

CS 536

Optimization Frameworks

Homework

- Homework 9 is due tomorrow
- Will post HW10 and P6 today

Final Exam

- Final exam will cover *all* material covered in course – i.e., all compiler phases
 - Focus will be on after-midterm material
- 2hr exam, similar in format to midterm

Other

- No class on Thursday
- We have time for non-standard material
 - Anything you'd like to know about?
- Next week, we'll do review

Roadmap

- Last time:
 - Optimization overview
 - Soundness and completeness
 - Simple optimizations
 - Peephole
 - LICM
- This time:
 - More Optimization
 - Analysis frameworks

Outline

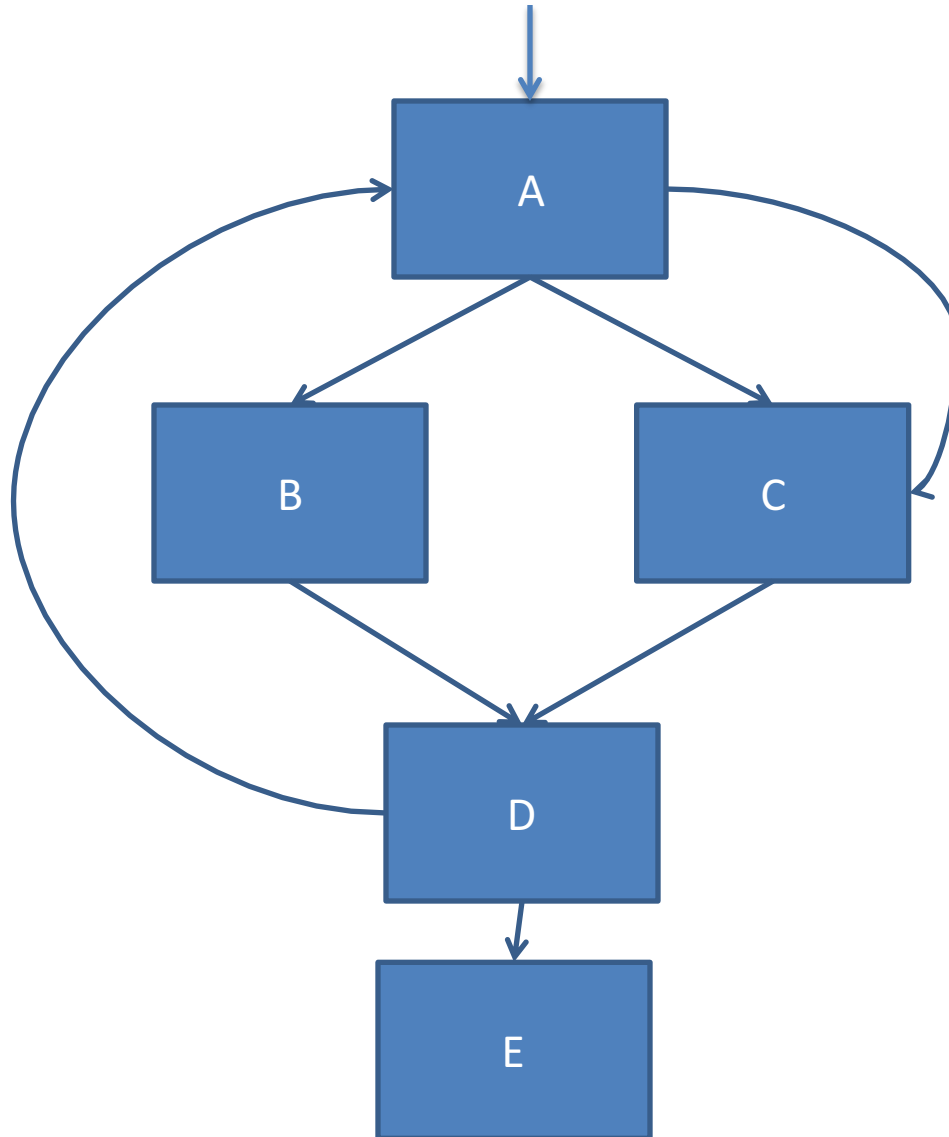
- Review Dominators
- Introduce some more advanced concepts
 - SSA
 - Dataflow propagation

