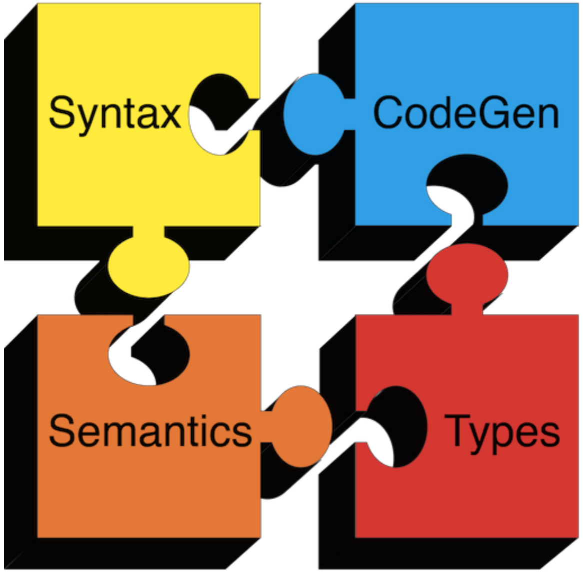

Compilers

Local Optimization

Alex Aiken

Local Optimization

- The simplest form of optimization

- Optimize one basic block
  - No need to analyze the whole procedure body

Local Optimization

- Some statements can be deleted

  x := x + 0

  x := x * 1

- Some statements can be simplified

x := x * 0	⇒	x := 0
y := y ** 2	⇒	y := y * y
x := x * 8	⇒	x := x << 3
x := x * 15	⇒	t := x << 4; x := t - x

(on some machines << is faster than *; but not on all!)

Local Optimization

- Operations on constants can be computed at compile time
  - If there is a statement x := y op z
  - And y and z are constants
  - Then y op z can be computed at compile time

- Example: x := 2 + 2 $\Rightarrow$ x := 4
- Example: if 2 < 0 jump L can be deleted

- Constant folding can be dangerous.

- Eliminate unreachable basic blocks:
  - Code that is unreachable from the initial block
    - E.g., basic blocks that are not the target of any jump or "fall through" from a conditional

- Removing unreachable code makes the program smaller
  - And sometimes also faster
    - Due to memory cache effects
    - Increased spatial locality

- Why would unreachable basic blocks occur?

Local Optimization

- Some optimizations are simplified if each register occurs only once on the left-hand side of an assignment

- Rewrite intermediate code in *single assignment* form

x := z + y		b := z + y
a := x	⇒	a := b
x := 2 * x		x := 2 * b

(b is a fresh register)

  – More complicated in general, due to loops

Local Optimization

- If
  - Basic block is in single assignment form
  - A definition x := is the first use of x in a block
- Then
  - When two assignments have the same rhs, they compute the same value
- Example:

```
x := y + z                    x := y + z
…                 ⇒           …
w := y + z                    w := x
```

(the values of x, y, and z do not change in the … code)

Local Optimization

- If w := x appears in a block, replace subsequent uses of w with uses of x
  - Assumes single assignment form

- Example:

b := z + y		b := z + y
a := b	$\Rightarrow$	a := b
x := 2 * a		x := 2 * b

- Only useful for enabling other optimizations
  - Constant folding
  - Dead code elimination

Local Optimization

- Example:

a := 5		a := 5
x := 2 * a	⇒	x := 10
y := x + 6		y := 16
t := x * y		t := x << 4

If

w := rhs appears in a basic block

w does not appear anywhere else in the program

Then

the statement w := rhs is dead and can be eliminated

– <u>Dead</u> = does not contribute to the program's result

Example: (a is not used anywhere else)

x := z + y		b := z + y		b := z + y
a := x	$\Rightarrow$	a := b	$\Rightarrow$	x := 2 * b
x := 2 * a		x := 2 * b		

- Each local optimization does little by itself

- Typically optimizations interact
  - Performing one optimization enables another

- Optimizing compilers repeat optimizations until no improvement is possible
  - The optimizer can also be stopped at any point to limit compilation time

- Initial code:

```
a := x ** 2
b := 3
c := x
d := c * c
e := b * 2
f := a + d
g := e * f
```

- Algebraic optimization:

```
a := x ** 2
b := 3
c := x
d := c * c
e := b * 2
f := a + d
g := e * f
```

- Algebraic optimization:

```
a := x * x
b := 3
c := x
d := c * c
e := b << 1
f := a + d
g := e * f
```

- Copy propagation:

```
a := x * x
b := 3
c := x
d := c * c
e := b << 1
f := a + d
g := e * f
```

- Copy propagation:

```
a := x * x
b := 3
c := x
d := x * x
e := 3 << 1
f := a + d
g := e * f
```

Local Optimization

- Constant folding:

```
a := x * x
b := 3
c := x
d := x * x
e := 3 << 1
f := a + d
g := e * f
```

- Constant folding:

a := x * x
b := 3
c := x
d := x * x
e := 6
f := a + d
g := e * f

- Common subexpression elimination:

```
a := x * x
b := 3
c := x
d := x * x
e := 6
f := a + d
g := e * f
```

- Common subexpression elimination:

```
a := x * x
b := 3
c := x
d := a
e := 6
f := a + d
g := e * f
```

- Copy propagation:

```
a := x * x
b := 3
c := x
d := a
e := 6
f := a + d
g := e * f
```

- Copy propagation:

```
a := x * x
b := 3
c := x
d := a
e := 6
f := a + a
g := 6 * f
```

- Dead code elimination:

```
a := x * x
b := 3
c := x
d := a
e := 6
f := a + a
g := 6 * f
```

Local Optimization

- Dead code elimination:

$$a := x * x$$

$$f := a + a$$
$$g := 6 * f$$

- This is the final form

Local Optimization

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := a + x
4   d := a * 3
5   e := b * 3
6   f := a + b
7   g := e - f
```

- ☐ Copy propagation: Line 4 becomes d := a * b.
- ☐ Common subexpression elimination: Line 5 becomes e := d.
- ☐ Dead code elimination: Line 3 is removed.
- ☐ After many rounds of valid optimizations, the entire block can be reduced to g := 5.