

## Reaching Definitions

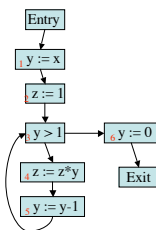
Done many ways

## How many ways can we do Reaching Definitions?

- Data Flow analysis
- Constraint based approach
- Abstract Interpretation
- Type and Effect Systems

## Running Example

```
y := x; // 1
z := 1; // 2
while y > 1 do { // 3
  z := z*y; // 4
  y := y-1 }; // 5
y := 0 // 6
```



## Data Flow Analysis

- $RD_{exit}(1) = RD_{entry}(1) - \{ (y, l) \mid l \text{ in Lab} \} \sqcup \{ (y, 1) \}$
- $RD_{exit}(2) = RD_{entry}(2) - \{ (z, l) \mid l \text{ in Lab} \} \sqcup \{ (z, 2) \}$
- ...
- $RD_{entry}(1) = \{ (x, ?), (y, ?), (z, ?) \}$
- $RD_{entry}(2) = RD_{exit}(1)$
- $RD_{entry}(3) = RD_{exit}(2) \sqcup RD_{exit}(5)$
- ...

## Finding a solution

- Let  $F$  be this set of equations
- Find  $RD$  s.t.  $RD = F(RD)$ 
  - find least such solution
  - what is a non-minimal solution?

## Constraint based approach

- $RD_{\text{exit}}(1) \supseteq RD_{\text{entry}}(1) - \{ (y, l) \mid l \text{ in Lab} \}$
- $RD_{\text{exit}}(1) \supseteq \{ (y, 1) \}$
- $RD_{\text{exit}}(2) \supseteq RD_{\text{entry}}(2) - \{ (z, l) \mid l \text{ in Lab} \}$
- $RD_{\text{exit}}(2) \supseteq \{ (z, 2) \}$
- ...
- $RD_{\text{entry}}(1) \supseteq \{ (x, ?), (y, ?), (z, ?) \}$
- $RD_{\text{entry}}(2) \supseteq RD_{\text{exit}}(1)$
- $RD_{\text{entry}}(3) \supseteq RD_{\text{exit}}(2)$
- $RD_{\text{entry}}(3) \supseteq RD_{\text{exit}}(5)$
- ...

## Something different

- $[[ \text{fn } x \Rightarrow [x]^1 ]^2 [ \text{fn } y \Rightarrow [y]^3 ]^4 ]^5$
- For each function application, which function may be applied?

- $C(l)$  = values that subexpression  $l$  can evaluate to
- $p(x)$  = values that  $x$  can be bound to
- $[[ \text{fn } x \Rightarrow [x]^1 ]^2 [ \text{fn } y \Rightarrow [y]^3 ]^4 ]^5$
- $\{ \text{fn } x \Rightarrow [x]^1 \} \sqsubseteq C(2)$
- $\{ \text{fn } y \Rightarrow [y]^3 \} \sqsubseteq C(4)$

## Values taken on by a variable use

- $[[ \text{fn } x \Rightarrow [x]^1 ]^2 [ \text{fn } y \Rightarrow [y]^3 ]^4 ]^5$
- $p(x) \sqsubseteq C(1)$
- $p(y) \sqsubseteq C(3)$

## Results of function application

- $[[ \text{fn } x \Rightarrow [x]^1 ]^2 [ \text{fn } y \Rightarrow [y]^3 ]^4 ]^5$
- Only one function application
  - but we don't know what function will be applied
- If function might be  $\text{fn } x \Rightarrow [x]^1$ 
  - $\{ \text{fn } x \Rightarrow [x]^1 \} \sqsubseteq C(2) \sqsubseteq C(4) \sqsubseteq p(x)$
  - $\{ \text{fn } x \Rightarrow [x]^1 \} \sqsubseteq C(2) \sqsubseteq C(1) \sqsubseteq C(5)$
- If function might be  $\text{fn } y \Rightarrow [y]^3$ 
  - $\{ \text{fn } y \Rightarrow [y]^3 \} \sqsubseteq C(2) \sqsubseteq C(4) \sqsubseteq p(y)$
  - $\{ \text{fn } y \Rightarrow [y]^3 \} \sqsubseteq C(2) \sqsubseteq C(3) \sqsubseteq C(5)$

## Least solution

- $[[ \text{fn } x \Rightarrow [x]^1 ]^2 [ \text{fn } y \Rightarrow [y]^3 ]^4 ]^5$
- $C(1) = \{ \text{fn } y \Rightarrow [y]^3 \}$
- $C(2) = \{ \text{fn } x \Rightarrow [x]^1 \}$
- $C(3) = \{ \}$
- $C(4) = \{ \text{fn } y \Rightarrow [y]^3 \}$
- $C(5) = \{ \text{fn } y \Rightarrow [y]^3 \}$
- $p(x) = \{ \text{fn } y \Rightarrow [y]^3 \}$
- $p(y) = \{ \}$

## Abstract Interpretation

- A trace is a series of (label, variable) pairs indicating the sequence of assignments
- A collecting semantics records the set of traces  $tr$  that can reach a given program point
- $RD(tr)(x) = l$  iff the rightmost pair  $(x, l')$  in  $tr$  has  $l = l'$

## Static Single Assignment Form

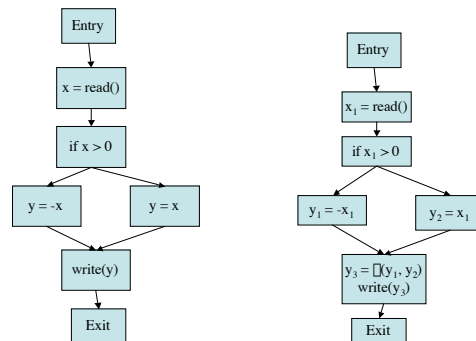
## SSA Form

- Each variable is assigned to once
- Each use of a variable has exactly one reaching definition
- Makes a number of analysis techniques simpler and more effective

## SSA of straight line code

Original	In SSA form
$x = 1$	$x_1 = 1$
$y = x+1$	$y_1 = x_1+1$
$x = x+1$	$x_2 = x_1+1$
$z = x+y$	$z_1 = x_2+y_1$

## SSA with branches



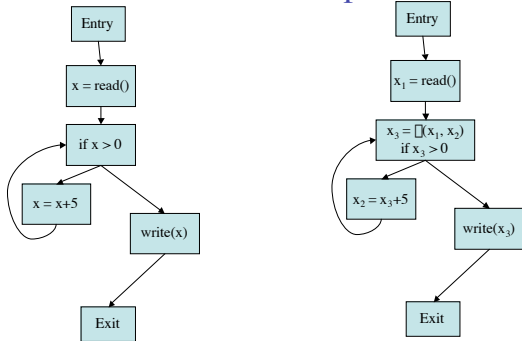
## Phi functions

- Used when merging two values
- Phi functions are easy to handle in analysis
  - not so easy to execute
- To generate executable code, need to eliminate the phi functions

## Why SSA?

- More efficient
  - avoids some quadratic cases:
    - programs with M defs and N uses can require  $M * N$  def-use chains
    - for almost all realistic programs, SSA is linear in size of program
- Makes algorithms more efficient and accurate
- Unrelated uses of the same variable name are irrelevant

## SSA with loops

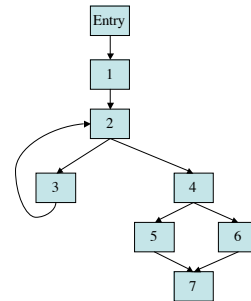


## Where do we place Phi functions?

- An assignment to x in node B generates a new assignment/phi function for x in the dominance frontier of B
  - which may, in turn, introduce additional assignments/phi functions for x

## Dominance Frontier

- C is in the dominance frontier of B iff
  - Exists a path from B to Exit through C
  - such that C is the first node not strictly dominated by B
- Equivalently:
  - C is the first node where a path from B to Exit and a path from Entry to Exit (not going through B) meet
- Equivalently:
  - B dominates a predecessor of C
  - B does not strictly dominate C



DF(1) = ?  
 DF(2) = ?  
 DF(3) = ?  
 DF(4) = ?  
 DF(5) = ?  
 DF(6) = ?  
 DF(7) = ?

## Computing Dominance Frontier

- Do in postorder on dominator tree:
  - $DF(B) = Succ(B) - Sdom(B)$
  - foreach block C s.t.  $idom(C) = B$  do
  - $DF(B) \sqcup = DF(C) - Sdom(B)$
- Equivalently:
  - $DF(B) = Succ(B)$
  - foreach block C s.t.  $idom(C) = B$  do
  - $DF(B) \sqcup = DF(C)$
  - $DF(B) -= Sdom(B)$

## phi function placement