Dominator Review

Dominator Terms

- Domination (A dominates B):
 - to reach block B, you must have gone through block A
- Strict Domination (A strictly dominates B)
 - A dominates B and A is not B
- Immediate Domination (A immediately dominates B)
 - A immediately dominates B if A dominates B and has no intervening dominators

Dominator Example



Dominance Frontier

- Definition: For a block X , the set of nodes Y such that X dominates an immediate predecessor of Y but does not strictly dominate Y



Static Single Assignment

SSA: Key Idea

- We'd like to build an intermediate representation of the program in which each variable gets a value in at most **1 program point**:



```
x = 1  
x = 2  
y = 3
```



```
x = 1  
z = 2  
y = 3
```



```
x = y  
z = y  
w = z
```



```
i = 0;  
while( i < 10){  
    k = i + 1;  
}
```

Conversion

- We'll make new variables to carry over the effect of the original program



```
x = 1  
x = x  
y = x
```



```
x1 = 1  
x2 = x1  
y1 = x2
```

Benefits

- There are some obvious advantages to this format for program analysis
 - Easy to see the *live range* of a variable at a program point (i.e. the definitions that could reach that point)
 - Places with the variable on the LHS
 - Easy to see when a variable is *dead* (i.e. unused)
 - Never used on the RHS of an statement

Optimizations that SSA Helps

- Dead Code Elimination

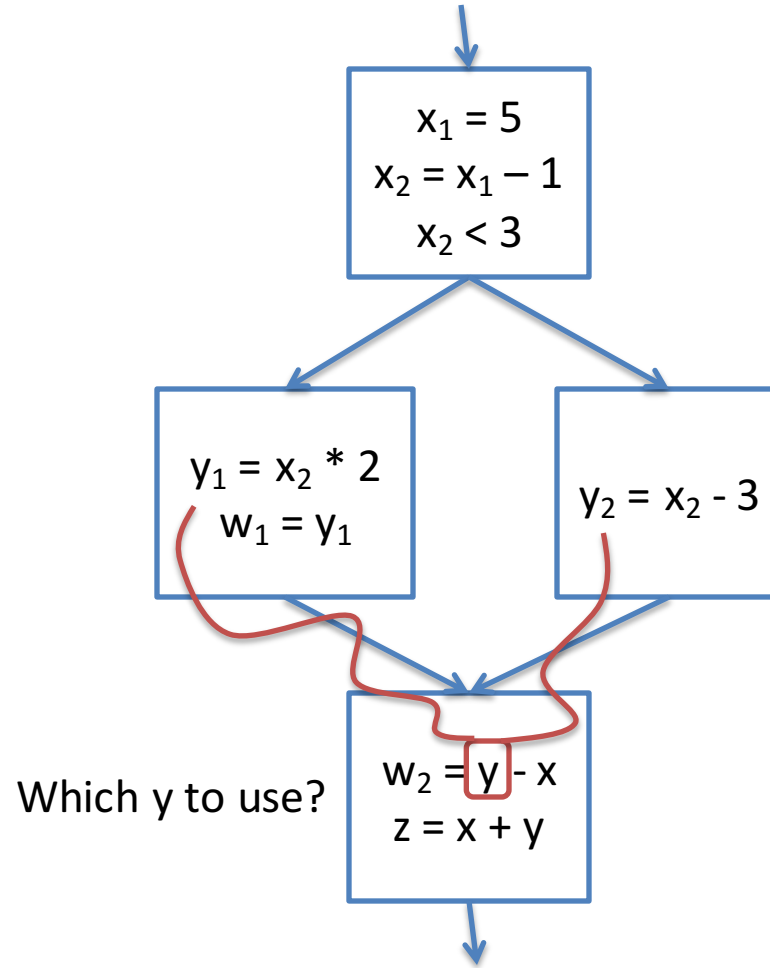
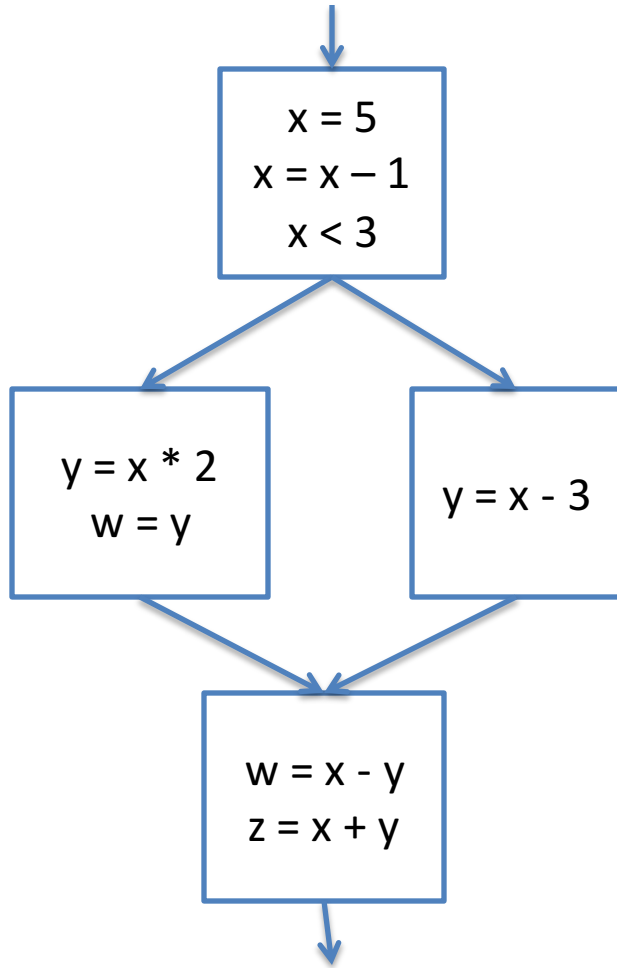
```
int a = 9;  
int b = 2;  
  
if ( g < 12){  
    a = 1;  
} else {  
    if (b < 4){  
        a = 2;  
    } else {  
        a = 3;  
    }  
}  
  
b = a;  
return 2;
```

