Control Flow Graphs

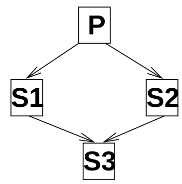
Nodes *Statements or Basic Blocks*
(Maximal sequence of code with branching only allowed at end)

Edges *Possible transfer of control*

Example:

CFG

if P
then S1
else S2
S3



P predecessor of S1 and S2
S1, S2 sucessors of P

Finding Basic Blocks

Identify Headers

The first instruction is a header
The target of any branch is a header
The instruction following any branch is a header
Add new nodes Entry, Exit as headers

For each header, add successive instructions to BB until reach next header

Ex.

```
    a := 1
    b := 2
    if P then go to L1
    c := 3
L1: d := 4
    e := 5
```

Finding Edges in CFG

There is a directed edge *B1* ⟶ *B2* if either:

There is a branch from last instruction in B1 to header of B2

B2 immediately follows B1, and B1 does not end in an unconditional branch

There is an edge from *Entry* to each initial BB

There is an edge from each final BB to *Exit*

There is at most one edge *B1* ⟶ *B2*

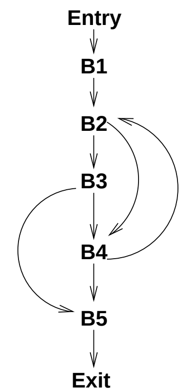
Example

```
    a := 1
    b := 2
L1: c := a + b
    d := c - a
    if ( ) go to L2
    d := a + b
    e := e + 1
    if ( ) go to L3
L2: b := a + b
    e := c - a
    if ( ) go to L1
L3: a := b + d
    b := a - d
```

Example

```
    a := 1    *        B1
    b := 2

L1: c := a + b   *     B2
    d := c - a
    if ( ) go to L2

    d := a + b   *     B3
    e := e + 1
    if ( ) go to L3

L2: b := a + b   *     B4
    e := c - a
    if ( ) go to L1

L3: a := b + d *       B5
    b := a - d
```

* = header



Extended Basic Blocks

Maximal connected set of basic blocks with a header, and

each block (except the header) having a single predecessor

Tree of basic block nodes rooted at header

Advantage: Larger region for local optimization

Ex.

```
    a := 1
    b := 2
    if P then go to L1
    c := 3
L1: d := 4
    e := 5
```

Why are CFG's Useful?

Can summarize info per BB

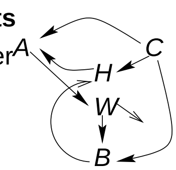
A pass over CFG is shorter than pass over program

Can easily find unreachable code

Makes syntactic structure (like loops) easy to find

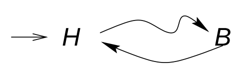
What are loops?

1. **Strongly connected components**
   any node reachable from any other
   Maximal SCC



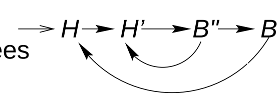
2. **Natural Loops**

   via dominators

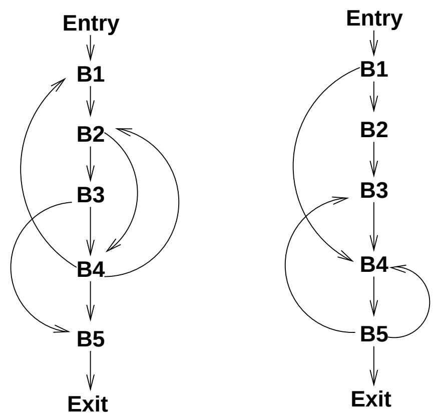


3. **Intervals**

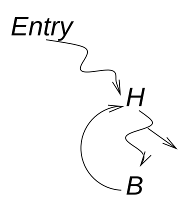
   via depth first spanning trees



Examples: Finding Maximal SCC's



What are loops?



Suppose

1. All paths from *Entry* to *B* go through *H*, and
   (Header H dominates B)
2. There is an edge from *B* to *H*
   (NL Back edge)
3. All nodes are reachable from Entry.

Natural Loop (wrt $B \longrightarrow H$ **) subgraph of CFG**

Nodes: *H*, and all nodes *N* that reach *B* without going through *H*

Edges: induced as subgraph

Can find natural loop by backward traversal from B

Dominators

Def. *A* dominates *B in CFG G iff*

A lies on every path in G from Entry to B.

