How many definitions and uses are here?

What are the possible values of x in its two uses?

How can we improve a program if we have this info?

Def-Use Chains
Reaching Definition

- Reach(n) definition (how large can a set be?)
- Local information
- Data flow equation
- Initialization
- Convergence

Dead Code Elimination
Which statements are useless (dead)?
What are the def-use chain?
Can we infer the dead code automatically?
What is the cost of the algorithm?

Static Single Assignment (SSA)

• Size of a def-use graph
• SSA
  • each static definition defines a new name
  • each use has a single static definition
• Meet operation
  • \( x \leftarrow \Phi(y, z) \)
• Naive SSA insertion
  • Algorithm?
  • How many \( \Phi \) functions are unnecessary?
  • What is the earliest point of each meet?

\[
\begin{align*}
  w & : a \leftarrow a + 1 \\
  x & : c \leftarrow a \times b \\
  y & : c \leftarrow b + d \\
  z & : \text{print}(c)
\end{align*}
\]
Minimal SSA

![Minimal SSA Diagram]

SSA Algorithm

- Dominance frontiers
  - DF(n) definition

- Dominators
  - Dom(n) definition
  - Data flow equation
  - Initialization
  - Convergence

- Dom to DF
  - backward alg. [EAC, Fig. 9.10]
  - forward alg., linear time [AK, Fig. 4.9]
SSA Algorithm

1. CFG
2. compute Dom
3. compute DF
4. insert phi
5. rename
6. reaching def
7. “destruct” SSA

Example 1

• IDOM  DF phi renaming reach destruction
Example 2

- IDOM  DF phi renaming reach destruction

A: \( a = 1 \)

B: \( a = a + 1 \)
$x \leftarrow \ldots$
$y \leftarrow \ldots$
$t \leftarrow x$
$x \leftarrow y$
$y \leftarrow t$

Original Code

$x_0 \leftarrow \ldots$
$y_0 \leftarrow \ldots$
$x_1 \leftarrow \phi(x_0, y_1)$
$y_1 \leftarrow \phi(y_0, x_1)$

SSA Form. Copies Folded

$x_0 \leftarrow \ldots$
$y_0 \leftarrow \ldots$
$x_3 \leftarrow x_0$
$y_1 \leftarrow y_0$

After Naive Copy Insertion

Copyright © Seth Copen Goldstein 2001-5, Tim Callahan 2007

15-745
SSA

C

W <- list of all defs
while !W.isEmpty {
  Stmt S <- W.removeOne
  if |S.users| != 0 then continue
  if S.hasSideEffects() then continue
  foreach def in S.definers {
    def.users <- def.users - {S}
    if |def.uses| == 0 then
      W <- W UNION {def}
  }
  delete S
}

init:
mark as live all stmts that have side-effects:
- I/O
- stores into memory
- returns
- calls a function that MIGHT have side-effects

while (|W| > 0) {
  S <- W.removeOne()
  if (S is live) continue;
  mark S live, insert S.defs into W
}