

## Global Optimization

### Lecture 19

Prof. Necula CS 164 Lecture 19

1

## Lecture Outline

- Global flow analysis
- Global constant propagation
- Liveness analysis

Prof. Necula CS 164 Lecture 19

2

## Local Optimization

Recall the simple basic-block optimizations

- Constant propagation
- Dead code elimination

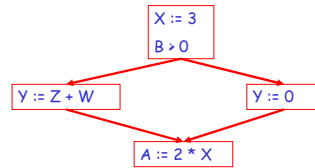
$X := 3$   
 $Y := Z * W$   
 $Q := X + Y$     $\longrightarrow$     $X := 3$   
 $Y := Z * W$   
 $Q := 3 + Y$     $\longrightarrow$     $Y := Z * W$   
 $Q := 3 + Y$

Prof. Necula CS 164 Lecture 19

3

## Global Optimization

These optimizations can be extended to an entire control-flow graph

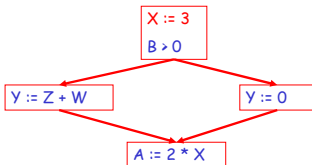


Prof. Necula CS 164 Lecture 19

4

## Global Optimization

These optimizations can be extended to an entire control-flow graph

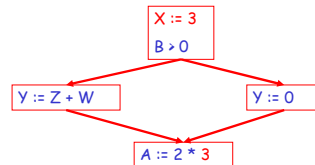


Prof. Necula CS 164 Lecture 19

5

## Global Optimization

These optimizations can be extended to an entire control-flow graph

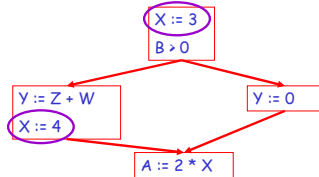


Prof. Necula CS 164 Lecture 19

6

## Correctness

- How do we know it is OK to globally propagate constants?
- There are situations where it is incorrect:



Prof. Neucula CS 164 Lecture 19

7

## Correctness (Cont.)

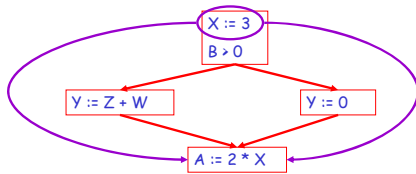
To replace a use of  $x$  by a constant  $k$  we must know that:

*On every path to the use of  $x$ , the last assignment to  $x$  is  $x := k$  \*\**

Prof. Neucula CS 164 Lecture 19

8

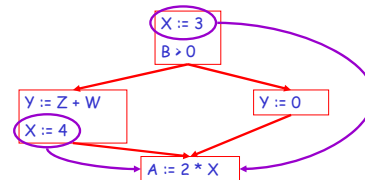
## Example 1 Revisited



Prof. Neucula CS 164 Lecture 19

9

## Example 2 Revisited



Prof. Neucula CS 164 Lecture 19

10

## Discussion

- The correctness condition is not trivial to check
- "All paths" includes paths around loops and through branches of conditionals
- Checking the condition requires global analysis
  - An analysis of the entire control-flow graph for one method body

Prof. Neucula CS 164 Lecture 19

11

## Global Analysis

Global optimization tasks share several traits:

- The optimization depends on knowing a property  $P$  at a particular point in program execution
- Proving  $P$  at any point requires knowledge of the entire method body
- Property  $P$  is typically undecidable !

