# CS738: Advanced Compiler Optimizations

## SSA Continued

Amey Karkare
karkare@cse.iitk.ac.in

http://www.cse.iitk.ac.in/~karkare/cs738
Department of CSE, IIT Kanpur

## Agenda

- Properties of SSA
- SSA to Executable
- SSA for Optimizations

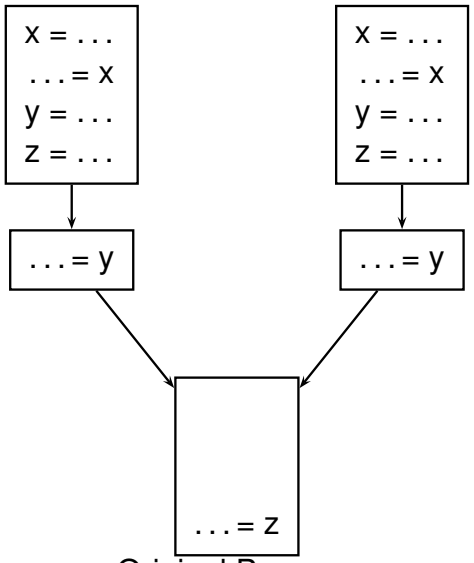
## Complexity of Construction

- $R = \max(N, E, A, M)$
- $N$: nodes, $E$: edges in flow graph
- $A$: number of assignments
- $M$: number of uses of variables
- Computation of DF: $O(R^2)$
- Computation of SSA: $O(R^3)$
- In practice, worst case is rare.
- Practical complexity: $O(R)$

## Linear Time Algorithm for $\phi$-functions

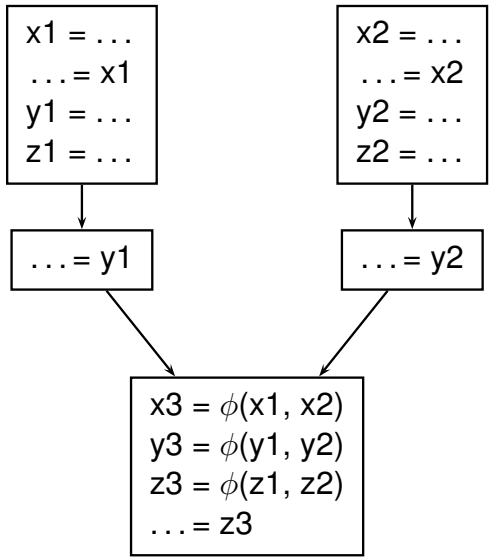
- By Sreedhar and Gao, in POPL'95
- Uses a new data structure called DJ-graph
- Linear time is achieved by careful ordering of nodes in the DJ-graph
- DF for a node is computed only once an reused later if required.

## Variants of SSA Form: Simple Example




Original Program

## Variants of SSA Form: Simple Example




Minimal SSA form

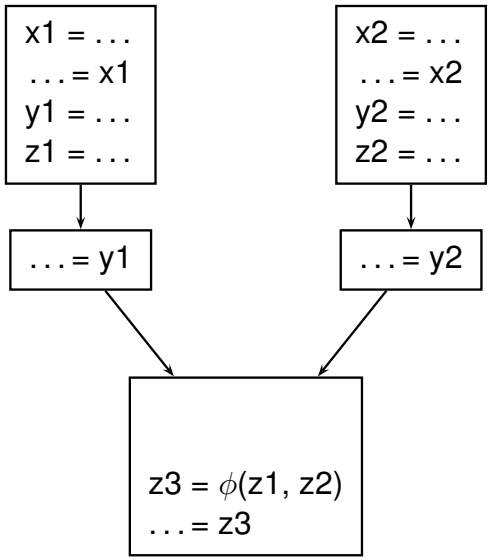
## Variants of SSA Form

- Minimal SSA still contains extraneous $\phi$-functions
  - Inserts some $\phi$-functions where they are dead
  - Would like to avoid inserting them
- Pruned SSA
- Semi-Pruned SSA

## Pruned SSA

- Only insert $\phi$-functions where their value is live
- Inserts fewer $\phi$-functions
- Costs more to do
- Requires global Live variable analysis

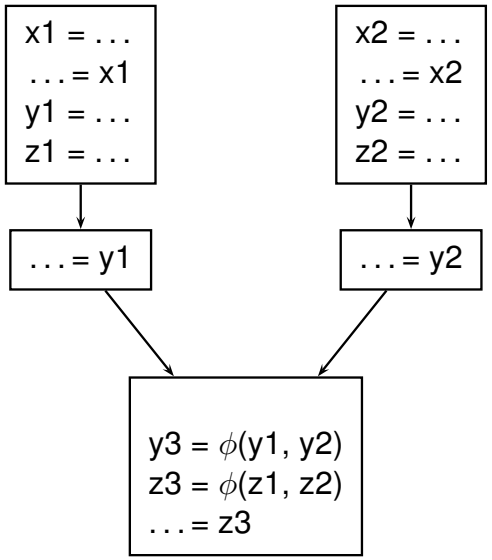
## Variants of SSA Form: Pruned SSA Example




## Semi-Pruned SSA Form

- Discard names used in only one block
- Total number of $\phi$-functions between minimal and pruned SSA
- Needs only local Live information
- Non-locals can be computed without iteration or elimination

