Lecture 13
SSA-Style Optimizations

(Slides courtesy of Seth Goldstein.)

Review: Minimal SSA
- Each assignment generates a fresh variable.
- At each join point insert Φ functions for all variables with multiple outstanding defs.

```
x <- 1
y <- x
y <- 2
z <- y + x
```

Review: Dominance Frontier and Path Convergence

```
1 2 3 4 5 6 7 8 9 10 11 12 13
```

Constant Propagation
- If "v <- c", replace all uses of v with c
- If "v <- Φ(c,c,c)", replace all uses of v with c

```
W <- list of all defs
while !W.isEmpty {
    Stmt S <- W.removeOne
    if S has form "v <- Φ(c,...,c)"
        replace S with V <- c
    if S has form "v <- c" then
        delete S
    foreach stmt U that uses v,
        replace v with c in U
    W.add(U)
}
```
Other Optimizations with SSA

- Copy propagation
  - delete "x ← \Phi(y,y,y)" and replace all x with y
  - delete "x ← y" and replace all x with y
- Constant Folding
  - (Also, constant conditions too!)
- Unreachable Code
  - Remember to delete all edges from unreachable block
### Constant Propagation

- \( i_1 \leftarrow 1 \)
- \( j_1 \leftarrow 1 \)
- \( k_1 \leftarrow 0 \)
- \( j_2 \leftarrow \Phi(j_4,1) \)
- \( k_2 \leftarrow \Phi(k_4,1) \)
- \( k_2 < 100? \)
- \( j_2 < 20? \)
- return \( j_2 \)
- \( j_3 \leftarrow 1 \)
- \( k_3 \leftarrow k_2 + 1 \)
- \( j_4 \leftarrow \Phi(1,j_5) \)
- \( k_4 \leftarrow \Phi(k_3,k_5) \)

### But, so what?

### Conditional Constant Propagation

- \( i_1 \leftarrow 1 \)
- \( j_1 \leftarrow 1 \)
- \( k_1 \leftarrow 0 \)
- \( j_2 \leftarrow \Phi(j_4,1) \)
- \( k_2 \leftarrow \Phi(k_4,1) \)
- \( k_2 < 100? \)
- \( j_2 < 20? \)
- return \( j_2 \)
- \( j_3 \leftarrow 1 \)
- \( k_3 \leftarrow k_2 + 1 \)
- \( j_4 \leftarrow \Phi(1,j_5) \)
- \( k_4 \leftarrow \Phi(k_3,k_5) \)

---

### Conditional Constant Propagation Algorithm

Keeps track of:
- **Blocks**
  - assume unexecuted until proven otherwise
- **Variables**
  - assume not executed (only with proof of assignments of a non-constant value do we assume not constant)

Lattice for representing variables:

```
T: not executed
...
-2 -1 0 1 2
1: we have seen evidence that the variable has been assigned a constant with the value
we have seen evidence that the variable can hold different values at different times
```

---

### Does block 6 ever execute?
- Simple CP can’t tell
- Conditional CP can tell:
  - Assumes blocks don’t execute until proven otherwise
  - Assumes values are constants until proven otherwise
### Conditional Constant Propagation

Initial values:
- \(i_1 = 1\)
- \(j_1 = 1\)
- \(k_1 = 0\)

Conditional operators:
- \(j_2 < 20?\)
- \(k_2 < 100?\)

Operations:
- \(j_3 \leftarrow k_2 + 1\)
- \(k_5 \leftarrow k_2 + 2\)

Final values:
- \(j_3\)
- \(k_5\)

### Dead Code Elimination

Algorithm:

1. \(W \leftarrow \text{list of all defs}\)
2. while \(!W.\text{isEmpty}\) {
   - \(\text{Stmt } S \leftarrow W.\text{removeOne}\)
   - if \(|S.\text{users}| \neq 0\) then continue
   - if \(S.\text{hasSideEffects}\) then continue
   - foreach \(\text{def in } S.\text{definers}\) {
     - \(\text{def.users} \leftarrow \text{def.users} - \{S\}\)
     - if \(|\text{def.users}| = 0\) then
       - \(W \leftarrow W.\text{UNION}(\text{def})\)
   - delete \(S\)
}

