CONTROL FLOW ANALYSIS

The slides adapted from Vikram Adve
Flow Graphs

Flow Graph: A triple $G=(N,A,s)$, where $(N,A)$ is a (finite) directed graph, $s \in N$ is a designated “initial” node, and there is a path from node $s$ to every node $n \in N$.

- An entry node in a flow graph has no predecessors.
- An exit node in a flow graph has no successors.
- There is exactly one entry node, $s$. We can modify a general DAG to ensure this. How?
- In a control flow graph, any node unreachable from $s$ can be safely deleted. Why?
- Control flow graphs are usually sparse. I.e., $|A| = O(|N|)$. In fact, if only binary branching is allowed $|A| \leq 2 |N|$. 
Control Flow Graph (CFG)

**Basic Block** is a sequence of statements $S_1 \ldots S_n$ such that execution control must reach $S_1$ before $S_2$, and, if $S_1$ is executed, then $S_2 \ldots S_n$ are all executed in that order

- Unless a statement causes the program to halt

**Leader** is the first statement of a basic block

**Maximal Basic Block** is a basic block with a maximum number of statements ($n$)
Control Flow Graph (CFG)

**CFG** is a directed graph in which:
- Each node is a single basic block
- There is an edge $b_1 \rightarrow b_2$ if block $b_2$ may be executed after block $b_1$ in *some* execution

We define it typically for a single procedure

A CFG is a conservative approximation of the control flow! *Why?*
Example

**Source Code**

```c
unsigned fib(unsigned n) {
    int i;
    int f0 = 0, f1 = 1, f2;
    if (n <= 1) return n;
    for (i = 2; i <= n; i++) {
        f2 = f0 + f1;
        f0 = f1;
        f1 = f2;
    }
    return f2;
}
```

