CS 473: COMPILER DESIGN
LOOPS AND DOMINATORS
Loops in Control-flow Graphs

• Most programs spend most of their time in loops, so if we want to optimize, loops are a good place to look

• Like other optimizations, loop optimizations are best applied to a control flow graph IR
  – Other opts may create opportunities for loop opts and vice versa, so it makes sense to alternate between them

• Loops may be hard to recognize at the CFG IR level
  – Many kinds of loops: while, do/while, for, loops with break/continue, goto... all of which turn into some combination of comparisons, labels, jumps, and body code

• Problem: How do we identify loops in the control flow graph?
Definition of a Loop

- A loop is a set of nodes in the control flow graph such that:
  - There is a single distinguished entry point called the header

- Every node is reachable from the header & the header is reachable from every node
  - A loop is a strongly connected component

- No edges enter the loop except to the header

- Nodes with outgoing edges are called loop exit nodes
Nested Loops

- Control-flow graphs may contain many loops
- Loops may contain other loops:

Loop Nesting Tree:
Loop Analysis

• Goal: Identify the loops and nesting structure of the CFG

• Loop analysis is based on the idea of dominators: node A dominates node B if the only way to reach B from the start node is through node A
  – The header of a loop dominates all the nodes in the loop

• An edge in the graph is a back edge if the target node dominates the source node

• Every loop contains at least one back edge
We can define $\text{Dom}[n]$ as a forward dataflow analysis.

Using our usual framework:

- “A node $B$ is dominated by another node $A$ if $A$ dominates all of the predecessors of $B$.”
  \[ \text{in}[n] := \bigcap_{n' \in \text{pred}[n]} \text{out}[n'] \]

- “Every node dominates itself.”
  \[ \text{out}[n] := \text{in}[n] \cup \{n\} \]

We start with every node in all of the sets, and in each iteration remove those that don’t dominate all of the node’s predecessors.

At the end, $\text{out}[n]$ will be the set of nodes that dominate $n$. 
Questions

Top
Dominator Trees

- Result: for each node, a set of nodes that dominate it
- Each node has one *immediate dominator*, the closest node that dominates it
- We can draw a tree where each node’s parent is its immediate dominator
Dominator Trees

- Result: for each node, a set of nodes that dominate it
- Each node has one *immediate dominator*, the closest node that dominates it

CFG:  

Dominator Tree:
Completing Loop Analysis

• Dominator analysis identifies *back edges*: edges $n \rightarrow h$ where $h$ dominates $n$

• Each back edge has a *natural loop*:
  – $h$ is the header
  – All nodes reachable from $h$ that also reach $n$ without going through $h$ are in the loop
Completing Loop Analysis

• Dominator analysis identifies *back edges*: edges \( n \rightarrow h \) where \( h \) dominates \( n \)

• Each back edge has a *natural loop*:
  – \( h \) is the header
  – All nodes reachable from \( h \) that also reach \( n \) without going through \( h \) are in the loop

• It’s convenient for each loop to have a unique header, so if two loops share the same header, we merge them

• If all the nodes in one loop are also in another loop, the first loop is *nested* inside the second
Example Natural Loops

Loop Nesting Tree:

Natural Loops
Questions

Top
Using Loop Information

• Loop nesting depth plays an important role in optimization heuristics
  – Deeply nested loops pay off the most for optimization

• Need to know loop headers / back edges for doing:
  – loop invariant code motion (remove code from loop if it’s the same every time)
  – loop unrolling (execute n iterations of the loop at once)
  – and lots more!

• Dominance information also plays a role in converting to Static Single Assignment form