Global Optimizations

Lecture 12
Organization of a Code Optimizer (Revisited)
Local Optimization

Recall the simple basic-block optimizations
  - Constant propagation
  - Dead code elimination

\[
\begin{align*}
X &:= 3 \\
Y &:= Z \times W \\
Q &:= X + Y
\end{align*}
\]

Prof. Aiken
Global Optimization

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

\[ X := 3 \]
\[ B > 0 \]
\[ Y := Z + W \]
\[ A := 2 \times X \]
\[ Y := 0 \]
Global Optimization

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

\[ X := 3 \]
\[ B > 0 \]
\[ Y := Z + W \]
\[ Y := 0 \]
\[ A := 2 \times X \]
Global Optimization

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

\[
X := 3
\]
\[
B > 0
\]
\[
Y := Z + W
\]
\[
Y := 0
\]
\[
A := 2 \times 3
\]
Correctness

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

```plaintext
X := 3
B > 0
Y := Z + W
X := 4
Y := 0
A := 2 * X

We cannot propagate constant 3 or 4 to X
```
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 $x := k$ **_
Example 1 Revisited

\[ X := 3 \]
\[ B > 0 \]
\[ Y := Z + W \]
\[ Y := 0 \]
\[ A := 2 \times X \]
Example 2 Revisited

\[ X := 3 \]
\[ B > 0 \]
\[ Y := Z + W \]
\[ X := 4 \]
\[ Y := 0 \]
\[ A := 2 \times X \]
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 dataflow analysis
  - An analysis of the entire control-flow graph
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 (we don’t do the optimization).
- It is always safe to say “don’t know.”
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
Global Constant Propagation

- Global constant propagation can be performed at any point where ** holds

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

- Consider the case of computing ** for a single variable $X$ at all program points
Global Constant Propagation (Cont.)

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

<table>
<thead>
<tr>
<th>value</th>
<th>interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\times$</td>
<td>This statement never executes</td>
</tr>
<tr>
<td>$c$</td>
<td>$X = \text{constant } c$</td>
</tr>
<tr>
<td>$\square$</td>
<td>$X$ is not a constant</td>
</tr>
</tbody>
</table>
Example

Prof. Aiken
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 $x = ?$
The analysis of a complicated program can be expressed as a combination of simple rules relating the change in information between adjacent statements.
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 $s$:

- $C(x, s, \text{in}) = \text{value of } x \text{ before } s \text{ is executed}$
- $C(x, s, \text{out}) = \text{value of } x \text{ after } s \text{ is executed}$

*Stands for ‘constant’ information*
Transfer Functions

• Define a *transfer* function that transfers information one statement to another

• In the following rules, let statement $s$ have immediate predecessor statements $p_1, \ldots, p_n$
Rule 1

if $\exists i \ (C(p_i, x, \text{out}) = \square)$
then $C(s, x, \text{in}) = \square$
Rule 2

if \( \exists_{i,j} \ (C(p_i, x, out) = c \land C(p_j, x, out) = d \land d \not= c) \)
then \( C(s, x, in) = \square \)
Rule 3

\[
\begin{align*}
\text{if } & \forall_i (C(p_i, x, \text{out}) = c \text{ or } \otimes) \\
& \text{then } C(s, x, \text{in}) = c
\end{align*}
\]
Rule 4

if \( \forall_i (C(p_i, x, \text{out}) = \otimes) \)
then \( C(s, x, \text{in}) = \otimes \)
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
Rule 5

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

Prof. Aiken [slightly modified]
Rule 6

Rule 6 has a lower priority than Rule 5

R6 is checked only when R5 cannot be applied

if c is a constant
then $C(x := c, x, out) = c$
Rule 7

Rule 7 has a lower priority than Rule 5

R7 is checked only when R5 cannot be applied

\[
x := f(\ldots)
\]

\[f(\ldots) \text{ is anything except a constant}\]

\[
C(x := f(\ldots), x, \text{out}) = \square
\]
Rule 8

if $x \not= y$
then $C(y := \ldots, x, \text{out}) = C(y := \ldots, x, \text{in})$
An Algorithm

1. For every entry $s$ to the program, set $C(s, x, \text{in}) = \square$

2. Set $C(s, x, \text{in}) = C(s, x, \text{out}) = \times$ everywhere else

3. Repeat until all points satisfy 1-8:
   Pick $s$ not satisfying 1-8 and update using the appropriate rule
Constant Propagation

\[
\begin{align*}
X &:= 3 \\
B &> 0 \\
Y &:= Z + W \\
X &:= 4 \\
A &:= 2 \times X \\
Y &:= 0 \\
X &:= 0
\end{align*}
\]
Constant Propagation

Rule 6: if $c$ is a constant then $C(x := c, x, \text{out}) = c$
Constant Propagation

Rule 8: if $x \neq y$ then $C(y := \ldots, x, \text{out}) = C(y := \ldots, x, \text{in})$
Constant Propagation

Rule 3: if $\forall_i (C(p_i, x, out) = c \text{ or } \otimes)$ then $C(s, x, in) = c$
Constant Propagation

\[
\begin{align*}
X &:= 3 \\
B &> 0 \\
Y &:= Z + W \\
X &:= 4
\end{align*}
\]

**Rule 8:** if \( x \neq y \) then 
\[
C(y := \ldots, x, out) = C(y := \ldots, x, in)
\]
Constant Propagation

Rule 6: if \( c \) is a constant then \( C(x := c, x, \text{out}) = c \)
**Constant Propagation**

\[
\begin{align*}
X &:= 3 \\
B &> 0 \\
X &:= 3 \\
Y &:= Z + W \\
X &:= 4 \\
X &:= 3 \\
Y &:= 0 \\
A &:= 2 \times X \\
X &:= 4 \\
X &:= 3 \\
X &:= \square \quad \text{by rule 2} \\
X &:= \otimes
\end{align*}
\]

