

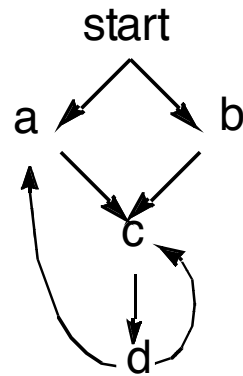


# Loop Invariant Computation and Code Motion

- I. Finding loops
- II. Loop-invariant computation
- III. Algorithm for code motion

## What is a Loop?

- **Goals:**
  - Define a loop in graph-theoretic terms (control flow graph)
  - Not sensitive to input syntax
  - A uniform treatment for all loops: DO, while, goto's
- **Not every cycle is a “loop” from an optimization perspective**

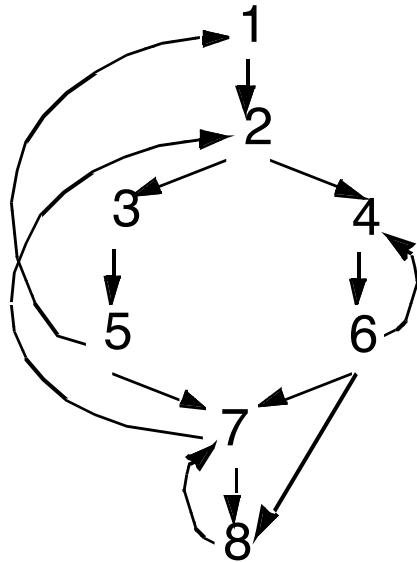


- **Intuitive properties of a loop**
  - single entry point
  - edges must form at least a cycle

## Formal Definitions

- **Dominators**

- Node  $d$  **dominates** node  $n$  in a graph ( $d \text{ dom } n$ ) if every path from the start node to  $n$  goes through  $d$



- Dominators can be organized as a **tree**
  - $a \rightarrow b$  in the **dominator tree** iff  $a$  immediately dominates  $b$

## Natural Loops

- **Definitions**

- Single entry-point: *header*
  - a header *dominates all nodes in the loop*
- A *back edge* is an arc whose *head dominates its tail* (tail -> head)
  - a back edge *must be a part of at least one loop*
- The *natural loop of a back edge* is the *smallest set* of nodes that *includes the head and tail of the back edge*, and has *no predecessors outside the set*, except for the predecessors of the header.

## Algorithm to Find Natural Loops

1. Find the dominator relations in a flow graph
2. Identify the back edges
3. Find the natural loop associated with the back edge

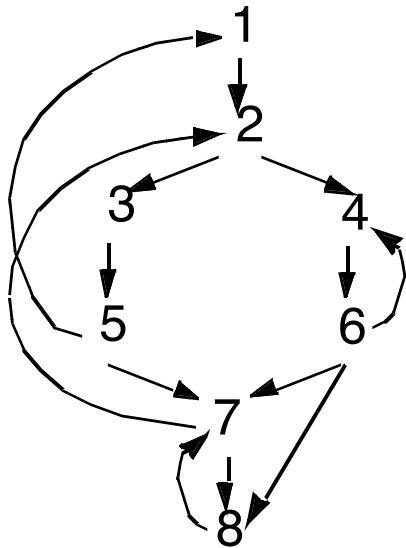
# 1. Finding Dominators

- **Definition**
  - Node  $d$  dominates node  $n$  in a graph ( $d \text{ dom } n$ ) if every path from the start node to  $n$  goes through  $d$
- **Formulated as MOP problem:**
  - node  $d$  lies on all possible paths reaching node  $n \Rightarrow d \text{ dom } n$ 
    - Direction:
    - Values:
    - Meet operator:
    - Top:
    - Bottom:
    - Boundary condition: start/entry node =
    - Initialization for internal nodes
    - Finite descending chain?
    - Transfer function:
- **Speed:**
  - With reverse postorder, most flow graphs (reducible flow graphs) converge in 1 pass

## 2. Finding Back Edges

- **Depth-first spanning tree**

- Edges traversed in a depth-first search of the flow graph form a depth-first spanning tree



- **Categorizing edges in graph**

- **Advancing** edges: from ancestor to proper descendant
- **Cross** edges: from right to left
- **Retreating** edges: from descendant to ancestor (not necessarily proper)

