Compiler Construction

BITS Pilani
Pilani Campus

Vinti Agarwal
March 2021

innovate achieve lead

BITS Pilani
Pilani Campus

CS F363, Compiler Construction

Lecture topic: Local Optimization

In previous lecture

- Some statements can be deleted

✗

$x = x + 0$

- Some statements can be simplified

$x := x * 0$

$x := 0$

- Constant folding ✓

✓

$x := 2 + 2$

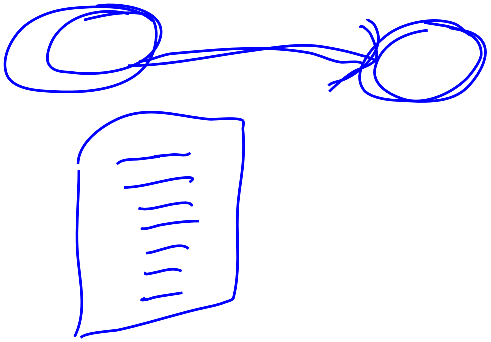
$x := 4$

Eliminate unreachable basic blocks

- Code that is unreachable from the initial basic block

  e.g. basic blocks that are not target of any “jump” or fall through from a “conditional”

- Removing unreachable block makes program smaller and sometimes also faster
  - due to memory cache effects
  - increase spatial locality



Why would unreachable basic blocks occur?

define DEBUG 0 ←

constant folding

If DEBUG Then

use of libraries

1000

3

997

e.g. constant folding

Single assignment form of Intermediate code

- Some optimization 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
a := x
x := 2 * x
```

≠

- b := z+y
- a := b
- x := 2 * ~~a~~b

Common Subexpression Elimination

if

- A basic block is in single assignment form
- A def **x :=** is the first use of **x** in the block

Then

- when two assignments have same **rhs**, they compute same value

example:

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

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

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

Copy propagation ✓

- if **w := x** appears in a block, replace subsequent uses of **w** with uses of **x**
    - assume single assignment form
- Example

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

⇒ copy prop

```
b := z + y
a := b ✓ X
x := 2 * b
```

⇒ dead code elimination

- useful for enabling other optimizations
    - ○ constant folding ✓
    - ○ dead code elimination ✓

Example:

→ Copy Propagation.

⇓

constant propagation.

```
1. a := 5
2. x := 2 * a
3. y := x + 6
4. t := x * y
```

⇒

```
a := 5
x := 10
y := 16
t := x*16
t := x<<4 .
```

Dead Code Elimination

if

w := rhs appears in a basic block

w does not appear anywhere else in the program

then

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

dead statements do not contribute to the program

Example:

x := z + y

a := x ⇒

x := 2 * a

b := z + y

a := b ⇒

x := 2 * a

b := z + y

a := b

x := 2 * b

⇓

b := z + y

$x := 2 * b$

Repeat Optimization

- Each local optimization does little by itself

- Typically optimizations interact
  - performing one optimization enables other

- Optimizing compiler repeat optimizations until no improvements are possible
  - The optimizers can be stop at any point to limit compilation time.

Initial Code

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

Algebraic Optim.

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

Copy propa.

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

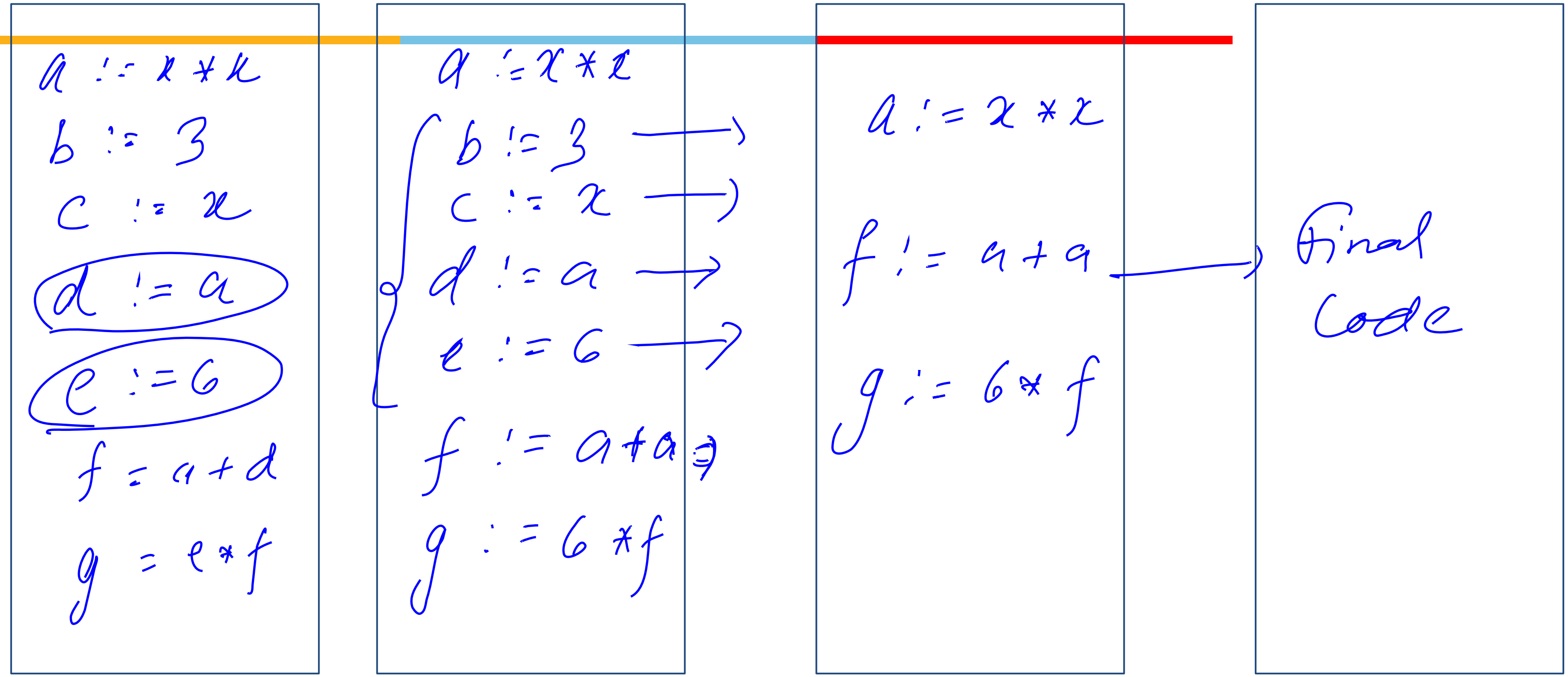
Const folding

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

SubExp Eli

Copy Prop.

Dead Code Elim.



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

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

```
a := x * x

f := a + a

g := 6 * f
```

Final Code

Example

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
```

Option 1: Copy propagation: Line 4 becomes d := a * b ✗

Option 2: Common subexpression elimination: Line 5 becomes e := d ✗

Example cont’d

Option 3: Dead code elimination: Line 3 is removed ✓

Option 4: After many rounds of valid optimizations, the entire block can be reduced to g := 5 ?

check on your own.

Thank You!