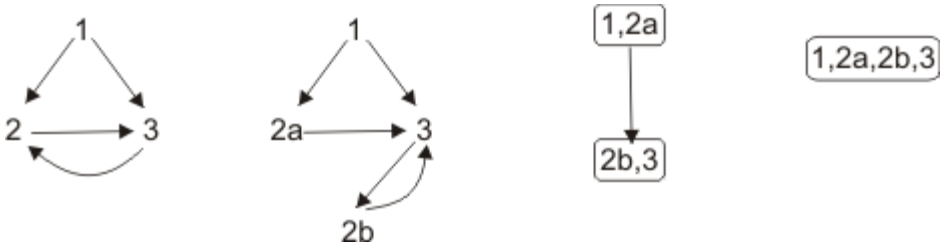


Previous Next

Node Splitting

Transformation used for converting non-reducible flow graphs to reducible flow graphs. If there is a node n with k predecessors:

- Split *n* into *k* nodes generating nodes *n1, n2, ..., nk*
- The ith predecessor of *n* becomes the predecessors of *ni*
- All the successors of *n* become successors of all of the *ni*'s



Interval Analysis

- Divides flow graph into various regions
- Consolidate each region into a (abstract) node
- Resulting flow graph is an abstract flow graph
- Result of transformations produce control tree
  - Root represents the original graph
  - Leaf of control tree are basic blocks
  - Internal nodes are abstract nodes
  - Edges represent relationship between abstract nodes

Intervals

- Each interval has a header node that dominates all the nodes in the interval.
- Given a flow graph with initial node n0, the interval with header n is denoted by I(n) and is defined as:
  - N is in I(n)
  - If all the predecessors of a node m are in I(n) then m is in I(n)
  - No other node is in I(n)

Interval Partition

Construct I(n0);
while there is a node m not yet selected
but with a selected predecessor do
construct I(m);

Construct I(n);
I(n) := {n};
while there exists a node m 6= n0
all of whose predecessors are in I(n) do
I(n) := I(n) {m}

I(1) = 1, 2
I(3) = 3
I(4) = 4, 5, 6
I(7) = 7, 8, 9, 10

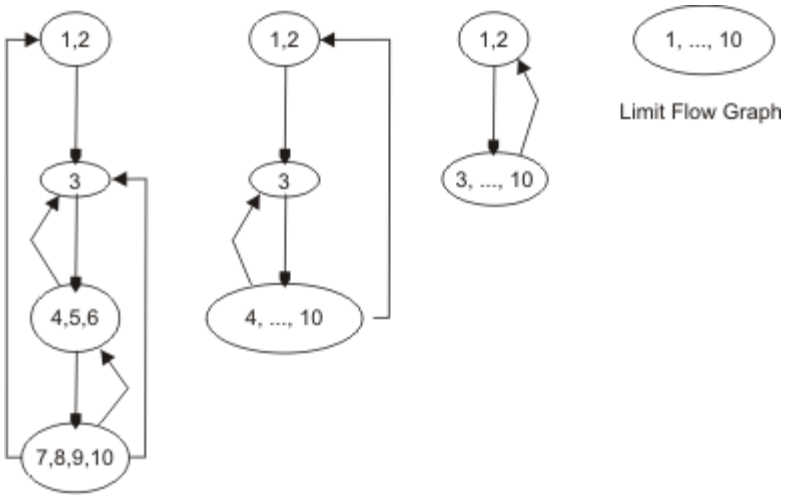
Interval Graphs

From the interval graph construct a new graph I(G)

- Nodes of the new graph correspond to interval partitions.
- Initial node is the node containing initial node of G.
- There is an edge from node I to node J if there is some edge from element of I to the header of J.

Interval partition can be repeatedly applied to the new Interval graph.

Previous Next

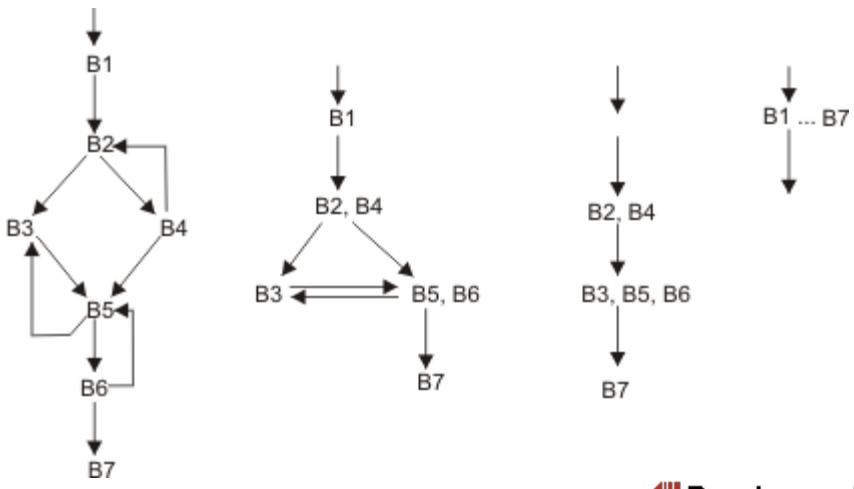


If the limit flow graph is a single node then the graph is reducible.

Control Tree

The result of applying a sequence of such transformations produces a control tree. It is defined as follows:

- The root is sn abstract graph representing the original flowgraph
- The leaves are individual basic blocks
- The internal nodes are abstract nodes representing regions of the flowgraph
- The edges of the tree represent the relationship between each abstract node and the regions that are its descendants



Previous Next