CS293S
SSA & Dead Code Elimination

Yufei Ding

Review of Last Class

- Static Single Assignment(SSA)
  - Maximal SSA (all variables in every joint block)
  - Minimal SSA
    - Dominance Frontier (DF)
    - (a def in block n results in an insertion in each of its DF(n))
  - Semi-pruned SSA
    - (similar as minimal SSA, but on only global variable)
  - Pruned SSA
    - (similar as semi-pruned SSA, but dead are removed)

Focus of This Class

- Dead code elimination
- Techniques for Removing φ-functions

Dead Code Elimination

- Useful statements
  - Output statements (e.g., printf)
  - Statements that compute values used by useful statements

- Algorithm to eliminate dead code
  - Start with absolutely useful statements
  - Repeatedly adds statements that compute variables used in current useful statements
    - through def-use chains (reaching definitions)

Dead-code Elimination

- Using def-use chain (review):

```
a ← 5
b ← 3
c ← b + 2
d ← a - 2
```

```
e ← a + b
e ← e + c
```

```
e ← 13
```

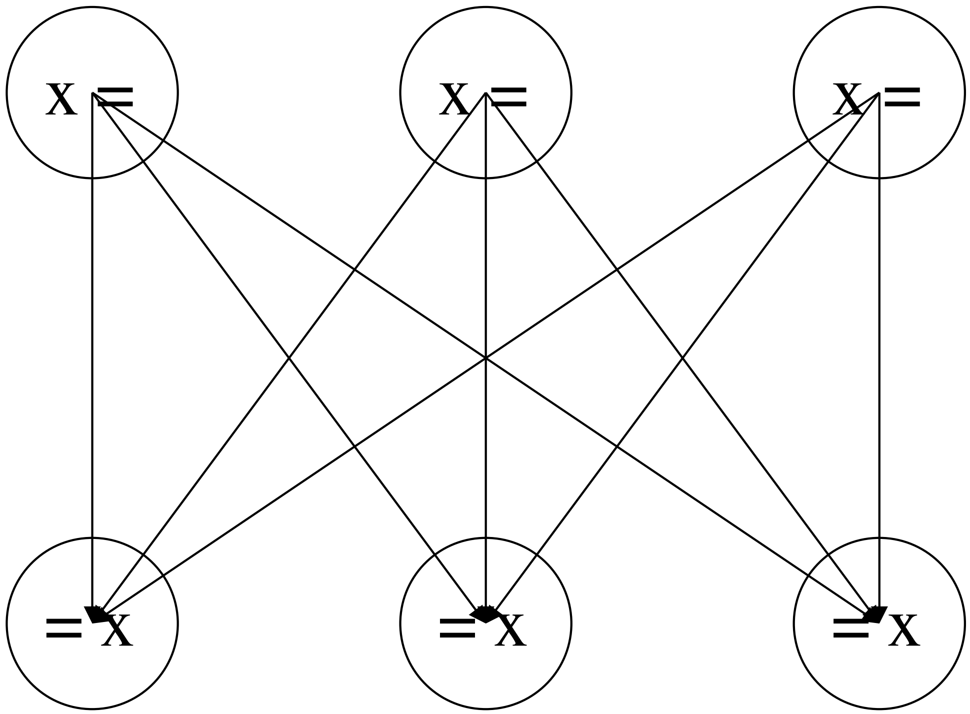
```
f ← 2 + e
Write f
```

`d` is dead

It has no use

Def-use w/o SSA form

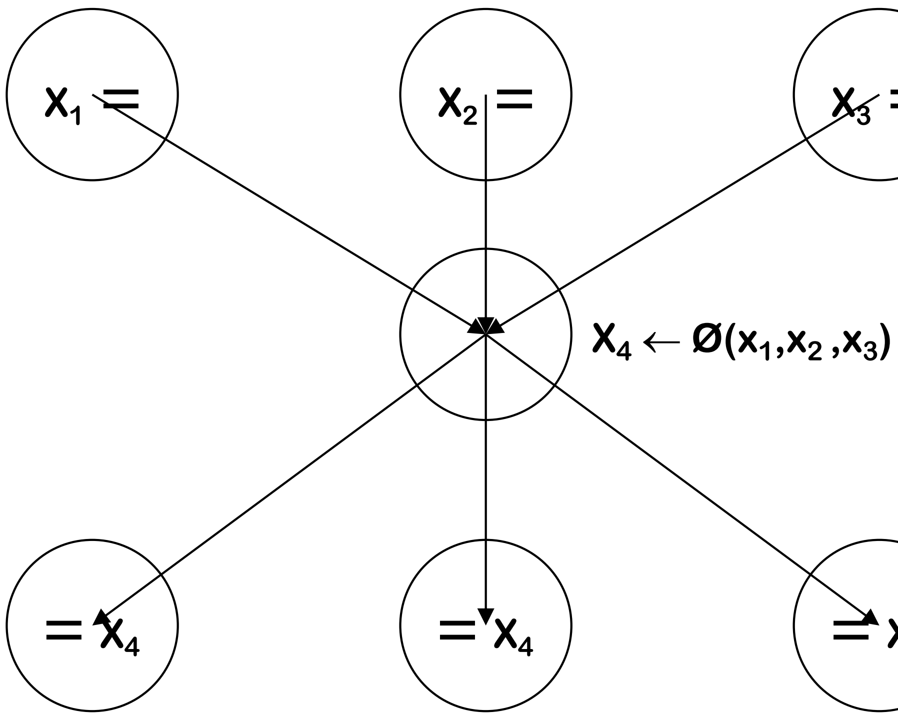
- Def-use edges grow very large



caused by branches

Def-use with SSA Form

- Edges reduced from 9 to 6



Example

```
if (x > 0) {
        printf("greater than zero");
}
```

- The printf statement (I/O statement) is inherently live. You also need to mark the "if (x>0)" live because the 'print' statement is control dependent on the 'if'.

Post-dominator Relation

- If X appears on every path from START to Y, then X dominates Y.
- If X appears on every path from Y to END, then X postdominates Y.
- Postdominator Tree
  - END is the root
  - Any node Y other than END has ipdom(Y) as its parent
  - Parent, child, ancestor, descendant

