

Global Optimization

Lecture 15

Prof. Aiken CS 143 Lecture 15

1

Lecture Outline

- Global flow analysis
- Global constant propagation
- Liveness analysis

Prof. Aiken CS 143 Lecture 15

2

Local Optimization

Recall the simple basic-block optimizations

- Constant propagation
- Dead code elimination

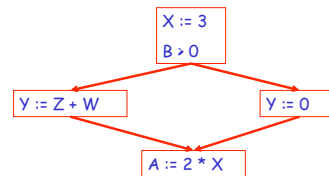
$X := 3$
 $Y := Z * W$
 $Q := X + Y$ \rightarrow $X := 3$
 $Y := Z * W$
 $Q := 3 + Y$ \rightarrow $Y := Z * W$
 $Q := 3 + Y$

Prof. Aiken CS 143 Lecture 15

3

Global Optimization

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

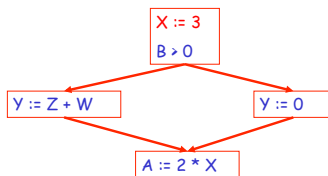


Prof. Aiken CS 143 Lecture 15

4

Global Optimization

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

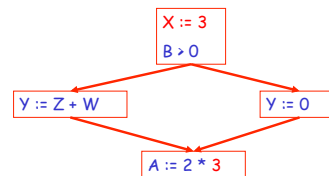


Prof. Aiken CS 143 Lecture 15

5

Global Optimization

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

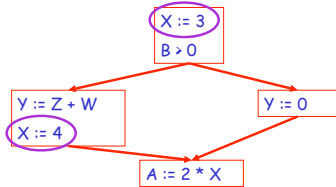


Prof. Aiken CS 143 Lecture 15

6

Correctness

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



Prof. Aiken CS 143 Lecture 15

7

Correctness (Cont.)

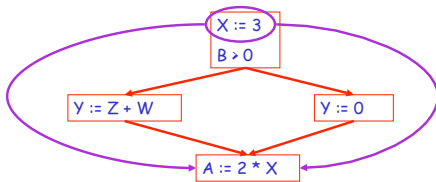
To replace a use of x by a constant k we must know that:

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

Prof. Aiken CS 143 Lecture 15

8

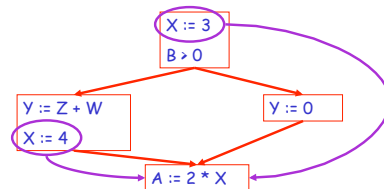
Example 1 Revisited



Prof. Aiken CS 143 Lecture 15

9

Example 2 Revisited



Prof. Aiken CS 143 Lecture 15

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

Prof. Aiken CS 143 Lecture 15

11

Global Analysis

Global optimization tasks share several traits:

- The optimization depends on knowing a property X at a particular point in program execution
- Proving X at any point requires knowledge of the entire program
- It is OK to be conservative. If the optimization requires X to be true, then want to know either
 - X is definitely true
 - Don't know if X is true
- It is always safe to say “don't know”

Prof. Aiken CS 143 Lecture 15

12

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. Aiken CS 143 Lecture 15

13

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. Aiken CS 143 Lecture 15

14

Global Constant Propagation (Cont.)

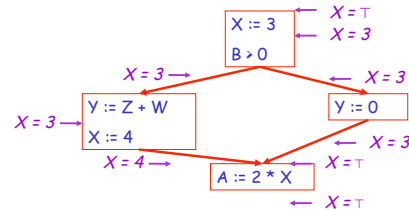
- To make the problem precise, we associate one of the following values with **X** at every program point

value	interpretation
\perp	This statement never executes
c	X = constant c
T	X is not a constant