Prof. Neucula CS 164 Lecture 19

12

## Undecidability of Program Properties

- Rice's theorem: Most interesting dynamic properties of a program are undecidable:
  - Does the program halt on all (some) inputs?
    - This is called the halting problem
  - Is the result of a function  $F$  always positive?
    - Assume we can answer this question precisely
    - Take function  $H$  and find out if it halts by testing function  $F(x) \{ H(x); \text{return } 1; \}$  whether it has positive result
- Syntactic properties are decidable!
  - E.g., How many occurrences of "x" are there?
- Same for programs without looping

Prof. Necula CS 164 Lecture 19

13

## Conservative Program Analyses

- So, we cannot tell for sure that "x" is always 3
  - Then, how can we apply constant propagation?
- It is OK to be conservative. If the optimization requires  $P$  to be true, then want to know either
  - $P$  is definitely true
  - Don't know if  $P$  is true
- It is always correct to say "don't know"
  - We try to say don't know as rarely as possible
- All program analyses are conservative

Prof. Necula CS 164 Lecture 19

14

## Global Analysis (Cont.)

- Global dataflow analysis is a standard technique for solving problems with these characteristics
- Global constant propagation is one example of an optimization that requires global dataflow analysis

Prof. Necula CS 164 Lecture 19

15

## Global Constant Propagation

- Global constant propagation can be performed at any point where  $**$  holds
- Consider the case of computing  $**$  for a single variable  $X$  at all program points

Prof. Necula CS 164 Lecture 19

16

## Global Constant Propagation (Cont.)

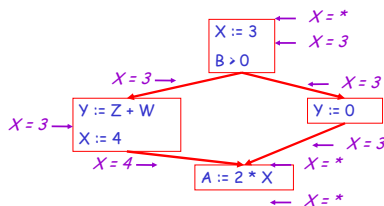
- To make the problem precise, we associate one of the following values with  $X$  at every program point

value	interpretation
#	This statement is not reachable
c	$X = \text{constant } c$
*	Don't know if $X$ is a constant

Prof. Necula CS 164 Lecture 19

17

## Example



Prof. Necula CS 164 Lecture 19

18

## Using the Information

- Given global constant information, it is easy to perform the optimization
  - Simply inspect the  $x = ?$  associated with a statement using  $x$
  - If  $x$  is constant at that point replace that use of  $x$  by the constant
- But how do we compute the properties  $x = ?$

Prof. Necula CS 164 Lecture 19

19

## The Idea

*The analysis of a complicated program can be expressed as a combination of simple rules relating the change in information between adjacent statements*

Prof. Necula CS 164 Lecture 19

20

## Explanation

- The idea is to "push" or "transfer" information from one statement to the next
- For each statement  $s$ , we compute information about the value of  $x$  immediately before and after  $s$

$C_{in}(x, s)$  = value of  $x$  before  $s$

$C_{out}(x, s)$  = value of  $x$  after  $s$

Prof. Necula CS 164 Lecture 19

21

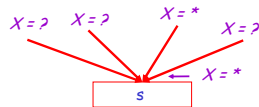
## Transfer Functions

- Define a transfer function that transfers information from one statement to another
- In the following rules, let statement  $s$  have immediate predecessor statements  $p_1, \dots, p_n$

Prof. Necula CS 164 Lecture 19

22

## Rule 1

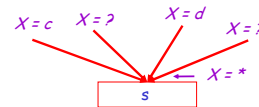


if  $C_{out}(x, p_i) = *$  for some  $i$ , then  $C_{in}(x, s) = *$

Prof. Necula CS 164 Lecture 19

23

## Rule 2

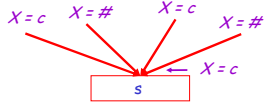


If  $C_{out}(x, p_i) = c$  and  $C_{out}(x, p_j) = d$  and  $d \neq c$  then  $C_{in}(x, s) = *$

Prof. Necula CS 164 Lecture 19

24

### Rule 3

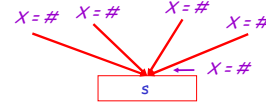


if  $C_{out}(x, p_i) = c$  or  $\#$  for all  $i$ ,  
then  $C_{in}(x, s) = c$

Prof. Neucula CS 164 Lecture 19

25

### Rule 4



if  $C_{out}(x, p_i) = \#$  for all  $i$ ,  
then  $C_{in}(x, s) = \#$

Prof. Neucula CS 164 Lecture 19

26

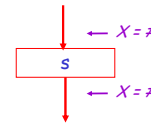
### The Other Half

- Rules 1-4 relate the *out* of one statement to the *in* of the successor statement
  - they propagate information forward across CFG edges
- Now we need rules relating the *in* of a statement to the *out* of the same statement
  - to propagate information across statements