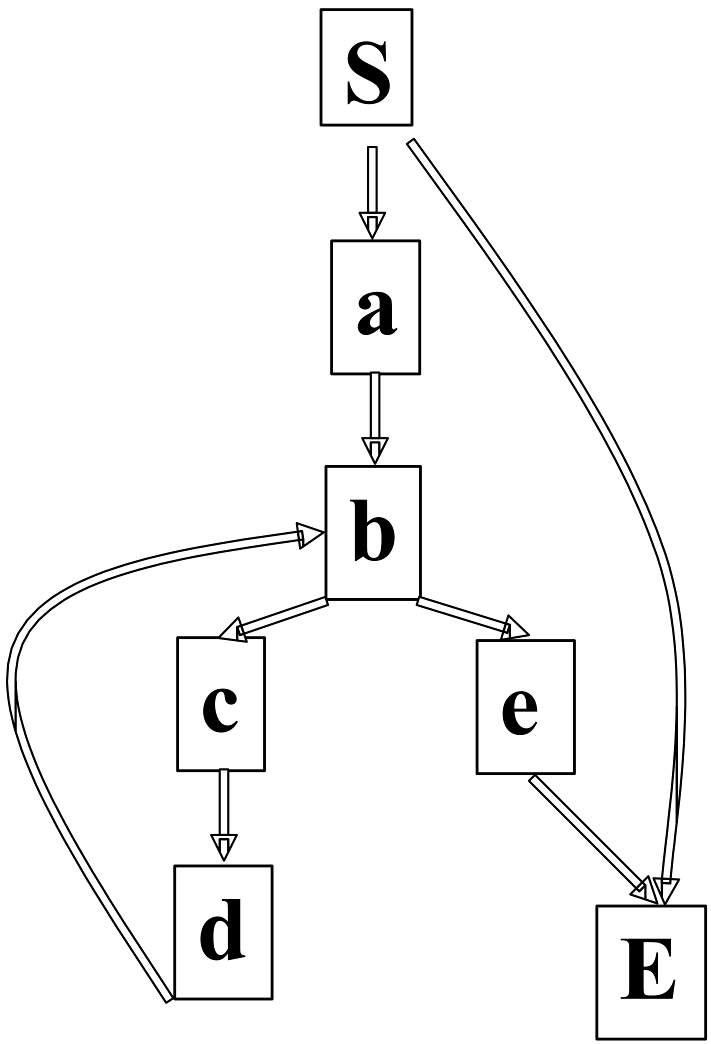
Control Dependence

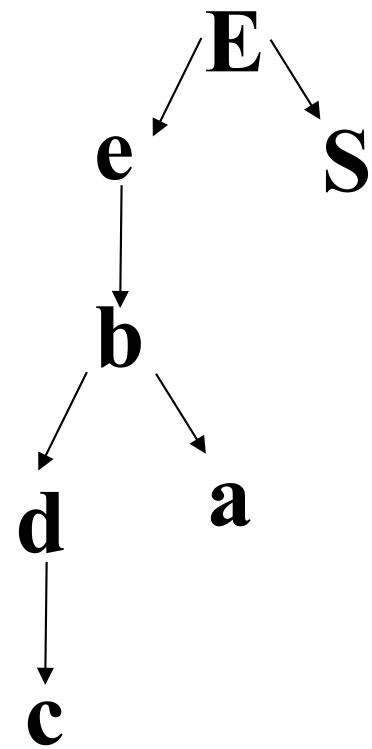
- There are two possible definitions.

- Node w is control dependent on edge ($u \rightarrow v$) if
  - w postdominates v
  - If $w \neq u$, w does not postdominate u

- Node w is control dependent on node u if there exists an edge $u \rightarrow v$
  - w postdominates v
  - If $w \neq u$, w does not postdominate u

Example





Pdom Tree

Control Dep Relation

	a	b	c	d
S->a	√	√		
b->c		√	√	√

Control Dependence V.S. Dominator Frontier

- Reverse control flow graph (RCFG)
- Let X and Y be nodes in CFG. X in DF(Y) in CFG iff Y is control dependent on X in RCFG.
- DF(Y) in CFG = conds(Y) in RCFG, where conds(Y) is the set of nodes that Y is control dependent on.

Control Dependence V.S. Dominator Frontier

- Forward direction:
  - By definition of "dominator Frontier", X in DF(Y) in CFG, if Y dominates V (i.e., one of X's parents in CFG), but Y does not strictly dominates X in CFG.
    - If there is an edge V → X in CFG and Y dominates V in CFG
    - If there is an edge X → V in RCFG and Y postdominates V in RCFG
    - Y does not strictly dominate X in CFG
    - If Y ≠ X, Y does not dominate X in CFG
    - If Y ≠ X, Y does not postdominate X in RCFG

  - If there is an edge X → V in RCFG, Y postdominates V in RCFG, If Y ≠ X, Y does not postdominate X in RCFG
  - By definition of "control dependent", we could know that Y is control dependent on X in RCFG.

Control Dependence V.S. Dominator Frontier

- Backward direction:
  - By definition of "control dependent", Y is control dependent on X in RCFG, if there exists an edge $X \rightarrow V$ in RCFG, Y postdominates V in RCFG, If $Y \neq X$, Y does not postdominate X in RCFG.
    - If there is an edge $V \rightarrow X$ in CFG, i.e., V is X's parent in CFG
    - Y dominates V (i.e., X's parent) in CFG
    - If $Y \neq X$, Y does not dominate X in CFG
    - Y does not strictly dominate X in CFG
  - Y dominates V (i.e., one of X's parents in CFG), but Y does not strictly dominates X in CFG
  - By definition of DF, we could know that X is in DF(Y).