Facts:

A dominates A **(reflexive)**

A dominates B & B dominates C implies

A dominates C **(transitive)**

A dominates B & B dominates A implies

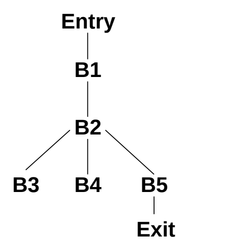
A = B **(anti-symmetric)**

Def. *A* immediately dominates *B iff A dominates B,*
A ≠ B, and there is no C distinct from A and B
such that A dominates C and C dominates B

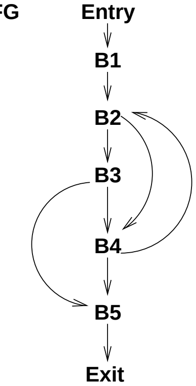
Immediate dominators form a **tree**

Example

Dominator Tree



CFG

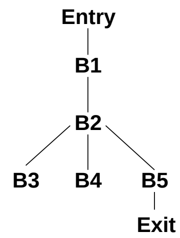


Natural Loop (B4 → B2)

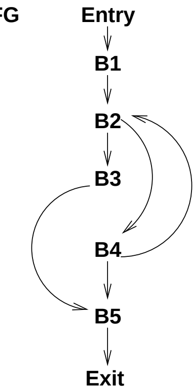
B2, B3, B4

Example

Dominator Tree



CFG

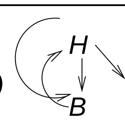


Natural Loop (B4 → B2)

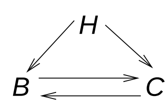
B2, B4

What’s wrong with natural loops?

Don’t find all "loops"
(H doesn’t dominate B)

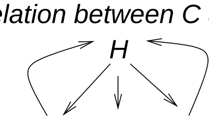


Don’t find irreducible subgraphs
(multiple-entry SCC)



(No dominator relation between C and B)

Hard to tell if nested
when same header H

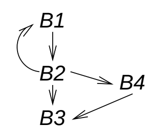


Intervals

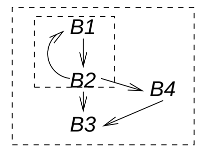
Find "Regions" in CFG
Make each region a node, and continue

Get hierarchical nesting (possibly, control tree)

Ex. Cocke-Allen Intervals

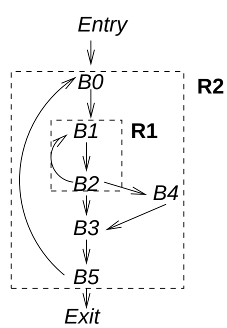


Structural Analysis

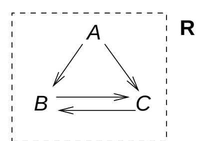


Tarjan Intervals

Ex.



Ex.



Smallest single-entry region containing irreducible subgraph

Depth First Spanning Trees

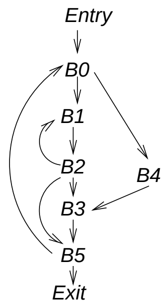
Spanning tree of graph (includes all nodes of graph)

Formed by depth-first seach

Visit descendants of node before non-descendant siblings

Assign depth-first numbering

Successive numbers, in order first visited



Process nodes

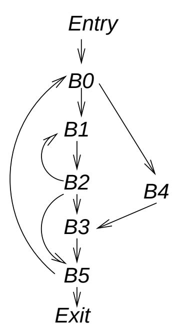
pre-order

in-order

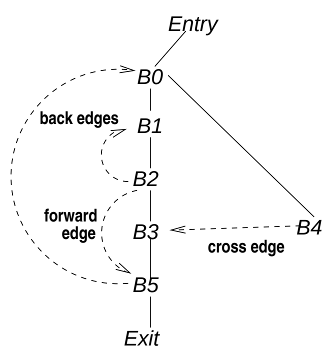
post-order

Constructing Tarjan Intervals

Ex. CFG



DFS Tree



Constructing Tarjan Intervals (cont'd)

Back:

From node to an ancestor in tree

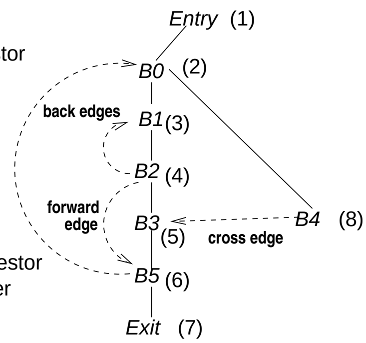
Forward:

From node to a descendant in tree

Cross:

From node to non-ancestor with smaller DF number

DFS Tree



Qu: How tell if ancestor or descendant?

Constructing Tarjan Intervals (cont'd)

For each back edge B → H, in reverse pre-order of headers:

Find Interval I(H):

Starting from B, traverse CFG edges backwards.

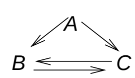
Stop when get to header H, or node with lower number than H.
(in the latter case, the CFG is irreducible).

The nodes traversed (including H) constitute I(H).

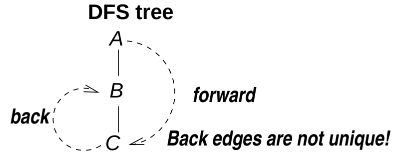
Replace nodes in I(H) with new node $N_{I(H)}$ that points to it.

Result: Hierarchical CFG with Tarjan Intervals

Irreducible subgraph



DFS tree



Irreducible subraph is an interval!