

Tour of common optimizations

How excited are you about this course?

- A. Super excited
- B. A little excited
- C. Not that excited
- D. Not at all excited

How nervous are you about this course?

- A. Super nervous
- B. A little nervous
- C. Not that nervous
- D. Not at all nervous

What is your primary reason for 231?

- A. I'm doing research and compilers and related areas, so I want to learn about compilers
- B. I'm not doing research in this area, but still want to learn about compilers
- C. A friend recommended it
- D. I want to only take AI and Machine learning courses, but the program requires me to take other classes too, so here I am. Ugh
- E. Other

Simple example

```
foo(z) {  
    x := 3 + 6 * 9  
    y := x - 5 * 4  
    return z * 24  
}
```

Simple example

```
foo(z) {  
    x := 3 + 6; g  
    Cant prop (CP) g  
    y := x - 5 4 (CF)  
    return z * y  
    z << 2 4 (CP)  
}
```

Strength reduction

Constant folding (CF)

Arith simpl

Another example

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

```
...
```

```
y := ax + 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);  
}  
  
1 x := a + b  
1 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 + ~~z~~ y

Another example

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

} x,y not modified
copy prop

Another example

$x := E$
 $x := y$
...
 $z := z + y$ $E \rightarrow x$

What if we run CSE now?

$x := E$
.
:
 $E \rightarrow x$

Another example

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

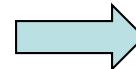
x := y
z := z + x

Copy prop



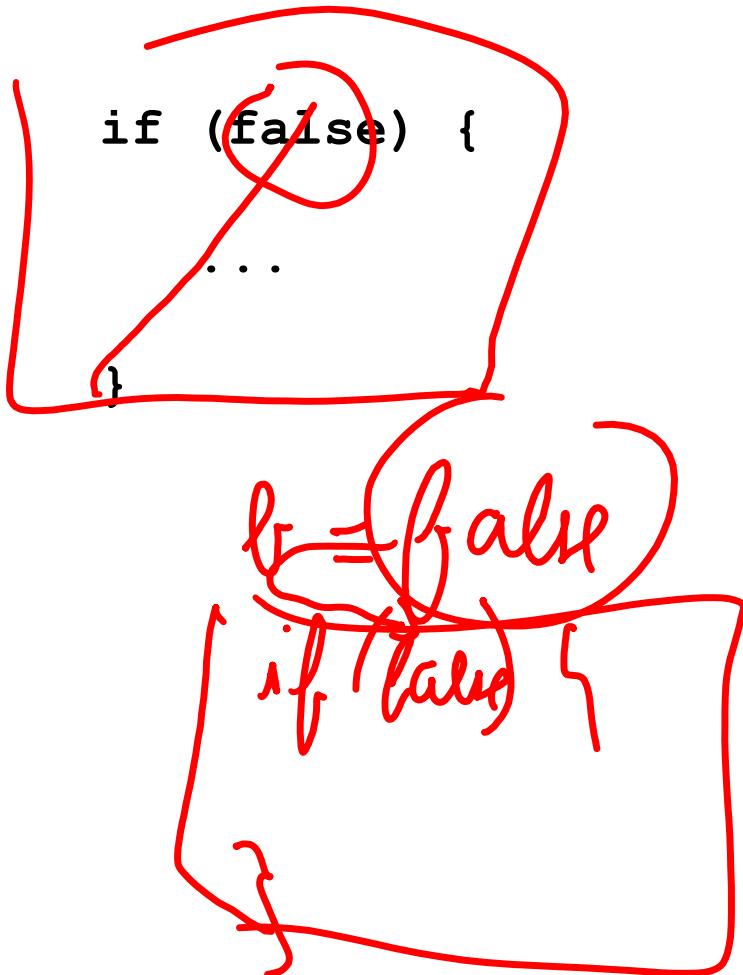
x := y
z := z + y

DAE



~~x := y~~
z := z + y

Another example



A hand-drawn diagram illustrating nested function definitions. It consists of two nested rounded rectangles. The inner rectangle contains the pseudocode:

```
f (b) {  
    if (b) {  
        ...  
    }  
}
```

The outer rectangle contains the pseudocode:

```
}
```

This illustrates that the function `f(b)` contains another function definition for `if (b)`, which itself has its own closing brace.

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

Handwritten annotations in red:

- A red box encloses the entire code block.
- The condition `if (index < 0) || (index >= a.length())` is circled with a red circle. Inside the circle, the word "false" is written above the first part and "true" below the second part.
- Red arrows point from the words "false" and "true" to the corresponding parts of the condition: "false" points to `index < 0` and "true" points to `index >= a.length()`.
- To the right of the condition, the words "index > 0" and "index < 10" are written vertically, connected by a red arrow pointing upwards.

Another example

- In “lowered” Java:

```
a = new int [10]; ①
for (index = 0; index < 10; index++) {
    if (index < 0 || index >= a.length()) {
        throw OutOfBoundsException;
    }
    a[index] = 0;
}
index ∈ {0..9} ← Range analysis
```

10 ← Kinda like CP
if we assume
stmt ① acts
like `a.length := 10`

10 ← Branch folding
+ unreachable
code elim

Another example

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

Another example

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

pointer / alias analysis

x := 5;
*p := 3
y := ~~z + 1;~~ 5  ???

Another example

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

A. Yes

B. No

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 "illy" 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