Prof. Neucula CS 164 Lecture 19

27

### Rule 5

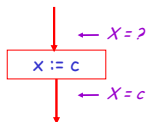


$C_{out}(x, s) = \#$  if  $C_{in}(x, s) = \#$

Prof. Neucula CS 164 Lecture 19

28

### Rule 6

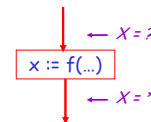


$C_{out}(x, x := c) = c$  if  $c$  is a constant

Prof. Neucula CS 164 Lecture 19

29

### Rule 7

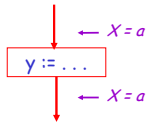


$C_{out}(x, x := f(\dots)) = *$

Prof. Neucula CS 164 Lecture 19

30

## Rule 8



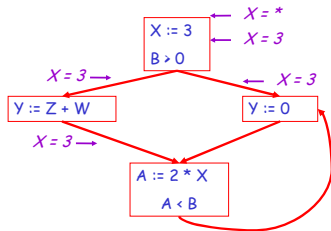
$$C_{out}(x, y := \dots) = C_{in}(x, y := \dots) \text{ if } x \neq y$$

## An Algorithm

1. For every entry  $s$  to the program, set  $C_{in}(x, s) = *$
2. Set  $C_{in}(x, s) = C_{out}(x, s) = \#$  everywhere else
3. Repeat until all points satisfy 1-8:  
Pick  $s$  not satisfying 1-8 and update using the appropriate rule

## The Value #

- To understand why we need  $\#$ , look at a loop



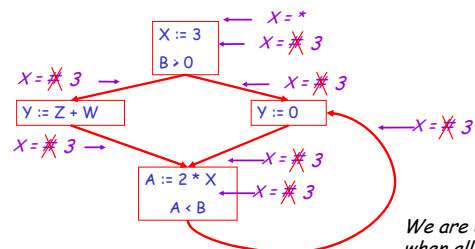
## Discussion

- Consider the statement  $Y := 0$
- To compute whether  $X$  is constant at this point, we need to know whether  $X$  is constant at the two predecessors
  - $X := 3$
  - $A := 2 * X$
- But info for  $A := 2 * X$  depends on its predecessors, including  $Y := 0$ !

## The Value # (Cont.)

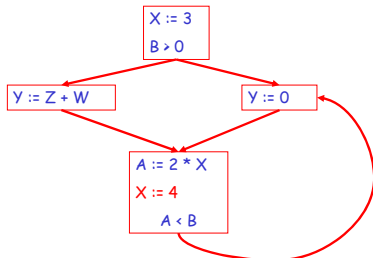
- Because of cycles, all points must have values at all times
- Intuitively, assigning some initial value allows the analysis to break cycles
- The initial value  $\#$  means "So far as we know, control never reaches this point"

## Example



*We are done when all rules are satisfied!*

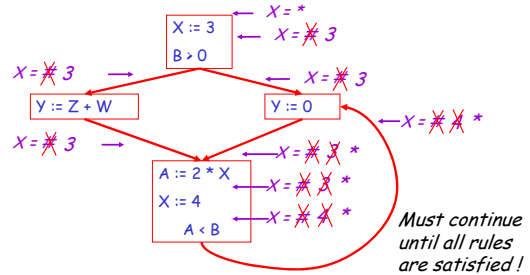
## Another Example



Prof. Necula CS 164 Lecture 19

37

## Another Example



Prof. Necula CS 164 Lecture 19

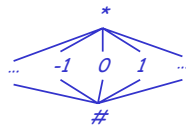
38

## Orderings

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

$$\# < c < *$$

- Drawing a picture with "lower" values drawn lower, we get



Prof. Necula CS 164 Lecture 19

39

## Orderings (Cont.)

- \* 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:
 
$$C_{in}(x, s) = \text{lub} \{ C_{out}(x, p) \mid p \text{ is a predecessor of } s \}$$