Prof. Aiken CS 143 Lecture 15

15

Example



Prof. Aiken CS 143 Lecture 15

16

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. Aiken CS 143 Lecture 15

17

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. Aiken CS 143 Lecture 15

18

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

$C(s, x, \text{in})$ = value of x before s

$C(s, x, \text{out})$ = value of x after s

Prof. Aiken CS 143 Lecture 15

19

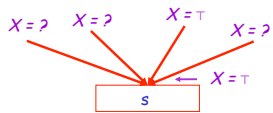
Transfer Functions

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

Prof. Aiken CS 143 Lecture 15

20

Rule 1

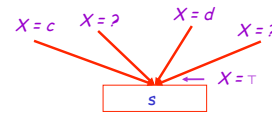


if $C(p_i, x, \text{out}) = \top$ for any i , then $C(s, x, \text{in}) = \top$

Prof. Aiken CS 143 Lecture 15

21

Rule 2

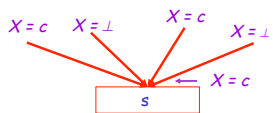


$C(p_i, x, \text{out}) = c$ & $C(p_j, x, \text{out}) = d$ & $d \neq c$ then
 $C(s, x, \text{in}) = \top$

Prof. Aiken CS 143 Lecture 15

22

Rule 3

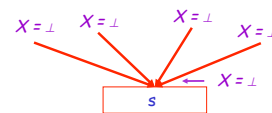


if $C(p_i, x, \text{out}) = c$ or \perp for all i ,
then $C(s, x, \text{in}) = c$

Prof. Aiken CS 143 Lecture 15

23

Rule 4



if $C(p_i, x, \text{out}) = \perp$ for all i ,
then $C(s, x, \text{in}) = \perp$

Prof. Aiken CS 143 Lecture 15

24

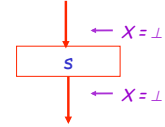
The Other Half

- Rules 1-4 relate the *out* of one statement to the *in* of the next statement
- Now we need rules relating the *in* of a statement to the *out* of the same statement

Prof. Aiken CS 143 Lecture 15

25

Rule 5

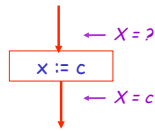


$$C(s, x, \text{out}) = \perp \text{ if } C(s, x, \text{in}) = \perp$$

Prof. Aiken CS 143 Lecture 15

26

Rule 6

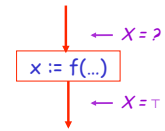


$$C(x := c, x, \text{out}) = c \text{ if } c \text{ is a constant}$$

Prof. Aiken CS 143 Lecture 15

27

Rule 7

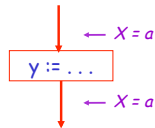


$$C(x := f(...), x, \text{out}) = \top$$

Prof. Aiken CS 143 Lecture 15

28

Rule 8



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

Prof. Aiken CS 143 Lecture 15

29

An Algorithm

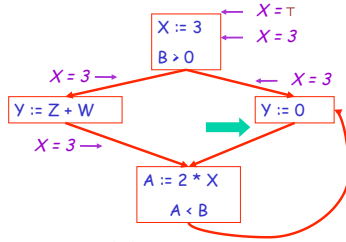
- For every entry s to the program, set $C(s, x, \text{in}) = \top$
- Set $C(s, x, \text{in}) = C(s, x, \text{out}) = \perp$ everywhere else
- Repeat until all points satisfy 1-8:
Pick s not satisfying 1-8 and update using the appropriate rule

Prof. Aiken CS 143 Lecture 15

30

The Value \perp

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



Prof. Aiken CS 143 Lecture 15

31

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 $Y := 0$!

Prof. Aiken CS 143 Lecture 15

32

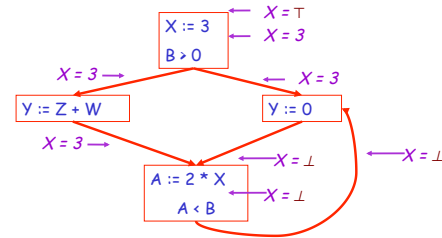
The Value \perp (Cont.)

