Lecture 6

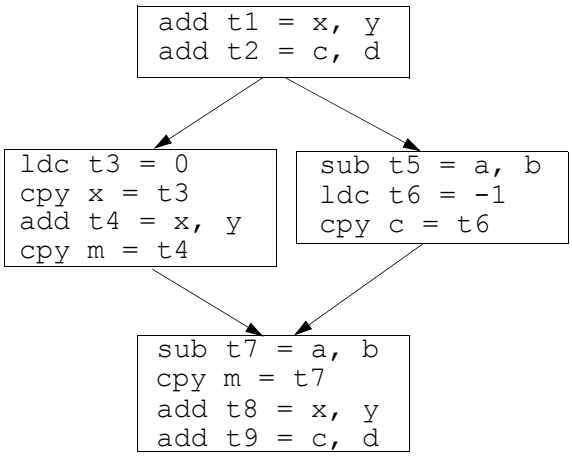
Global Common Subexpression Elimination

I Available Expressions Analysis

II Eliminating CSEs

Reference: Muchnick 13.1

Global Common Subexpression



- **Availability of an expression E at point P**
  - DEFINITION: Along every path to P in the flow graph:
    - E must be evaluated at least once
    - no variables in E redefined after the last evaluation
  - Observations: E may have different values on different paths

Formulating the Problem

- **Domain:**
  - a bit vector,
    a bit for each **textually unique** expression in the program
- **Forward or Backward?**
- **Lattice Elements?**
- **Meet Operator?**
  - check: commutative, idempotent, associative
- **Partial Ordering**

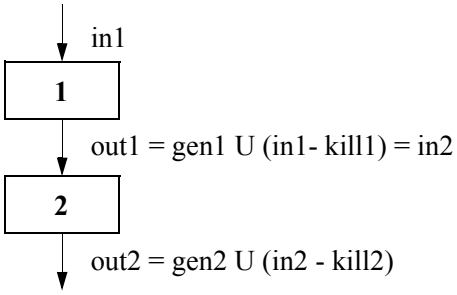
- **Top?**
- **Bottom?**
- **Boundary condition: entry/exit node?**
- **Initialization for iterative algorithm?**

Transfer Functions

- **Can use the same equation as reaching definitions**
  - out[b] = gen[b] U (in[b] - kill[b])
- **Start with the transfer function for a single instruction**
  - When does the instruction generate an expression?
  - When does it kill an expression?
- **Calculate transfer functions for complete basic blocks**
  - Compose individual instruction transfer functions

Composing Transfer Functions

- **Derive the transfer function for an entire block**



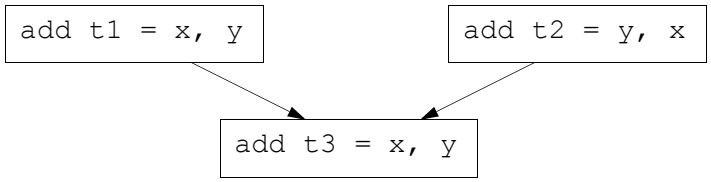
- **Since out1 = in2 we can simplify:**
  - out2 = gen2 U ((gen1 U (in1 - kill1)) - kill2)
    out2 = gen2 U (gen1 - kill2) U (in1 - (kill1 U kill2))
    out2 = gen2 U (gen1 - kill2) U (in1 - (kill2 U (kill1 - gen2)))
- **Result**
  - gen = gen2 U (gen1 - kill2)
  - kill = kill2 U (kill1 - gen2)

II. Eliminating CSEs

- **Available expressions (across basic blocks)**
  - provides the set of expressions available at the start of a block
- **Value Numbering (within basic block)**
  - Initialize Values table with available expressions
- **If CSE is an "available expression" => transform the code**
  - Original destination may be:
    - a temporary register
    - overwritten
    - different from the variables on other paths
  - A solution: Copy the expression to a new variable at each evaluation reaching the redundant use
  - Textbook discusses another solution

III. Limitation: Textually Identical Expression

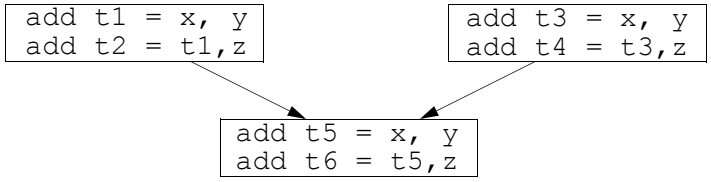
- **Commutative operations**



  - sort the operands

Further Improvements

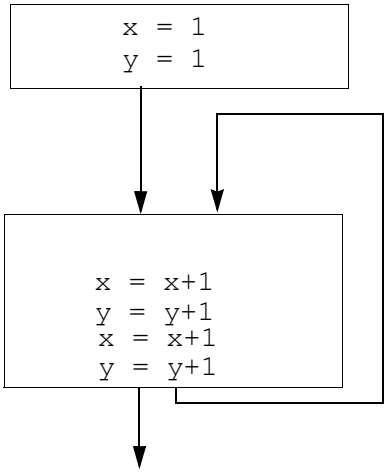
- **Examples**
  - Expressions with more than two operands



  - Textually different expressions may be equivalent

```
add   t1 = x, y
beq   t1, t2, L1
cpy   z = x
add   t3 = z, y
```

Another Example



Summary

	Reaching Definitions	Available Expressions
Domain	Sets of definitions	Sets of expressions
Transfer function $f_b(x)$ Generate $\cup$ Propagate		
direction of function	forward: out[b] = f_b(in[b])	forward: out[b] = f_b(in[b])
Generate	Gen_b: exposed definitions	Gen_b: exprs. evaluated
Propagate	x - $Kill_b$: killed definitions	x - $Kill_b$: exprs. killed
Merge operation	$\cup$ (in[b]=$\cup$ out[predecessors])	$\cap$(in[b]=$\cap$out[predecessors])
Initialization	out[entry] = $\varnothing$	out[entry] = $\varnothing$
	out[b] = $\varnothing$	out[b] = all expressions

Other Optimizations

Constant Propagation:

Copy Propagation:

Dead Code Elimination: