BITS Pilani
Pilani Campus

Compiler Construction

Vinti Agarwal
April 2021

innovate achieve lead

BITS Pilani
Pilani Campus

CS F363, Compiler Construction

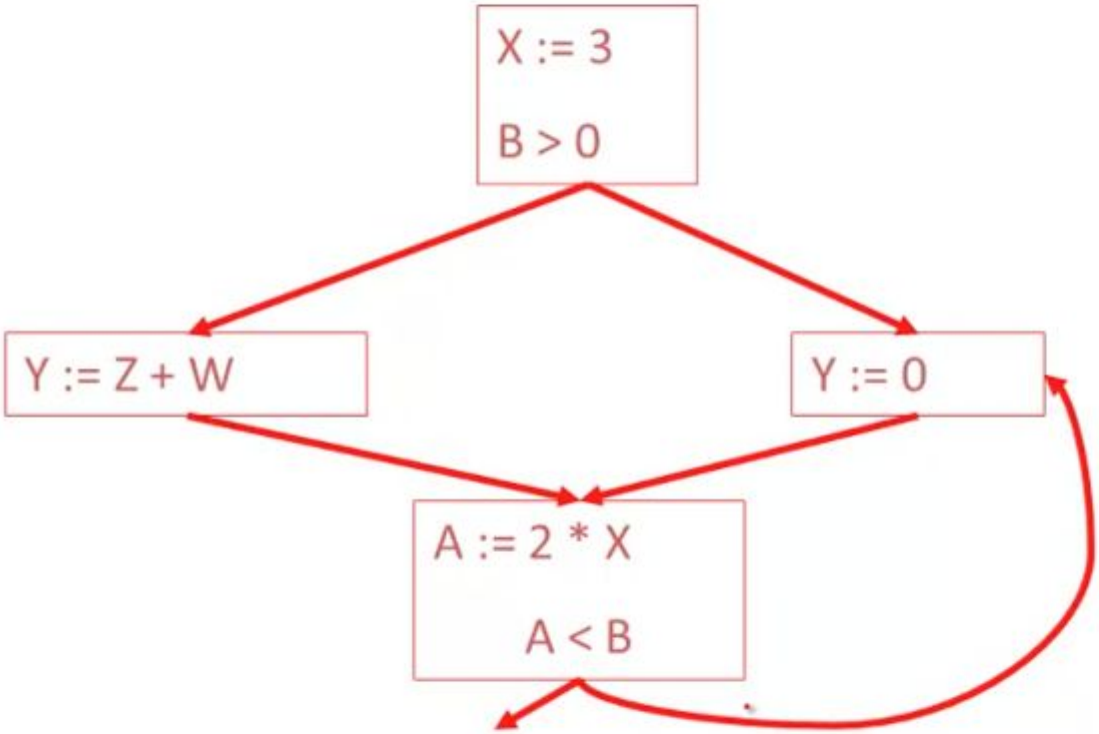
Lecture topic: Global Optimization Part II

In previous lecture

- Dataflow analysis
- Global constant propagation
- top, constant, bottom values
- Transfer function

Analysis of loops

- Need of ⊥ tied to the analysis of loops. To understand why we need this symbol, look at a loo



Analysis of loops

- Consider the statement Y:=0
- To compute whether X is constant at this point, we need to know whether is constant at the two predecessor

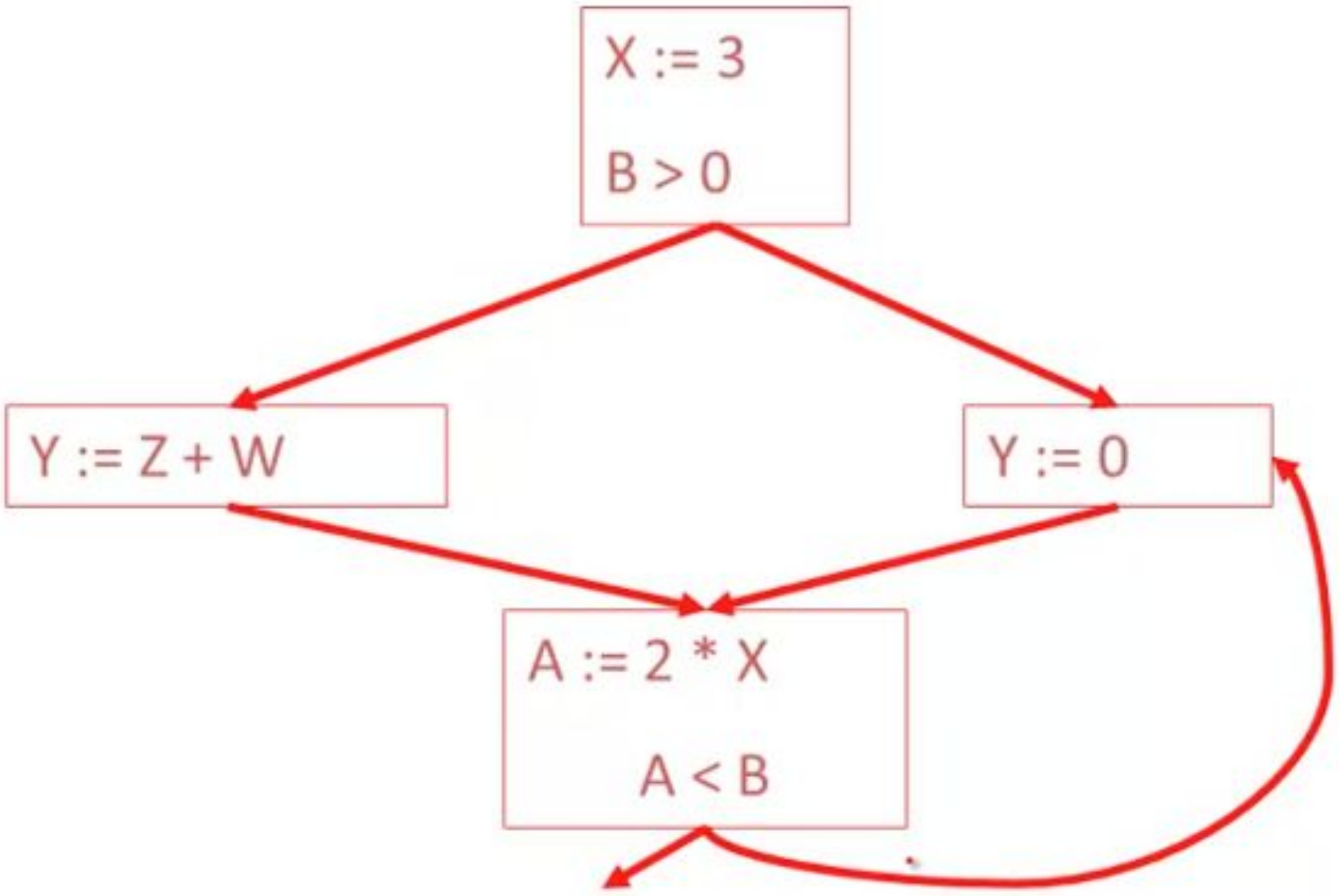
  X := 3

  A := 2*X
- But info for A := 2 *X depends on its predecessors, including Y :=0

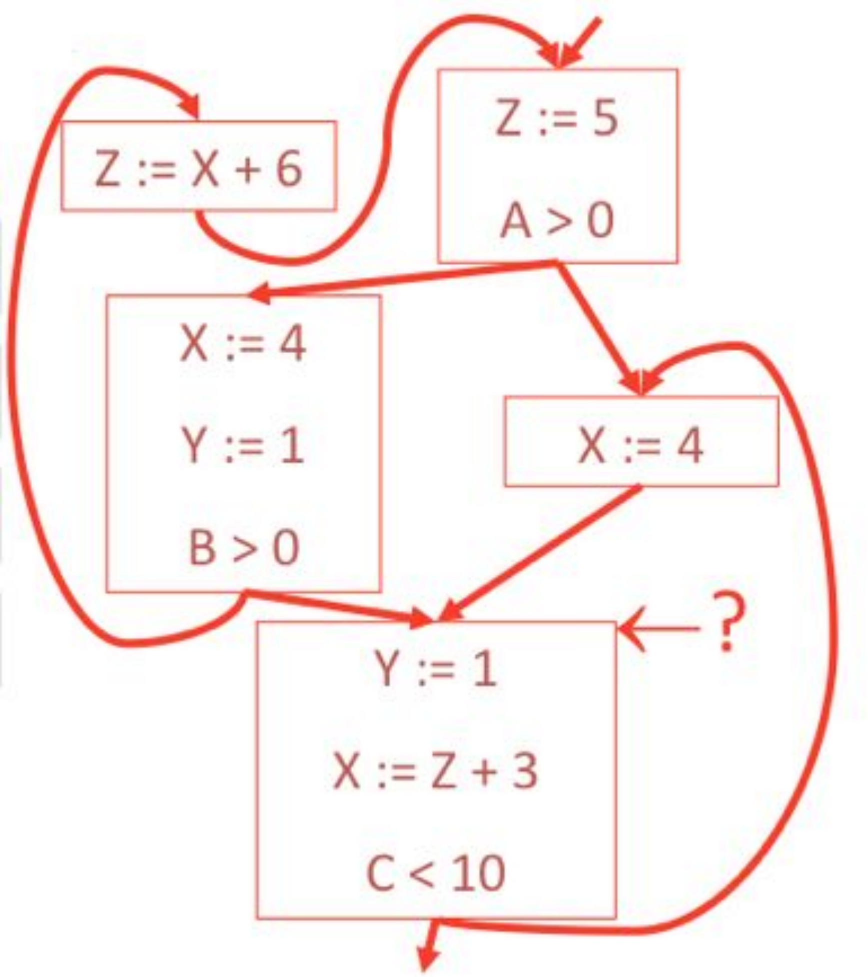
Analysis of loops

- Because of cycles, all point must have values at all times
- Intuitively, assigning some initial values allow the analysis to break cycles
- The initial value bottom ( ) means “so far as we know, control never reaches this point”

Analysis of loops



Example: Find out the values of X,Y, Z at the program point labeled at right



Orderings

- We can simplify the presentation of the analysis by ordering the values

- Drawing the picture with “lower” values drawn lower, we get

Orderings

- T is the greatest value,   is the least
  - all constants are in between and incomparable
- Let ***lub*** be the least-upper bound in this ordering
- Rules 1-4 can be written using ***lub***:
  - C(s, x, in) = lub { C(p, x, out) | p is a predecessor of s}

Orderings

- Simply saying “repeat until nothing changes” doesn’t guarantee that eventually nothing changes
- The use of lub explains why algorithm terminates
  - value start as and only increase
  - can change to a constant and constant to T
  - Thus C (s, x, _) can change at most twice

Orderings

- Thus the constant propagation algorithm is linear in program size

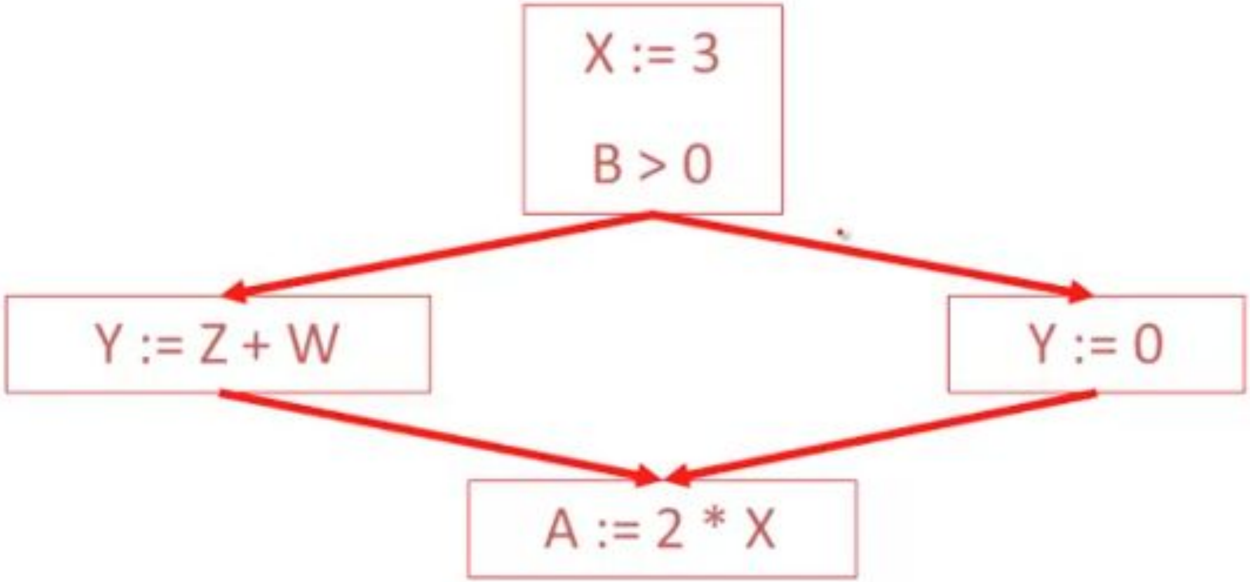
Number of steps =

Number of C (...) values computed *2 =

Number of program statements *4

Liveness Analysis

- Once constants have been globally propagated, we would eliminate dead code



After constant propagation, X := 3 is dead (assuming X is not used elesewhere)

Liveness Analysis

- The first value of x is dead (never used)
- the second value of x is live (may be used)
- liveness is an important concept

Liveness Analysis

- A variable x is live at statement s if
  - There exists a statement s’ that uses x
  - there is a path from s to s’
  - that path has no intervening assignment to x

Liveness Analysis

- A statement x:=.. is dead code if x is dead after the assignment
- Dead statements can be deleted from the program
- but we need liveness information first

Liveness Analysis

- We can express liveness in terms of information transferred between adjacent statement, just as in copy propagation
- Liveness is simpler than constant propagation, since it is boolean property (true/false)

Rule 1:

Rule 2:

Rule 3:

Rule 4:

Liveness analysis

1. Let all L (..) = false initially
2. Repeat until all statements s satisfy rules 1-4

   pick s where one of the 1-4 rule does not hold and update using the appropriate rule

Example

Example

Summary

1. A value can change from false to true, but the other way around
2. Each value can change only once, so termination is guaranteed
3. Once the analysis is computed, it is simple to eliminate dead code

Thank You!