Optimizations that SSA Helps

- Constant propagation / constant folding

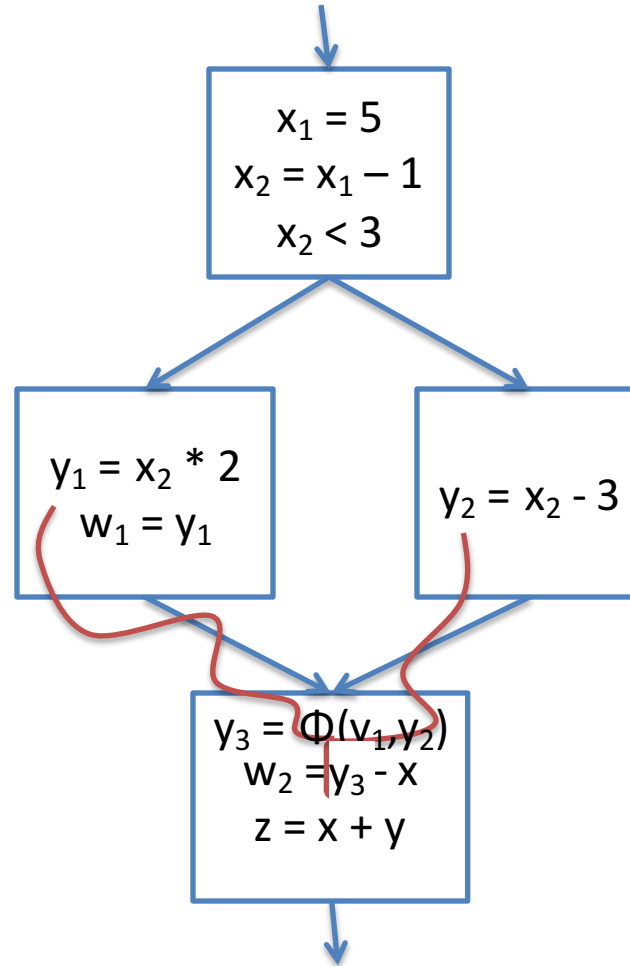
```
int a = 30;           6  
int b = 3; - (a / 5);  
int c;  
c = b * 4; true 12  
if (c > 10) {  
    c = 2; - 10; 2  
}  
return 4; - 260 4 a);
```


What about Conditionals?



