Loop analysis

Program representations:

- Control Flow Graph (seen already)
- Spanning trees
- Strongly connected regions
- Identifying loops
- Reducible CFGs

Spanning Trees

Given CFG $\langle V, E, Entry, Exit \rangle$:

- V: vertices v
- E: edges $v \to w$
- $Entry \in V$: unique entry
- $Exit \in V$: unique exit

construct spanning tree ST $\langle V_T, E_T, Root_T, Exit_T \rangle$:

- $V_T = V$
- $E_T \subseteq E$
- $Root_T = Entry$
- $Exit_T = Exit$

Given ST, each edge $v \to w$ in the CFG can be partitioned as follows:

1. *Tree*: $v \to w$ in both CFG and ST
2. *Advancing*: $v \to w$ is not in ST, but w is a descendant of v in ST
3. *Back*: $v = w$ or w is a proper ancestor of v in ST
4. *Cross*: w neither an ancestor nor descendant of v in ST

Spanning Tree Algorithm

Span($Root$)

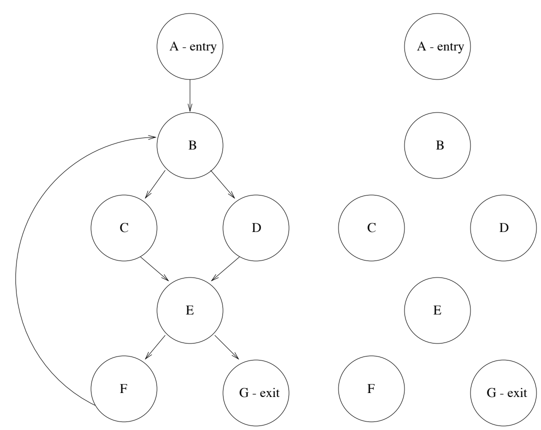
procedure Span(v)
  ST.addVertex(v)
  for $w : v \to w \in$ CFG **do**
    if $w \notin$ ST **then**
      ST.addEdge($v \to w$)
      Span(w)
  endfor
end Span

For convenience, assume all V are reachable from $Entry$:

$$\forall v \in V : Entry \to^* v$$

$Entry \not\to^* v \Rightarrow$ add edge $Entry \to v$

Spanning Tree: Example

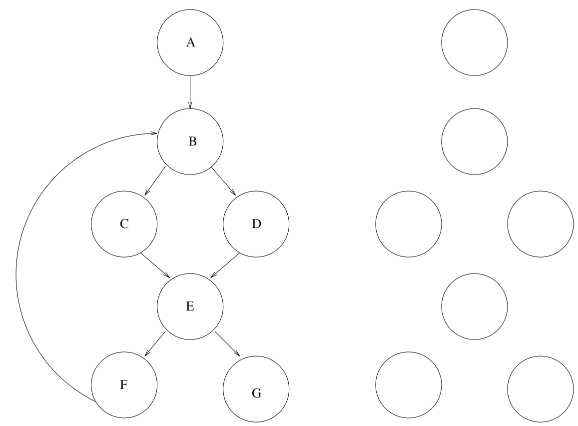


Spanning Edge Identification

$i \leftarrow 0$
DFST($Root$)

procedure DFST(v)
  ST.addVertex(v)
  v.num $\leftarrow$ i++
  v.inStack $\leftarrow$ true
  for $w : v \rightarrow w \in$ CFG **do**
    if $w \notin$ ST **then**
      $v \rightarrow w$ is a tree edge
      ST.addEdge($v \rightarrow w$)
      DFST(w)
    else if w.inStack
      $v \rightarrow w$ is a back edge
    else if $w \in$ ST and w.num $>$ v.num
      $v \rightarrow w$ is a forward edge
    else
      $v \rightarrow w$ is a cross edge
  endfor
  v.inStack $\leftarrow$ false
end DFST

Spanning Edge Identification: Example



Cycles: Strongly Connected Regions (SCR)

$\forall s_1, s_2 \in S$, if S is a cycle then $s_1 \rightarrow^* s_2$ and $s_2 \rightarrow^* s_1$

Compute maximal SCR on a directed graph.

Robert Tarjan, *Depth-First Search and Linear Graph Algorithms*, *SIAM J. Computing 1*(2):146–160, June 1972

- Uses a depth-first spanning tree
  left-to-right pre-order number in *Number*
- Tracks the lowest numbered v to which each vertex has a path in *lowlink*
- Determines a number for SCR to which v belongs.