### Example DCE

**B0**
- \(i \leftarrow 0\)
- \(j \leftarrow 0\)

**B1**
- \(i \leftarrow i \times 2\)
- \(j \leftarrow j + 1\)
- \(j < 10?\)

**B2**
- return \(j\)

**Standard DCE leaves Zombies!**
Aggressive Dead Code Elimination

Assume a statement is dead until proven otherwise.

init:
mark as live all stmts that have side-effects:
- I/O
- stores into memory
- returns
- calls a function that MIGHT have side-effects
As we mark S live, insert S.defs into W

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

Example DCE

i0 <- 0
j0 <- 0
j2 <- j1+1
j < 10?

Problem!

Fixing DCE

if S is live, then
    if T determines if S can execute, T should be live

Control Dependence

Y is control-dependent on X if
- X branches to u and v
- 3 a path u->exit which does not go through Y
- 3 paths v->exit go through Y
i.e. X can determine whether or not Y is executed.
Aggressive Dead Code Elimination
Assume a statement is dead until proven otherwise.

```java
while (|W| > 0) {
    S <- W.removeOne()
    if (S is live) continue;
    mark S live, insert:
    - forall operands, S.operand.definers into W
    - S.CD into W
}
```

Example DCE

```
B0  i0<-0
  j0<-0
B1  j1 <- Φ (j0, j3)
    i1 <- Φ (i0, i3)
    j2<10?
B2  return j2
```

Conditional Constant Propagation

```
1  i1 <- 1
  j1 <- 1
  k0 <- 0
2  j2 <- Φ (j4, 1)
    k2 <- Φ (k4, 1)
    k2 < 100?
3  j2 < 20?
4  return j3
5  j3 <- 1
    k3 <- k2 + 1
6  j3 <- k3
    k3 <- k2 + 2
7  j4 <- Φ (1, j3)
    k4 <- Φ (k3, k5)
```

(Recall from earlier.)

- Does block 6 ever execute?
- Simple CP can’t tell
- Conditional CP can tell:
  - Assumes blocks don’t execute until proven otherwise
  - Assumes values are constants until proven otherwise
Applying Dead Code Elimination to the Result of CCP

After CCP

\[
\begin{align*}
& i_1 \leftarrow 1 \\
& j_1 \leftarrow 1 \\
& k_1 \leftarrow 0 \\
& k_2 \leftarrow \Phi(k_3, 0) \\
& k_2 < 100?
\end{align*}
\]

After DCE

\[
\text{return } 1
\]

Small problem.

Finding the Control Dependence Graph

Y is control-dependent on X if
- X branches to u and v
- \exists a path u \rightarrow \text{exit} which does not go through Y
- Y paths v \rightarrow \text{exit} go through Y

i.e. X can determine whether or not Y is executed.

Dominance Frontier and Path Convergence

Finding the Control Dependence Graph

Y is control-dependent on X if
- X branches to u and v
- \exists a path u \rightarrow \text{exit} which does not go through Y
- Y paths v \rightarrow \text{exit} go through Y

i.e. X can determine whether or not Y is executed.
Finding the CDG

- Construct CFG
- Add entry node and exit node
- Add (entry, exit)
- Create $G'$, the reverse CFG
- Compute $D$-tree in $G'$ (post-dominators of $G$)
- Compute $DFG'(y)$ for all $y \in G'$ (post-DF of $G$)
- Add $(x, y) \in G$ to CDG if $x \in DFG'(y)$

CDG of example

Entry:

exit: {}
2: {entry}
1: {1, entry}
0: {entry}
entry: ()

CDG of example

Exit:

exit
2
1
0