Control Flow Analysis - Part 1

Y.N. Srikant

Department of Computer Science
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Compiler Design

Outline of the Lecture

- Why control flow analysis?
- Dominators and natural loops
- Intervals and reducibility
- $T_1 - T_2$ transformations and graph reduction
- Regions

Why Control-Flow Analysis?

- Control-flow analysis (CFA) helps us to understand the structure of control-flow graphs (CFG)
- To determine the loop structure of CFGs
- Formulation of conditions for code motion use dominator information, which is obtained by CFA
- Construction of the static single assignment form (SSA) requires dominance frontier information from CFA
- It is possible to use interval structure obtained from CFA to carry out data-flow analysis
- Finding Control dependence, which is needed in parallelization, requires CFA

Dominators

- We say that a node d in a flow graph *dominates* node n, written d *dom* n, if every path from the initial node of the flow graph to n goes through d
- Initial node is the root, and each node dominates only its descendents in the tree (including itself)
- The node x *strictly dominates* y, if x dominates y and $x \neq y$
- x is the *immediate dominator* of y (denoted *idom*(y)), if x is the closest strict dominator of y
- A *dominator tree* shows all the immediate dominator relationships
- Principle of the dominator algorithm
  - If $p_1, p_2, ..., p_k$, are all the predecessors of n, and $d \neq n$, then d *dom* n, iff d *dom* p_i for each i

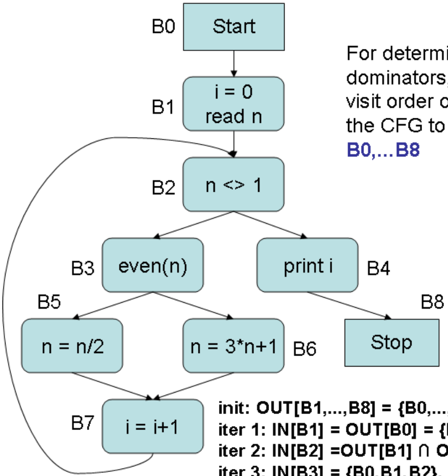
An Algorithm for finding Dominators

- $D(n) = OUT[n]$ for all n in N (the set of nodes in the flow graph), after the following algorithm terminates
- { /* n_0 = initial node; N = set of all nodes; */
  $OUT[n_0] = \{n_0\}$;
  for n in $N - \{n_0\}$ do $OUT[n] = N$;
  while (*changes to any* $OUT[n]$ *or* $IN[n]$ *occur*) do
  for n in $N - \{n_0\}$ do

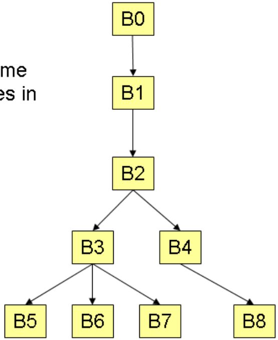
$$\begin{aligned} IN[n] &= \bigcap_{P \text{ a predecessor of } n} OUT[P]; \\ OUT[n] &= \{n\} \cup IN[n] \end{aligned}$$

  }

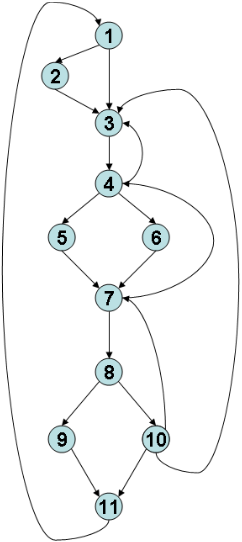
Dominator Example



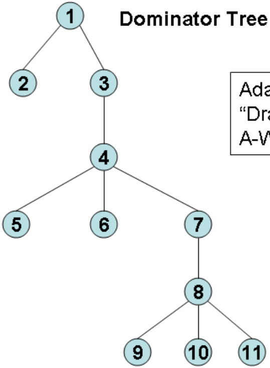
For determining dominators, assume visit order of nodes in the CFG to be **B0,...B8**



init: OUT[B1,...,B8] = {B0,...,B8}, OUT[B0] = {B0}
iter 1: IN[B1] = OUT[B0] = {B0}, OUT[B1] = {B0,B1}
iter 2: IN[B2] =OUT[B1] ∩ OUT[B7] = {B0,B1} , OUT[B2] = {B0,B1,B2}
iter 3: IN[B3] = {B0,B1,B2}, OUT[B3] = {B0,B1,B2,B3}
IN[B4] = {B0,B1,B2}, OUT[B4] = {B0,B1,B2,B4}
iter 4: IN[B5] = {B0,B1,B2,B3} = IN[B6], OUT[B5] = {B0,B1,B2,B3,B5}
OUT[B6] = {B0,B1,B2,B3,B6}, OUT[B8] = {B0,B1,B2,B4,B8}
iter 5: IN[B7] = OUT[B5] ∩ OUT[B6] = {B0,B1,B2,B3}
OUT[B7] = {B0,B1,B2,B3,B7}



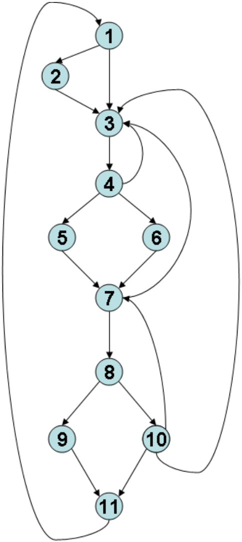
Flow Graph



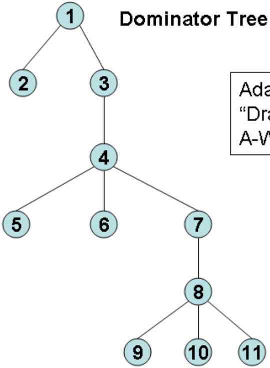
Dominator Tree

Adapted from the "Dragon Book", A-W, 1986

Dominators, Back Edges, and Natural Loops



Flow Graph



Dominator Tree

Adapted from the "Dragon Book", A-W, 1986

Dominators and Natural Loops

- Edges whose heads dominate their tails are called *back edges* ($a \rightarrow b$: $b = head$, $a = tail$)
- Given a back edge $n \rightarrow d$
  - The *natural loop* of the edge is d plus the set of nodes that can reach n without going through d
  - d is the header of the loop
    - A single entry point to the loop that dominates all nodes in the loop
    - Atleast one path back to the header exists (so that the loop can be iterated)