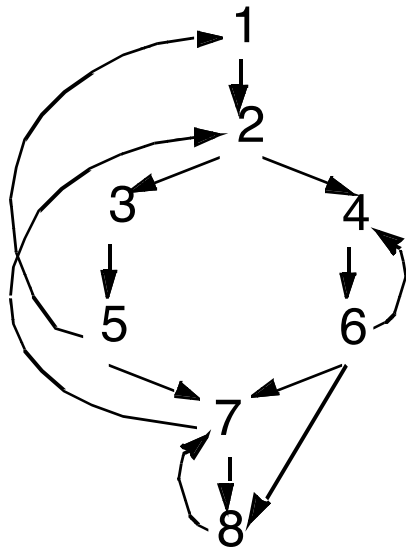
## Back Edges

- **Definition**
  - **Back edge**:  $t \rightarrow h$ ,  $h$  dominates  $t$
- **Relationships between graph edges and back edges**
- **Algorithm**
  - Perform a depth first search
  - For each retreating edge  $t \rightarrow h$ , check if  $h$  is in  $t$ 's dominator list
- **Most programs (all structured code, and most GOTO programs) have **reducible flow graphs****
  - retreating edges = back edges



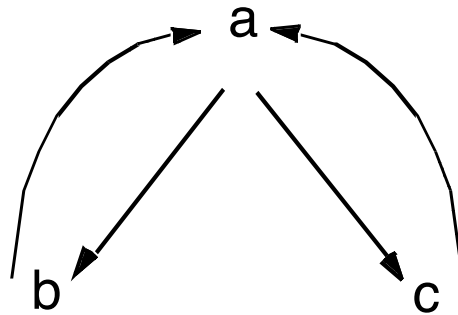
### 3. Constructing Natural Loops

- The **natural loop of a back edge** is the smallest set of nodes that includes the head and tail of the back edge, and has no predecessors outside the set, except for the predecessors of the header.
- **Algorithm**
  - delete  $h$  from the flow graph
  - find those nodes that can reach  $t$   
(those nodes plus  $h$  form the natural loop of  $t \rightarrow h$ )



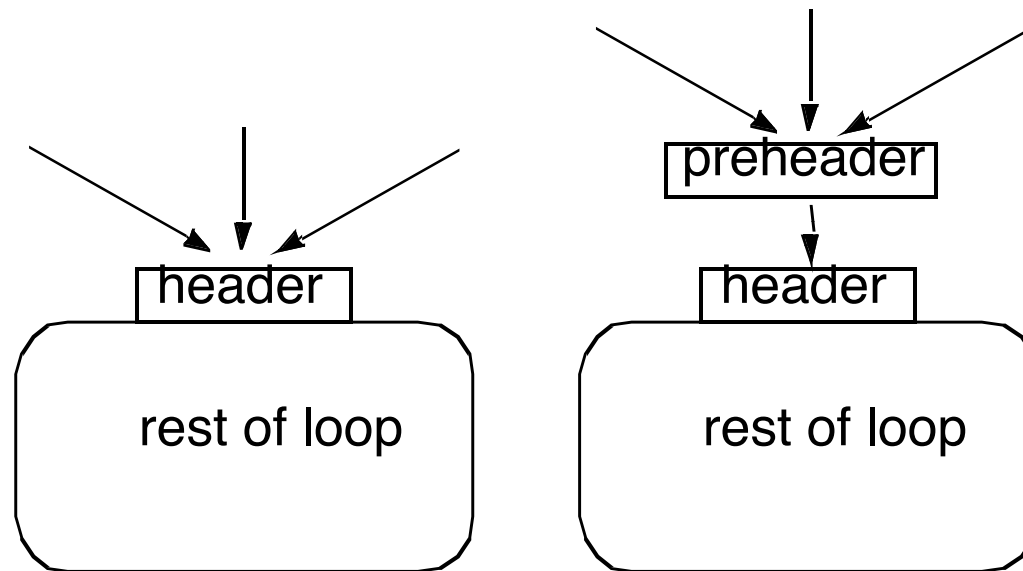
## Inner Loops

- **If two loops do not have the same header:**
  - they are either disjoint, or
  - one is entirely contained (nested within) the other
    - inner loop: one that contains no other loop.
- **If two loops share the same header:**
  - Hard to tell which is the inner loop
  - Combine as one



## Preheader

- Optimizations often require code to be executed once before the loop
- Create a preheader basic block for every loop