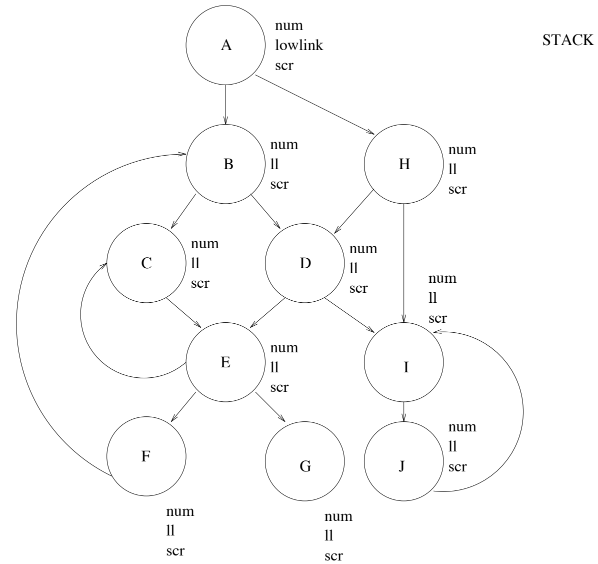
Tarjan's maximal SCR algorithm

$i \leftarrow 0$
for $v \in V$ **do** v.lowlink $\leftarrow 0$; v.num $\leftarrow 0$ **endfor**
scr $\leftarrow 0$
stack $\leftarrow$ empty
for $v \in V$ **do**
  if v.num $= 0$ **then** Tarjan(v)
endfor

Tarjan's maximal SCR algorithm (*cont.*)

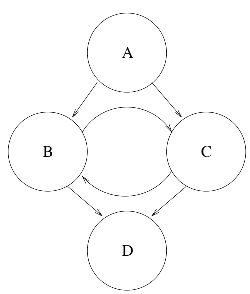
procedure Tarjan(v)
  i++; v.num $\leftarrow i$; v.lowlink $\leftarrow i$
  stack.push(v)
  for $w : v \to w$ **do**
    if w.num $= 0$ **then**
      Tarjan(w)
      v.lowlink $\leftarrow \min(v$.lowlink$, w$.lowlink$)$
    else if $w \in$ stack **then**
      v.lowlink $\leftarrow \min(v$.lowlink$, w$.lowlink$)$
  endfor
  if v.lowlink $= v$.num **then**
    scr++
    repeat
      $w \leftarrow$ stack.pop()
      w.scr $\leftarrow$ scr
    until $w = v$
end Tarjan

Tarjan's maximal SCR algorithm:Example

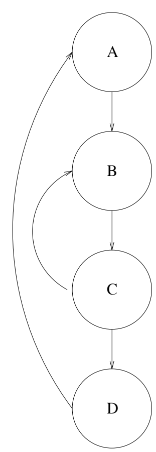


Identifying Loops and Loop Headers

- DFST does not find a unique header in irreducible graphs
- SCRs do not differentiate inner loops



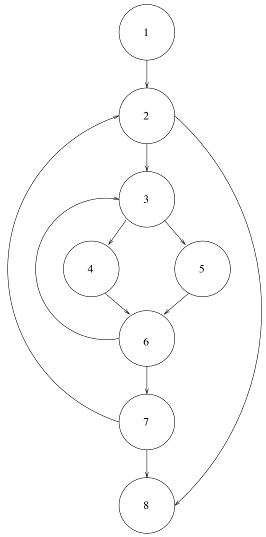
IRREDUCIBLE



Natural Loop

- Single entry, *header* dominates all vertices in loop.
  dominates: $v \text{ dom } w \Leftrightarrow$ every path $Entry \to^* w$ passes through v.
- There is at least one path from the header to itself.
- All vertices and edges on a path from the header to any back edges to the header are in the loop.
- Two natural loops are either entirely disjoint, or one is a proper subset of the other.

Natural Loop Example



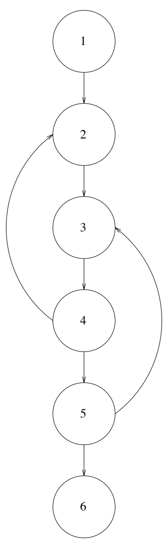
Natural Loop Algorithm

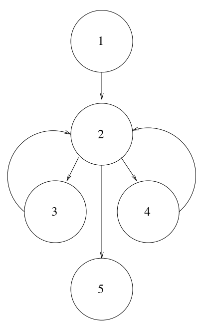
Given a back edge $(t \rightarrow h)$:

loop.addVertex(h)
loop.addEdge($t \rightarrow h$)
insert(t)

procedure insert(v)
  if $v \notin$ loop **then**
    loop.addVertex(v)
    for $p \rightarrow v$ **do**
      loop.addEdge($p \rightarrow v$)
      insert(p)
    endfor
end insert

Improperly Nested Loops





One loop or two?

Inner loop?

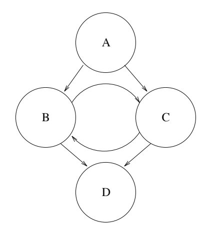
Outer loop?

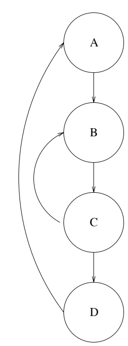
Reducible Control Flow Graphs

Intuitively, if all loops are single entry, the CFG is reducible.

More formally:

- Given a spanning tree, for every back edge in the CFG, the head *dominates* the tail (i.e., you cannot execute the tail without executing the head first).





Next Time

Dominators and Control Dependences