Dead Code Elimination

```
Mark
   for each op i
      clear i's mark
      if i is critical then
         mark i
         add i to WorkList

   while (Worklist ≠ Ø)
      remove i from WorkList
         (i has form "x←y op z" )
      if def(y) is not marked then
         mark def(y)
         add def(y) to WorkList
      if def(z) is not marked then
         mark def(z)
         add def(z) to WorkList

      for each b ∈ RDF(block(i))
         mark the block-ending
            branch in b
         add it to WorkList
```

```
Sweep
   for each op i
      if i is not marked then

         if i is a branch then
            rewrite with a jump to
               i's nearest useful
               post-dominator

         if i is not a jump then
            delete i
```

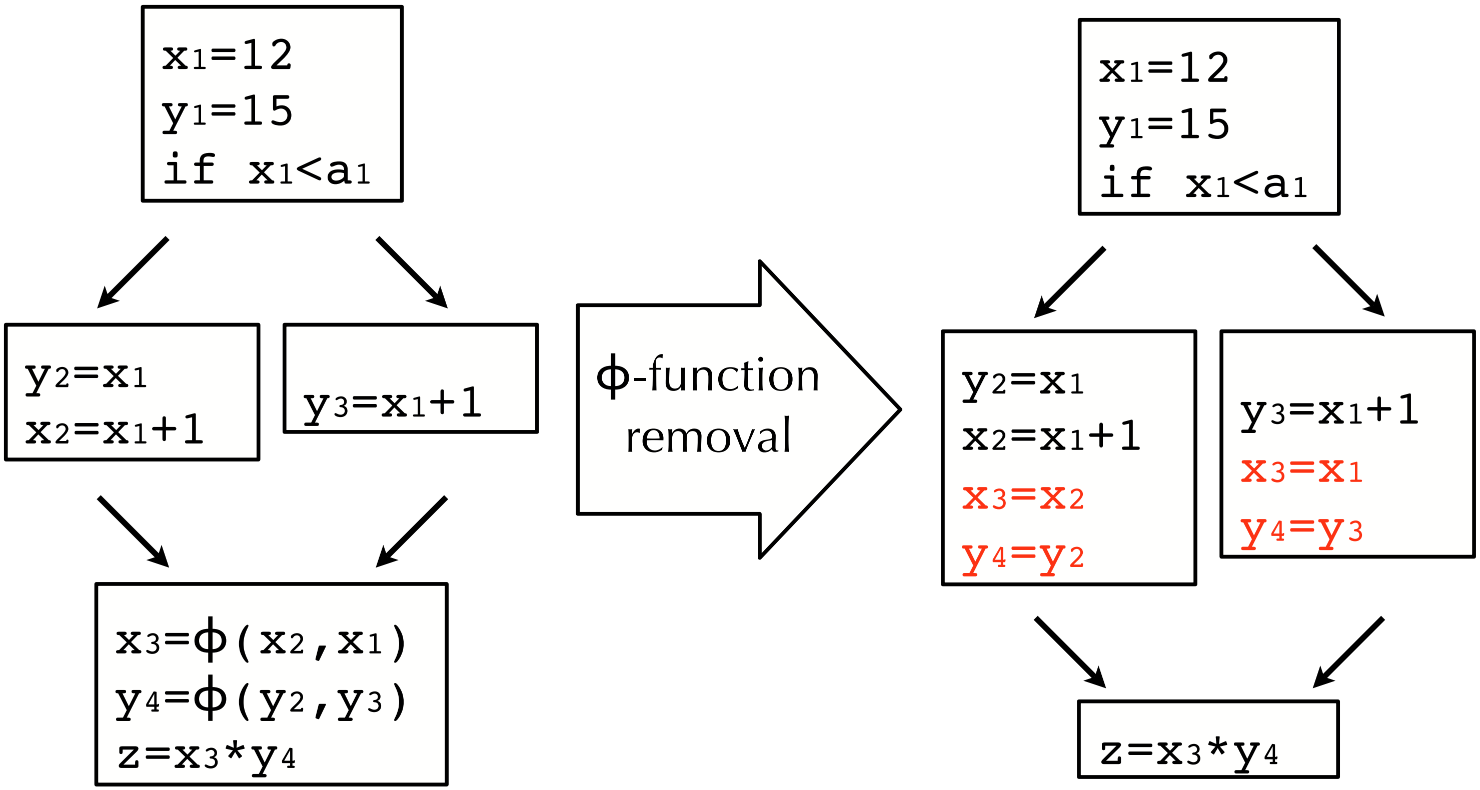
Notes:

- Eliminates some branches
- Reconnects dead branches to the remaining live code

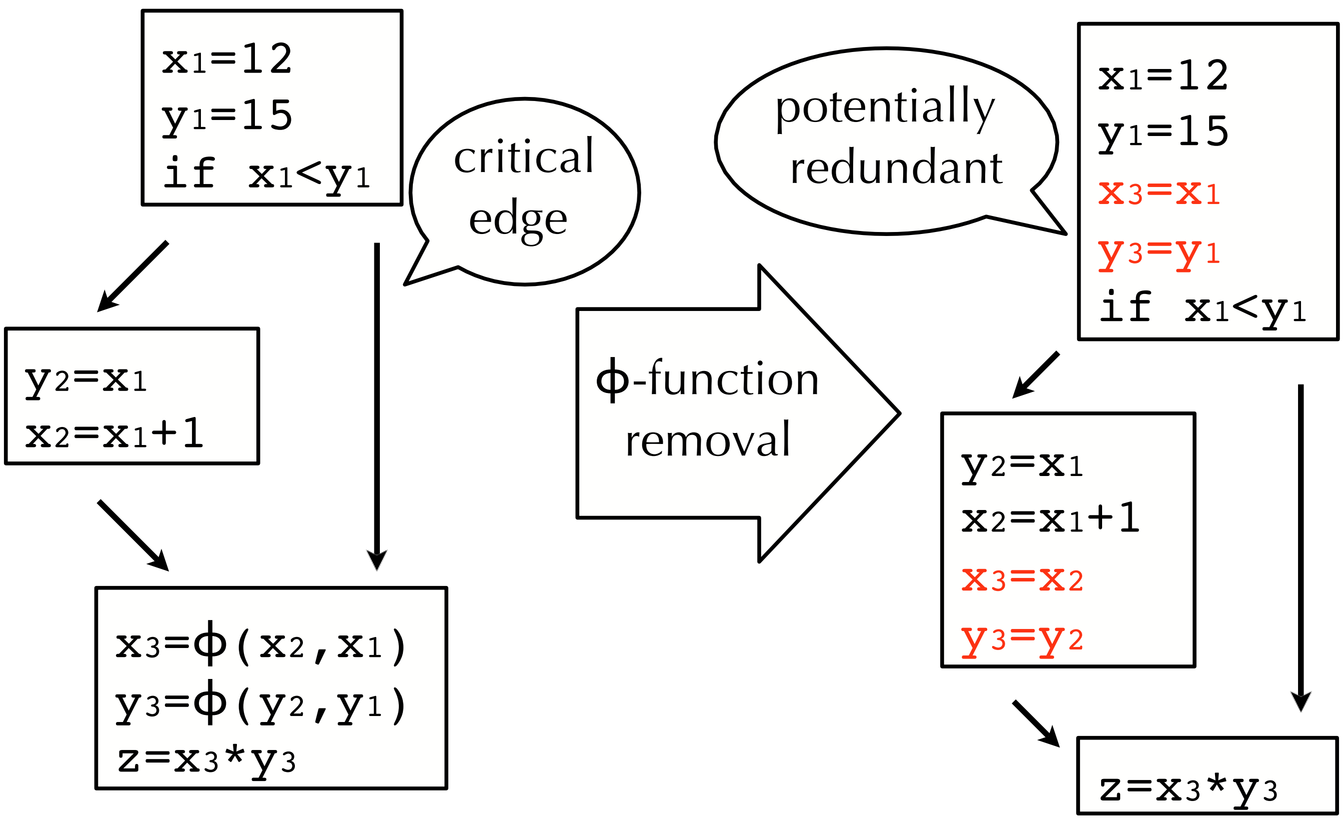
Removing ϕ-functions

- After the program has been turned into SSA form and the various optimizations performed on that representation, it must be transformed into executable form.
- This implies in particular that φ-functions must be removed, as they cannot be implemented on standard machines.