Prof. Necula CS 164 Lecture 19

40

## Termination

- Simply saying "repeat until nothing changes" doesn't guarantee that eventually nothing changes
- The use of lub explains why the algorithm terminates
  - Values start as # and only *increase*
  - # can change to a constant, and a constant to \*
  - Thus,  $C_{in}(x, s)$  can change at most twice

Prof. Necula CS 164 Lecture 19

41

## Termination (Cont.)

Thus the algorithm is linear in program size

Number of steps =

Number of  $C_{in}(\dots)$  values computed \* 2 =

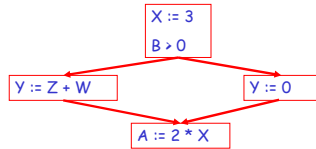
Number of program statements \* 4

Prof. Necula CS 164 Lecture 19

42

## Liveness Analysis

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



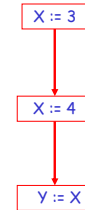
After constant propagation,  $X := 3$  is dead (assuming this is the entire CFG)

Prof. Neucula CS 164 Lecture 19

43

## Live and Dead

- The first value of  $x$  is *dead* (never used)
- The second value of  $x$  is *live* (may be used)
- Liveness is an important concept



Prof. Neucula CS 164 Lecture 19

44

## Liveness

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  $x$

Prof. Neucula CS 164 Lecture 19

45

## Global Dead Code Elimination

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

Prof. Neucula CS 164 Lecture 19

46

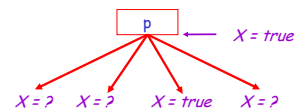
## Computing Liveness

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

Prof. Neucula CS 164 Lecture 19

47

## Liveness Rule 1

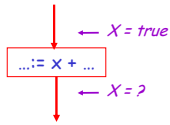


$$L_{\text{out}}(x, p) = \vee \{ L_{\text{in}}(x, s) \mid s \text{ a successor of } p \}$$

Prof. Neucula CS 164 Lecture 19

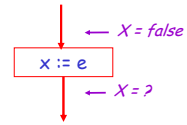
48

### Liveness Rule 2



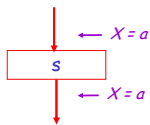
$L_{in}(x, s) = true$  if  $s$  refers to  $x$  on the rhs

### Liveness Rule 3



$L_{in}(x, x := e) = false$  if  $e$  does not refer to  $x$

### Liveness Rule 4

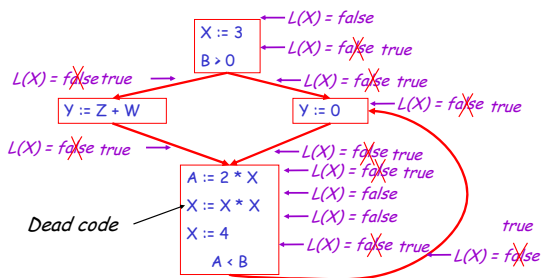


$L_{in}(x, s) = L_{out}(x, s)$  if  $s$  does not refer to  $x$

### Algorithm

1. Let all  $L_{in}(\dots) = false$  initially
2. Repeat until all statements  $s$  satisfy rules 1-4  
Pick  $s$  where one of 1-4 does not hold and update using the appropriate rule

### Another Example



### Termination

- A value can change from **false** to **true**, but not the other way around
- Each value can change only once, so termination is guaranteed
- Once the analysis is computed, it is simple to eliminate dead code

## Forward vs. Backward Analysis

---

We've seen two kinds of analysis:

Constant propagation is a *forwards* analysis: information is pushed from inputs to outputs

Liveness is a *backwards* analysis: information is pushed from outputs back towards inputs

## Analysis

---

- There are many other global flow analyses
- Most can be classified as either forward or backward
- Most also follow the methodology of local rules relating information between adjacent program points