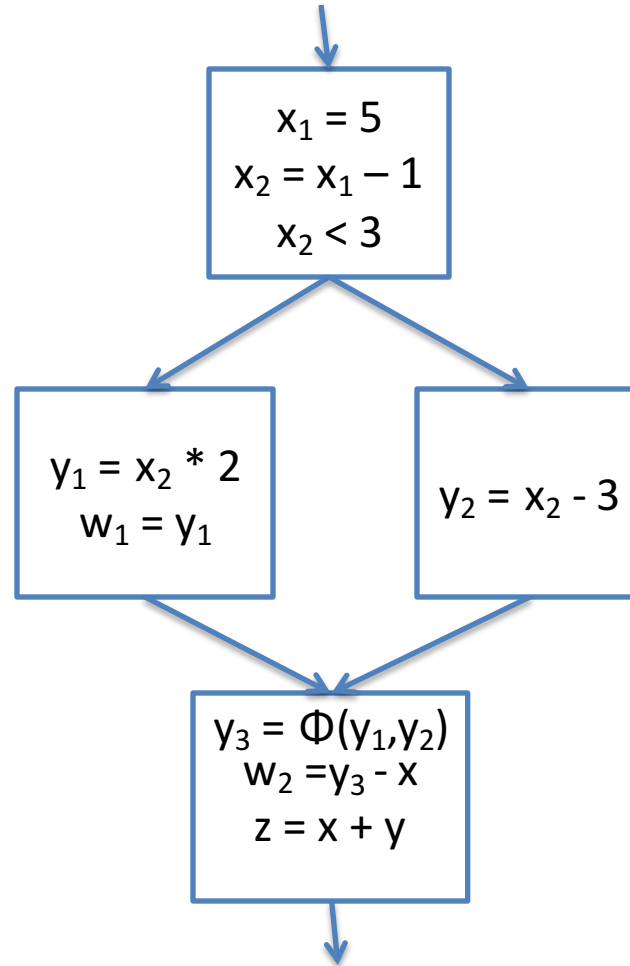
Phi Functions

- We'll introduce a special symbol Φ that represents a join like this
- Takes arguments of all variables to join
- Returns the “correct” one
- Do we need a Φ for x ?
 - Nope!



Computing Phi Functions

- Intuitively, we want to figure out cases where there are multiple assignments that can reach a node
- To be safe, we can place Φ functions for each assignment at every node in the ***dominance frontier***



Pruned Phi Functions

- This causes a bunch of useless Φ functions
 - Cases where the result is never used
- ***Pruned SSA*** is a version where statically removable Φ nodes are never used

Dataflow Frameworks

Dataflow Framework Idea

- Many analyses can be formulated as how data is transformed over the control flow graph
 - Propagate static information from:
 - the beginning of a single basic block
 - the end of a single basic block
 - The join points of multiple basic blocks

Dataflow Framework Idea

- Meet Lattice
- Transfer function
 - How data is propagated from one end of a basic block to the other
- Meet operation
 - Means of combining lattice between blocks

Dataflow Analysis Direction

- Forward analysis
 - Start at the beginning of a function's CFG, work along the control edges
- Backwards analysis
 - Start at the end of a function's CFG, work against the control edges
- Continuously propagate values until there is no change

Dataflow Example 1

- Available Expression analysis
 - Whether an expression that has been previously computed may be reused
 - Forward dataflow problem: from expression to points of re-use
 - Meet Lattice:


True



False

- Meet operation:
 - AND of all predecessors
- At the beginning of each block, everything is True
- * **This causes some problems for loops**

Dataflow Example 2

- Very Busy Expression analysis
 - An expression is very busy at a point p if it is guaranteed that it will be computed at some time in the future
 - Backwards dataflow problem: from computation to use
 - Meet Lattice:

A vertical line representing a lattice. The word "True" is at the top and "False" is at the bottom, connected by a vertical line.
 - Meet operation: AND

The END... or is it?

- Covered a broad range of topics
 - Some formal concepts
 - Some practical concepts
- What we skipped
 - Linking and loading
 - Interpreters
 - Register allocation
 - Performance analysis / Proofs