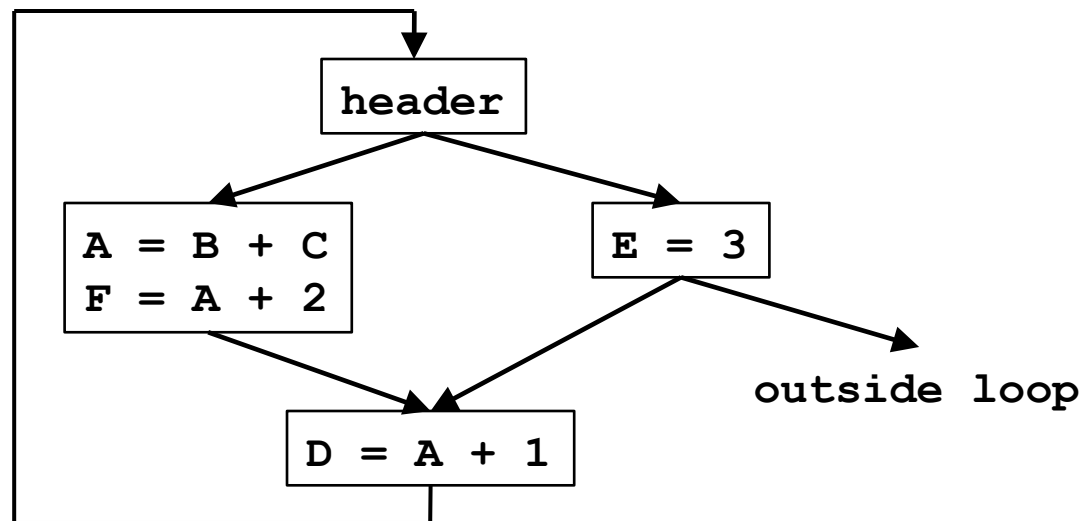


## Finding Loops: Summary

- **Define loops in graph theoretic terms**
- **Definitions and algorithms for:**
  - Dominators
  - Back edges
  - Natural loops

## II. Loop-Invariant Computation and Code Motion

- **A loop-invariant computation:**
  - a computation whose value does not change as long as control stays within the loop
- **Code motion:**
  - to move a statement within a loop to the preheader of the loop



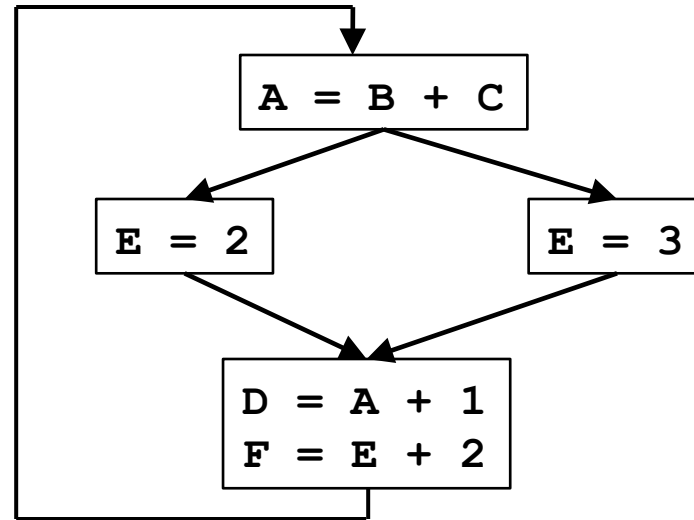
# Algorithm

- **Observations**
  - Loop invariant
    - operands are defined outside loop or invariant themselves
  - Code motion
    - not all loop invariant instructions can be moved to preheader
- **Algorithm**
  - Find invariant expressions
  - Conditions for code motion
  - Code transformation

## Detecting Loop Invariant Computation

- Compute reaching definitions
  - Mark INVARIANT if all the definitions of B and C that reach a statement  $A=B+C$  are outside the loop
    - constant B, C?
  - Repeat: Mark INVARIANT if
    - all reaching definitions of B are outside the loop, or
    - there is exactly one reaching definition for B, and it is from a loop-invariant statement inside the loop
    - similarly for C
- until no changes to set of loop-invariant statements occur.

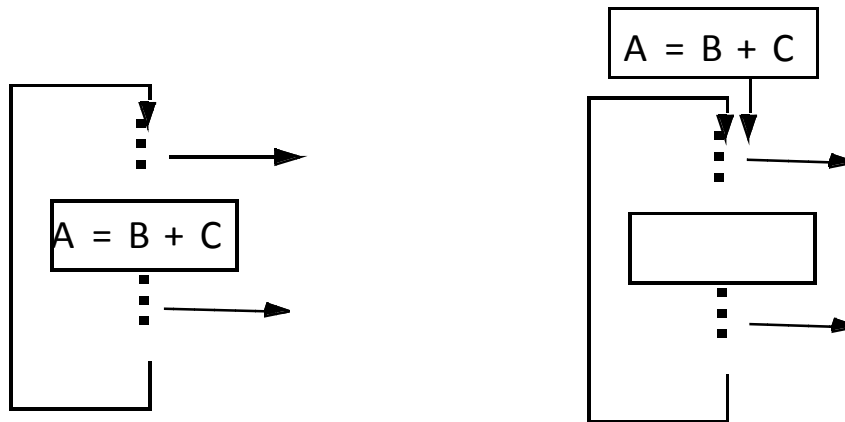
## Example





### III. Conditions for Code Motion

- **Correctness:** Movement does not change semantics of program
- **Performance:** Code is not slowed down