- 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 \perp means "So far as we know, control never reaches this point"

Prof. Aiken CS 143 Lecture 15

33

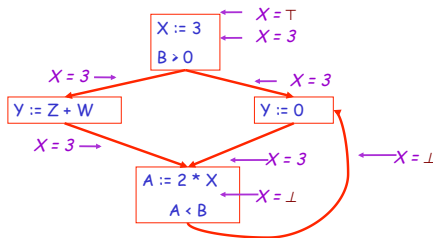
Example



Prof. Aiken CS 143 Lecture 15

34

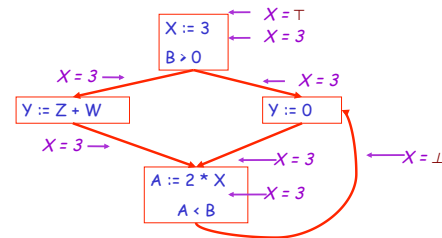
Example



Prof. Aiken CS 143 Lecture 15

35

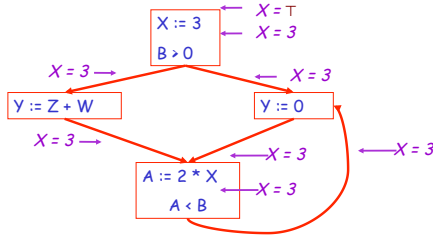
Example



Prof. Aiken CS 143 Lecture 15

36

Example



Prof. Aiken CS 143 Lecture 15

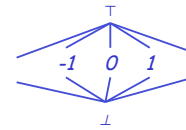
37

Orderings

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

$$\perp < C < T$$

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



Prof. Aiken CS 143 Lecture 15

38

Orderings (Cont.)

- T is the greatest value, \perp 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:

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

Prof. Aiken CS 143 Lecture 15

39

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 \perp and only *increase*
 - \perp can change to a constant, and a constant to T
 - Thus, $C(s, x, _)$ can change at most twice

Prof. Aiken CS 143 Lecture 15

40

Termination (Cont.)

Thus the algorithm is linear in program size

Number of steps =

Number of $C(\dots)$ value computed $\times 2 =$

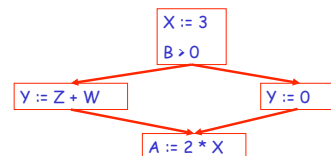
Number of program statements $\times 4$

Prof. Aiken CS 143 Lecture 15

41

Liveness Analysis

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



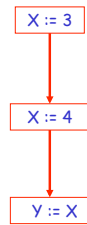
After constant propagation, $X := 3$ is dead (assuming X not used elsewhere)

Prof. Aiken CS 143 Lecture 15

42

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. Aiken CS 143 Lecture 15

43

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. Aiken CS 143 Lecture 15

44

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. Aiken CS 143 Lecture 15

45

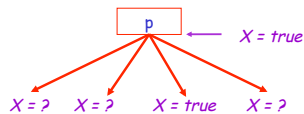
Computing Liveness

- 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. Aiken CS 143 Lecture 15

46

Liveness Rule 1

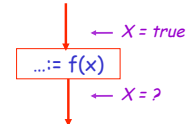


$$L(p, x, \text{out}) = \vee \{ L(s, x, \text{in}) \mid s \text{ a successor of } p \}$$

Prof. Aiken CS 143 Lecture 15

47

Liveness Rule 2

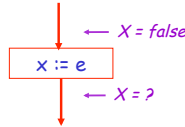


$$L(s, x, \text{in}) = \text{true} \text{ if } s \text{ refers to } x \text{ on the rhs}$$

Prof. Aiken CS 143 Lecture 15

48

Liveness Rule 3

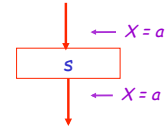


$L(x := e, x, in) = \text{false}$ if e does not refer to x

Prof. Aiken CS 143 Lecture 15

49

Liveness Rule 4



$L(s, x, in) = L(s, x, out)$ if s does not refer to x

Prof. Aiken CS 143 Lecture 15

50

Algorithm

1. Let all $L(\dots) = \text{false}$ 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

Prof. Aiken CS 143 Lecture 15

51

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

Prof. Aiken CS 143 Lecture 15

52

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

Prof. Aiken CS 143 Lecture 15

53

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

Prof. Aiken CS 143 Lecture 15

54