

Tour of common optimizations

Simple example

```
foo(z) {  
    x := 3 + 6;  
    y := x - 5  
    return z * y  
}
```

Simple example

```
foo(z) {  
    x := 3 + 6; 9      Constant folding (CF)  
    y := x - 5; 4 (CF)  const prop (CF)  
    return z * y; 4 (CF)  
}
```

Strength reduction (points to the `z` in the return statement)

With simpl (bracketed on the right side of the code)

Another example

```
x := a + b;  
...  
y := a + b;
```

Another example

```
x := a + b;  
...  
y := a + b; x
```

only if x, a, b not modified!

Another example

```
if (...) {  
    a := read();  
    x := a + b;  
    print(x);  
}  
...  
y := a + b;
```

Another example

```
if (...) {  
  a := read(); t := a + b  
  x := a + b; t  
  print(x);  
} else { t := a + b; }  
  
...  
y := a + b; t
```

Partial Redundancy Elimination PRE

Another example

```
x := y  
...  
z := z + x
```

Another example

```
x := y  
...  
z := z + x * y
```

*x, y not modified
Copy prop*

Another example

```
x := y  
...  
z := z + y
```

What if we run CSE now?

Another example

```
x := y  
...  
z := z + x
```

What if we run CSE now?

Another example

```
x := y**z  
...  
x := ...
```

Another example

```
x := y**z
...
x := ...
```

*if x is not used
dead assignment elim
(unused assignment elim)*

- Often used as a clean-up pass

```
x := y      Copy prop   x := y      DAE   x := y
z := z + x  →          z := z + y →   z := z + y
```

Another example

```
if (false) {
    ...
}
```

Another example

```
if (false) {
    ...
}
```

*dead code elim
(unreachable code elim)
Another common clean up opt*

Another example

- In Java:

```
a = new int [10];
for (index = 0; index < 10; index++) {
    a[index] = 100;
}
```

Another example

- In “lowered” Java:

```
a = new int [10];
for (index = 0; index < 10; index++) {
    if (index < 0 || index >= a.length()) {
        throw OutOfBoundsException;
    }
    a[index] = 0;
}
```

Another example

- In “lowered” Java:

```
a = new int [10];
for (index = 0; index < 10; index++) {
    if (index < 0 || index >= a.length()) {
        throw OutOfBoundsException;
    }
    a[index] = 0;
}
```

Branch folding + unreachable code elim
index ∈ [0..9] ← Range analysis
10 ← Kinda like CP if we assume that 0 acts like a.length := 10

Another example

```
p := &x;  
*p := 5  
y := x + 1;
```

Another example

```
p := &x;  
*p := 5  
y := x + 1; 6
```

pointer/alias analysis

```
x := 5;  
*p := 3  
y := x + 1; → ???
```

Another example

```
for j := 1 to N  
  for i := 1 to M  
    a[i] := a[i] + b[j]
```

Another example

```
for j := 1 to N t := b[j]  
  for i := 1 to M  
    a[i] := a[i] + b[j] t
```

*Loop invariant
Code motion*

Another example

```
area(h,w) { return h * w }  
  
h := ...;  
w := 4;  
a := area(h,w)
```

Another example

```
area(h,w) { return h * w }  
  
h := ...;  
w := 4;  
a := area(h,w)  
h * w  
h * 4  
h << 2
```

*Many "silly" opts become
important after inlining*

Optimization themes

- Don't compute if you don't have to
 - unused assignment elimination
- Compute at compile-time if possible
 - constant folding, loop unrolling, inlining
- Compute it as few times as possible
 - CSE, PRE, PDE, loop invariant code motion
- Compute it as cheaply as possible
 - strength reduction
- Enable other optimizations
 - constant and copy prop, pointer analysis
- Compute it with as little code space as possible
 - unreachable code elimination