Removing ϕ-functions



Potential redundancy with critical edge



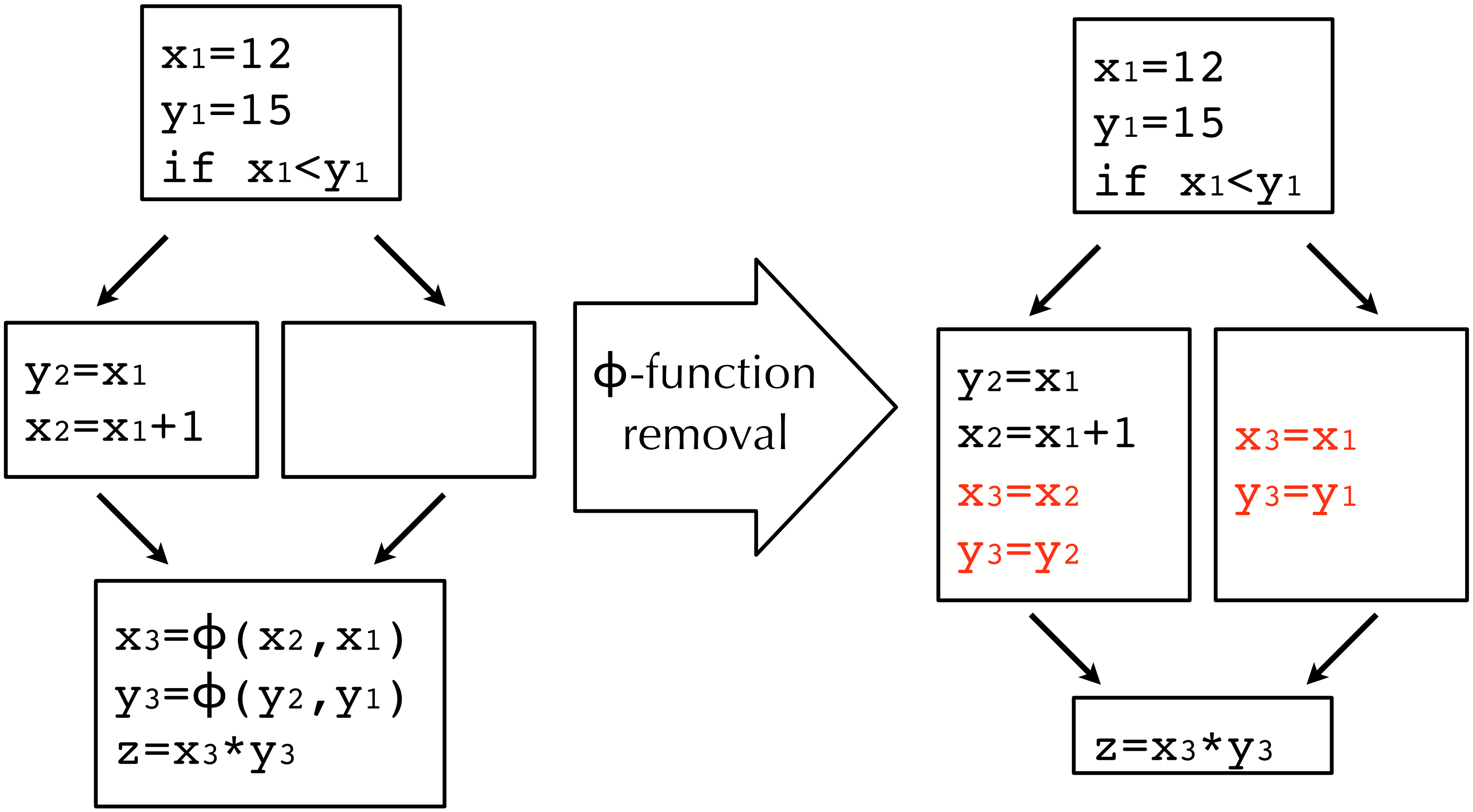
Critical edges

- CFG edges that go from a node with multiple successors to a node with multiple predecessors are called critical edges.

- While removing φ-functions, the presence of a critical edge from n_1 to n_2 leads to the insertion of redundant *move instructions* in n_1, corresponding to the φ-functions of n_2.

- Ideally, they should be executed only if control reaches n_2 later, but this is not certain when n_1 executes.

With edge splitting



Summary

Dead code elimination algorithm.

Important concepts of control dependence

- postdominator, reverse dominance frontier
- Relations between control dependence and dominance relations