**LLVM bitcode**

```llvm
define i32 @fib(i32) {
    %2 = icmp ult i32 %0, 2
    br i1 %2, label %12, label %3
}

; <label>:3:
    br label %4

; <label>:4:
    %5 = phi i32 [ %8, %4 ], [ 1, %3 ]
    %6 = phi i32 [ %5, %4 ], [ 0, %3 ]
    %7 = phi i32 [ %9, %4 ], [ 2, %3 ]
    %8 = add i32 %5, %6
    %9 = add i32 %7, 1
    %10 = icmp ugt i32 %9, %0
    br i1 %10, label %11, label %4

; <label>:11:
    br label %12

; <label>:12:
    %13 = phi i32 [ %0, %1 ], [ %8, %11 ]
    ret i32 %13
}
Dominance in Flow Graphs

Let $d, d_1, d_2, d_3, n$ be nodes in $G$.

$d$ dominates $n$ ("$d$ dom $n$") iff every path in $G$ from $s$ to $n$ contains $d$

$d$ properly dominates $n$ if $d$ dominates $n$ and $d \neq n$

$d$ is the immediate dominator of $n$ ("$d$ idom $n$") if $d$ is the last proper dominator on any path from initial node to $n$,

$\text{DOM}(x)$ denotes the set of dominators of $x$. 
Dominator Properties

Lemma 1: \( \text{DOM}(s) = \{ s \} \).

Lemma 2: \( s \) dom \( d \), for all nodes \( d \) in \( G \).

Lemma 3: The dominance relation on nodes in a flow graph is a partial ordering

- Reflexive — \( n \) dom \( n \) is true for all \( n \).
- Antisymmetric — If \( d \) dom \( n \), then not \( n \) dom \( d \).
- Transitive — \( d_1 \) dom \( d_2 \) \( \land \) \( d_2 \) dom \( d_3 \) \( \Rightarrow \) \( d_1 \) dom \( d_3 \)

Lemma 4: The dominators of a node form a list.

Lemma 5: Every node except \( s \) has a unique immediate dominator.
Finding Dominators in a Flow Graph

Input: A flow graph \( G = (N,A,s) \).

Output: The sets \( \text{DOM}(\text{node}) \) for each node \( \in N \).

\[
\begin{align*}
\text{DOM}(s) & := \{ s \} \\
\text{forall} \ n \in N \setminus \{ s \} \ \text{do} \\
\quad \text{DOM}(n) & := N \\
\text{od} \\
\text{while} \ \text{changes to any DOM}(n) \ \text{occur} \ \text{do} \\
\quad \text{forall} \ n \ \text{in} \ N \setminus \{ s \} \ \text{do} \\
\quad \quad \text{DOM}(n) & := \{ n \} \cup \bigcap_{p \rightarrow n} \text{DOM}(p) \\
\quad \text{od} \\
\text{od}
\end{align*}
\]
Finding Dominators in a Flow Graph

Input: A flow graph $G = (N,A,s)$.

Output: The sets $\text{DOM}(\text{node})$ for each node $\in N$.

\begin{align*}
\text{DOM}(s) & := \{ s \} \\
\text{forall } n \in N - \{s\} \text{ do} & \\
\text{DOM}(n) & := N \\
\text{od} & \\
\text{while changes to any } \text{DOM}(n) \text{ occur do} & \\
\text{forall } n \text{ in } N - \{s\} \text{ do} & \\
\text{DOM}(n) & := \{n\} \bigcup \bigcap_{p \rightarrow n} \text{DOM}(p) \\
\text{od} & \\
\text{od} &
\end{align*}
Loops

while (b) { ... }  \Rightarrow  ?
Loops

The right definition of “loop” is not obvious.

Obviously bad definitions

- **Cycle:** Not necessarily properly nested or disjoint
- **Strongly Connected Components:** Too coarse; no nesting information

What properties of the loops do we want to extract from CFG?
Loops: Two Definitions

**Natural loop** — Defined using dominators

**Intervals** — Defined in terms of reachability in flow graph
Natural Loops

Def. **Back Edge:** An edge $n \rightarrow d$ where $d \text{ dom } n$

Def. **Natural Loop:** Given a back edge, $n \rightarrow d$, the natural loop corresponding to $n \rightarrow d$ is the set of nodes \{d + all nodes that can reach n without going through d\}

Def. **Loop Header:** A node $d$ that dominates all nodes in the loop

- Header is unique for each natural loop *Why?*
- Implies $d$ is the unique entry point into the loop
- Uniqueness is very useful for many optimizations
Natural Loops

Pros:
+ Intuitive, and similar to SCC.
+ Single entry point: “loop header”.
+ Identifies nested loops (if different headers)

Cons:
- Nested loops are not disjoint.
- Some nodes are not part of any natural loop.
- Does not include some cycles in “irreducible” flow graphs.
Reducibility of Flow Graphs

Def. **Reducible** flow graph: a flow graph $G$ is called reducible iff we can partition the edges into 2 disjoint sets:

- **forward edges**: should form a DAG in which every node is reachable from initial node $s$ (or also header)
- **remaining edges must be back edges**: i.e., only those edges $n \rightarrow d$ such that $d \text{ dom } n$

Idea:
Every “cycle” has at least one back edge
⇒ All “cycles” are natural loops
Otherwise graph is called irreducible.

*Well-structured*
Loops: Two Definitions

Natural loop — Defined using dominators

Intervals — Defined in terms of reachability in flow graph
Interval Analysis*

**Idea:** Partition flow graph into disjoint subgraphs so that each subgraph has a single entry (header).

**Definition:** The interval with node $h$ as header, denoted $I(h)$, is the subset of nodes of $G$ constructed as:

* It’s different from the interval analysis on numerical quantities
Transformation Rules T1 and T2

**T1**: Reduce a self-loop $x \rightarrow x$ to a single node

**T2**: If $x \rightarrow y$, and there is no other predecessor of $y$, then reduce $x$ and $y$ to a single node.

**Important**: If $G$ is reducible, successive applications of T1 and T2 produce the trivial graph.

$\Rightarrow$ Reducibility by T1 and T2 is equivalent to reducibility by intervals.
Node Splitting

Claim: If a node has \( n > 1 \) predecessors and \( m > 1 \) successors, split the node into \( n \) copies:
T2 is always applicable to a graph after a node is split

⇒ Any graph can be reduced to the trivial graph by applying T1, T2, and splitting.

Challenge: Finding a “minimal” splitting of a graph is not easy. Typically involves an NP-complete problem.
Interval Analysis*

Idea: Partition flow graph into disjoint subgraphs so that each subgraph has a single entry (header).

Definition: The interval with node $h$ as header, denoted $I(h)$, is the subset of nodes of $G$ constructed as:

$$I(h) := \{h\}$$

while $\exists$ node $m$ such that $m \notin I(h)$ and $m \neq s$ and all arcs entering $m$ leave nodes in $I(h)$
do
  $$I(h) := I(h) + m$$
od

* It's different from the interval analysis on numerical quantities
Derived Flow Graphs

Def. Derived Flow Graph, $I(g)$: If $G$ is a flow graph, then its $I(G)$ is:

(a) The nodes of $I(G)$ are the intervals of $G$
(b) The initial node of $I(G)$ is $I(s)$
(c) There is an arc from node $I(h)$ to $I(k)$ in $I(G)$ if there is any arc from a node in $I(h)$ to node $k$ in $G$.

Def. Derived sequence: the sequence $G = G_0, G_1, \ldots, G_k$ is derived iff

- $G_{i+1} = I(G_i)$ for $0 \leq i < k$,
- $G_{k-1} \neq G_k$,
- $I(G_k) = G_k$. $G_k$ is called the limit flow graph of $G$.

Definition: A flow graph is reducible iff its limit flow graph is a single node with no arc. Otherwise it is called irreducible.
Intervals Properties

Lemma 6. \( I(h) \) is unique: does not depend on order of node insertion. (See *Hecht* for proof)

Lemma 7. The subgraph generated by \( I(h) \) is itself a flow graph.

Lemma 8.
(a) Every arc entering a node of the interval \( I(h) \) from the outside enters the header \( h \).
(b) \( h \) dominates every node in \( I(h) \)
(c) every cycle in \( I(h) \) includes \( h \)
See You Next Time!

Review in the next few weeks:
Muchnick, Chapter 21: Case Studies of Compilers

Review by next class: Sections from Muchnick Sections §4.1-4.5, 4.9: Intermediate Representations
Section §7.1: Control Flow Graphs
(or equivalent sections in Cooper & Torczon or Aho, Lam, Sethi & Ullman)