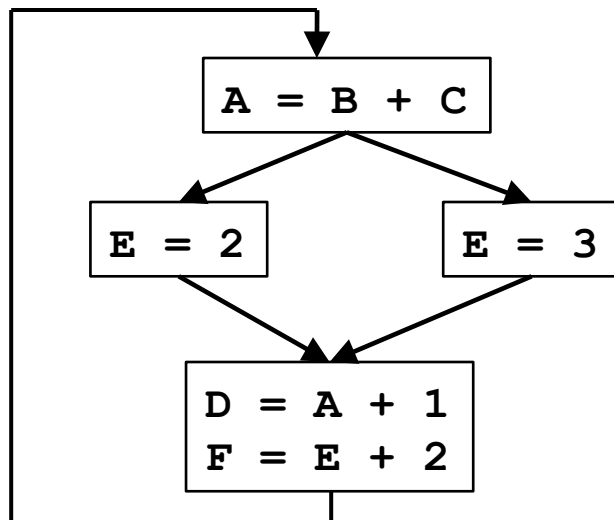
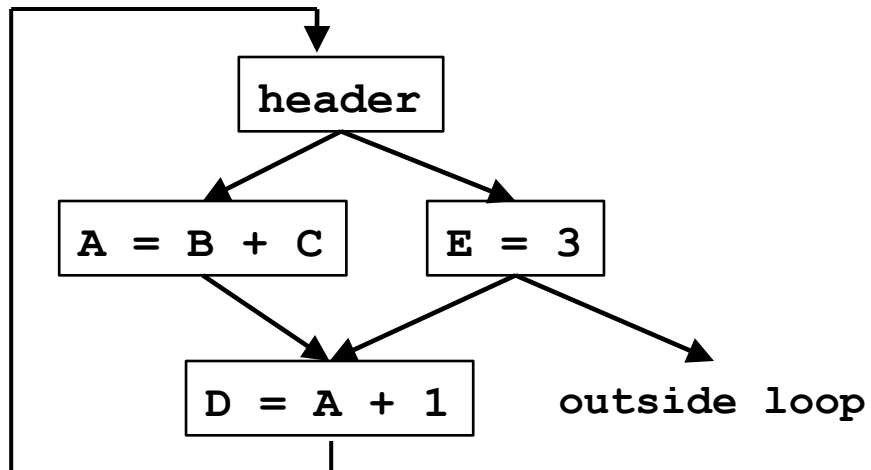
- **Basic idea:** defines once and for all
  - control flow:
  - other definitions:
  - other uses:

## Code Motion Algorithm

Given: a set of nodes in a loop

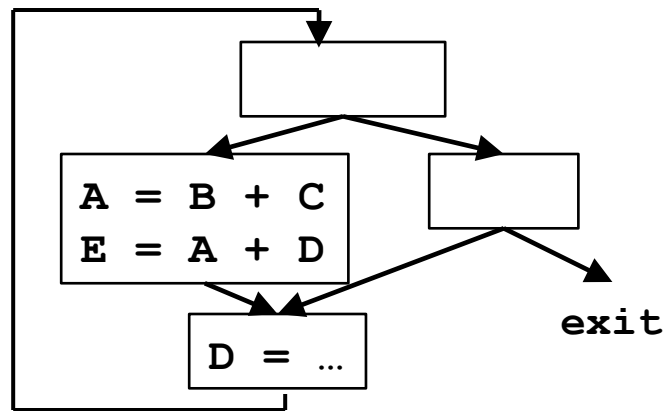
- **Compute reaching definitions**
- **Compute loop invariant computation**
- **Compute dominators**
- **Find the exits of the loop (i.e. nodes with successor outside loop)**
- **Candidate statement for code motion:**
  - loop invariant
  - in blocks that dominate all the exits of the loop
  - assign to variable not assigned to elsewhere in the loop
  - in blocks that dominate all blocks in the loop that use the variable assigned
- **Perform a depth-first search of the blocks**
  - Move candidate to preheader if all the invariant operations it depends upon have been moved

## Examples



## More Aggressive Optimizations

- **Gamble on: most loops get executed**
  - Can we relax constraint of dominating all exits?



- **Landing pads**

```
While p do s    →    if p {
                    preheader
                    repeat
                      s
                    until not p;
                  }
```

## Summary

- **Precise definition and algorithm for loop invariant computation**
- **Precise algorithm for code motion**
- **Use of reaching definitions and dominators in optimizations**