

## Flow Graphs

*A fundamental representation for global optimizations.*

**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  $n \in N$ .

### Properties:

- 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.
- Control flow graphs are usually *sparse*. That is,  $|A| = O(|N|)$ . In fact, if only binary branching is allowed  $|A| \leq 2|N|$ .

## Control Flow Graph: CFG

### Definitions

**Basic Block**  $\equiv$  a sequence of statements (or instructions)  $S_1 \dots S_n$  such that execution control must reach  $S_1$  before  $S_{21}$ , and, if  $S_1$  is executed, then  $S_2 \dots S_n$  are all executed in that order (unless one of the statements causes the program to halt)

**Leader**  $\equiv$  the first statement of a basic block

**Maximal Basic Block**  $\equiv$  a *maximal-length* basic block

**CFG**  $\equiv$  a directed graph (usually for a single procedure) 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

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

**Homework:** Read Section 9.4 of *Aho, Sethi & Ullman*: algorithm to partition a procedure into basic blocks.

## Dominance in Flow Graphs

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

### Definitions

$d$  **dominates**  $n$  (write " $d \text{ dom } n$ ") 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  $n$  (write " $d \text{ idom } n$ ") if  $d$  is the last dominator on any path from initial node to  $n$ ,  $d \neq n$

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

### Properties

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

**Lemma 2:**  $s \text{ dom } d, \forall \text{ nodes } d \text{ in } G$ .

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

*Reflexive* :  $n \text{ dom } n$  is true  $\forall n$ .

*Antisymmetric* : If  $d \text{ dom } n$ , then  $n \text{ dom } d$  cannot hold.

*Transitive* :  $d_1 \text{ dom } d_2 \wedge d_2 \text{ dom } d_3 \implies d_1 \text{ dom } d_3$

## The Dominator Graph

**Lemma 4:** The dominators of a node form a chain.

**Proof** : Suppose  $x$  and  $y$  dominate node  $z$ . Then there is a path from  $s$  to  $z$  where both  $x$  and  $y$  appear only once (if not "cut and paste" the path). Assume that  $x$  appears first. Then if  $x$  does not dominate  $y$  there is a path from  $s$  to  $y$  that does not include  $x$  contradicting the assumption that  $x$  dominates  $z$ .

**Lemma 5** : Every node except  $s$  has a unique immediate dominator.

**Proof** : The dominators form a chain. The last node in the chain is the immediate dominator, and the last node always exists and is unique.

**Lemma 6** : Create the **dominator graph** for  $G$ :

- Same nodes in  $G$ .
- Edge  $n_1 \rightarrow n_2$  iff  $n_1 \text{ idom } n_2$ .

The dominator graph is a \_\_\_\_\_?

## Dominator Construction

**Algorithm DOM** : Finding Dominators in A Flow Graph

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

*Output* : The sets  $DOM(x)$  for each  $x \in N$ .

*Algorithm*:

```

DOM(s) := { s }
forall n in N - {s} do
  DOM(n) := N
od
while changes to any DOM(n) occur do
  forall n in N - {s} do
    DOM(n) := {n} ∪ ∩p→n DOM(p)
  od
od

```

## Identifying Program “Loops” in Control Flow Graphs

*The right definition of “loop” is not obvious.*

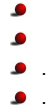
**Obviously bad definitions**

**Cycle**: Not necessarily properly nested or disjoint

**SCC**: Too coarse; no nesting information

**Properties of a Good Definition**

Q. What information about “loops” do we want to extract from CFG?



## Better Definitions

**Natural loop**: Defined using dominators

- + Intuitive, and similar to SCC.
- + Single entry point: “loop header”.
- + Identifies nested loops (if different headers)
- Nested loops are not disjoint.
- Some nodes are not part of any natural loop.
- Does not include some cycles in “irreducible” flow graphs.

**Intervals**: Defined in terms of reachability in flow graph

- + Single entry point: “loop header”.
- + Identifies nested loops
- + Nested loops are disjoint.
- + In reducible graphs, all nodes are part of some interval.
- Not an intuitive definition.
- Does not include some cycles in “irreducible” flow graphs.

## 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  $\{d + \text{all nodes that can reach } n \text{ without going through } d\}$

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

- Header is unique for each natural loop Why?
- $\Rightarrow d$  is the unique entry point into the loop
- Uniqueness is very useful for many optimizations

## Reducible and Irreducible Flow Graphs

Def. Reducible flow graph: *the two easier cases*  
 A flow graph  $G$  is called reducible iff we can partition the edges into 2 sets:

1. *forward edges*: should form a DAG in which every node is reachable from initial node
2. *other edges must be back edges*: i.e., only those edges  $n \rightarrow d$  where  $d \text{ dom } n$

Idea: Every "cycle" has at least one back edge

⇒ All "cycles" are natural loops

Otherwise graph is called irreducible.

*the difficult case*

## Intervals

### Idea

Partition flow graph into disjoint subgraphs where each subgraph has a single entry (header).

Intervals are closely connected to concept of reducibility (see later slides).

### Definition

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

```

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

**Lemma 7.**  $I(h)$  is unique: does not depend on order of node insertion.

**Proof:** See *Hecht*

## Properties of Intervals

**Lemma 8.** The subgraph generated by  $I(h)$  is itself a flow graph.

### Lemma 9.

- (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  $h$

### Proof.

- (a) Consider a node  $m \in I(h)$  that is also an entry node of  $I(h)$ . Then  $m$  could not have been added to  $I(h)$ .
- (b) Consider a node  $m \in I(h)$  not dominated by  $h$ . Then  $m$  could not have been added to  $I(h)$ .
- (c) Suppose there is a cycle in  $I(h)$  that does not include  $h$ . Then no node in the cycle could have been added to  $I(h)$ , because before any such node could be added the preceding node in the cycle would have to be added.

## Reducibility by Intervals

**Definition:** If  $G$  is a flow graph, then the derived flow graph of  $G$ ,  $I(G)$ , 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  $G$ .

**Definition:** The sequence  $G = G_0, G_1, \dots, G_k$  is called the derived sequence for  $G$  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.

## The T1 and T2 Transformations

---

**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.

*Example 1:*

*Example 2: Irreducible graph*

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

⇒ Reducibility by T1 and T2 is equivalent to reducibility by intervals.

## Properties of Irreducible Graphs

---

**The (\*) subgraph:**

*The lines represent edge-disjoint paths.*

**Lemma.** The absence of the (\*) subgraph in a flow graph is preserved by T1 and T2.

**Theorem.** A flow graph is irreducible *iff* it contains a (\*) subgraph.

## Node Splitting

---

If a node has  $n > 1$  predecessors and  $m > 1$  successors, split the node into  $n$  copies:

*Claim:* 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.