## Variants of SSA Form: Semi-pruned SSA Example



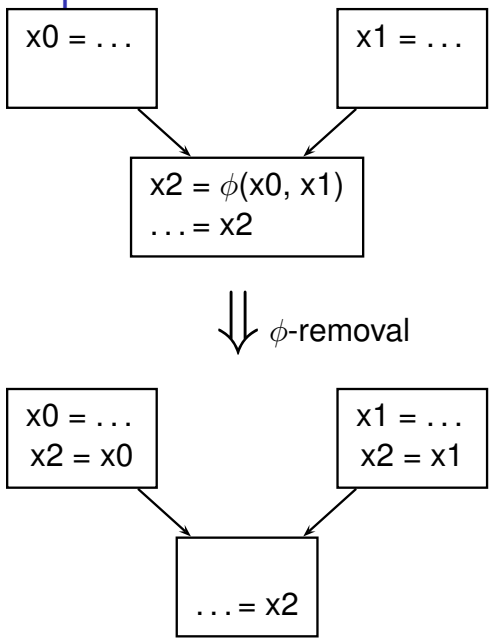

## Computing Non-locals

```
foreach block B {
   defined = {}
   foreach instruction v = x op y {
      if x not in defined
          non-locals = non-locals ∪ {x}
      if y not in defined
          non-locals = non-locals ∪ {y}
      defined = defined ∪ {v}
   }
}
```

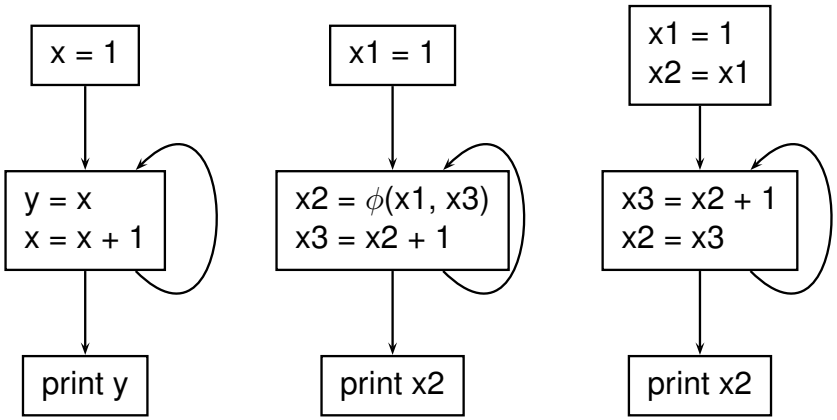
## SSA to Executable

- At some point, we need executable code
  - Need to fix up the $\phi$-function
- Basic idea
  - Insert copies in predecessors to mimic $\phi$-function
  - Simple algorithm
    - Works in most cases, but not always
  - Adds lots of copies
    - Many of them will be optimized by later passes

## $\phi$-removal: Example




## Lost Copy Problem

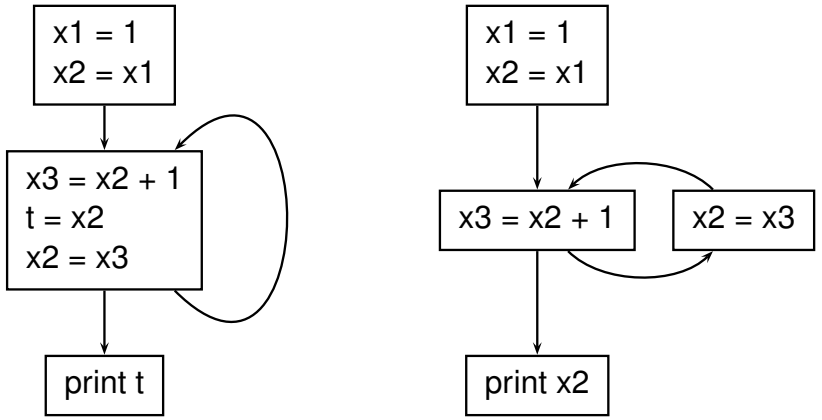



Program

SSA from with copy propagation

After $\phi$-removal

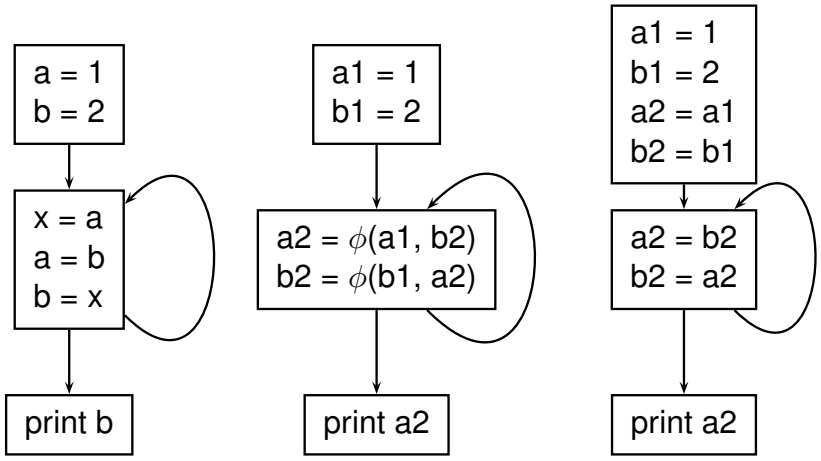
## Lost Copy Problem: Solutions




1. Use of Temporary

2. Critical Edge Split

## Swap Problem




Program

SSA form with copy propagation

After $\phi$-removal

## Swap Problem: Solution

- Fix requires compiler to detect and break dependency from output of one $\phi$-function to input of another $\phi$-function.
- May require temporary if cyclic dependency exists.

## SSA Form for Optimizations

- SSA form can improve and/or speed up many analyses and optimizations
  - (Conditional) Constant propagation
  - Dead code elimination
  - Value numbering
  - PRE
  - Loop Invariant Code Motion
  - Strength Reduction