Rule 2: if \( \exists_{i,j} (C(p_i, x, \text{out}) = c \land C(p_j, x, \text{out}) = d \land d \not\leftrightarrow c) \)
then \( C(s, x, \text{in}) = \square \)

Prof. Aiken [slightly modified]
**Constant Propagation**

Rule 8: if \( x \neq y \) then \( C(y := ..., x, out) = C(y := ..., x, in) \)
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 $\otimes$ (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 $\otimes$ means “So far as we know, control never reaches this point”
Example

\[ X := 3 \]
\[ B > 0 \]
\[ Y := Z + W \]
\[ A := 2 \times X \]
\[ A < B \]
Example

\[ X := 3 \]
\[ B > 0 \]

\[ Y := Z + W \]

\[ A := 2 \times X \]
\[ A < B \]

\[ Y := 0 \]
Example

\[ X := 3 \]
\[ B > 0 \]
\[ Y := Z + W \]
\[ A := 2 \times X \]
\[ A < B \]
\[ Y := 0 \]
Example

\[ X := 3 \]
\[ B > 0 \]
\[ Y := Z + W \]
\[ Y := 0 \]
\[ A := 2 \times X \]
\[ A < B \]
Example

$X := 3$
$B > 0$

$Y := Z + W$

$A := 2 \times X$
$A < B$

$Y := 0$

$X = 3$
Example

\[
X := 3 \\
B > 0
\]

\[
Y := Z + W
\]

\[
A := 2 \times X \\
A < B
\]

\[
Y := 0
\]

\[
X = \square
\]

\[
X = 3
\]

\[
X = 3
\]

\[
X = 3
\]

\[
X = 3
\]

\[
X = 3
\]

\[
X = 3
\]

\[
X = 3
\]

\[
X = 3
\]
Example

X := 3
B > 0

Y := Z + W

A := 2 * X
A < B

Y := 0

X = 3
Orderings

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

  \( \otimes < c < \square \)

  these are abstract values

  e.g., \( \square \) stands for any possible run time value

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

  constants are not comparable

  e.g., 0 is not less than 1

Prof. Aiken [slightly modified]
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
  - Examples: lub(1, 2) = □, lub(□, □) = □, lub(1, □) = 1, ...

• Rules 1-4 are in fact computing lub:
  \[ C(s, x, \text{in}) = \text{lub} \{ C(p, x, \text{out}) \mid p \text{ is a predecessor of } s \} \]
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 $\otimes$ and only increase
    $\otimes$ can change to a constant, and a constant to $\otimes$
  - Thus, $C(s, x, \_)$ can change at most twice
Termination (Cont.)

Thus the algorithm is linear in program size

Number of steps =
Number of $C(....)$ value computed * 2 =
Number of program statements * 4

for each statement $s$, we have
one $c(s, x, \text{in})$ and one $c(s, x, \text{out})$,
each can change at most twice
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)
Live and Dead

- The first value of $x$ is dead (never used)

- The second value of $x$ is live (may be used in the future)

- Liveness is an important concept

Prof. Aiken [slightly modified]
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 $x$
Global Dead Code Elimination

• A statement $x := \ldots$ is dead code if $x$ is dead after the assignment

• Dead statements can be deleted from the program

• But we need liveness information first . . .
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).
Liveness Rule 1

\[ L(p, x, \text{out}) = \lor \{ L(s, x, \text{in}) | s \text{ a successor of } p \} \]
Liveness Rule 2

\[ L(s, x, \text{in}) = \text{true} \text{ if } s \text{ refers to } x \text{ on the rhs} \]
Liveness Rule 3

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

because \( x \) is being overwritten here
Liveness Rule 4

\[ L(s, x, \text{in}) = L(s, x, \text{out}) \text{ if } s \text{ does not refer to } x \]
Algorithm

1. Let all \( L(...) = false \) 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

Example

If (x == 10)

x := x + 1

where x is live?
Example

Rule 2: \( L(s, x, \text{in}) = \text{true} \) if \( s \) refers to \( x \) on the rhs
Example

Rule 2: $L(s, x, \text{in}) = \text{true}$ if $s$ refers to $x$ on the rhs

Prof. Aiken
Example

If \( x = 10 \)

\[
x := x + 1
\]

where \( x \) is live?

\[
X := 0 \leftarrow \text{false}
\]

by rule 2

Rule 2: \( L(s, x, \text{in}) = \text{true} \) if \( s \) refers to \( x \) on the rhs
Example

Rule 1: $L(p, x, \text{out}) = \lor \{ L(s, x, \text{in}) \mid s \text{ a successor of } p \}$
Example

Rule 3: \( L(x := e, x, \text{in}) = \text{false} \) if \( e \) does not refer to \( x \)
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 \textit{forwards} analysis: information is pushed from inputs to outputs

Liveness is a \textit{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
**Question?**

After running the constant propagation algorithm to completion, choose the correct dataflow information for X, Y, and Z at the program point labeled at right.

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Z := 5
C > 0

Y := 1
X := 4
Z := X + Y

A := X * Y
B := A * Z

Prof. Aiken
Question?

After running the constant propagation algorithm to completion, choose the correct dataflow information for $X$, $Y$, and $Z$ at the program point labeled at right.

<table>
<thead>
<tr>
<th></th>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Z := X + 6
A > 0
X := 4
Y := 1
B > 0
Y := 1
X := Z + 3
C < 10

Prof. Aiken
Question?

After running the liveness analysis algorithm to completion, which of $W$, $X$, $Y$, and $Z$ are live at the program point labeled at right? Assume all variables are dead on exit.