

IR Optimization

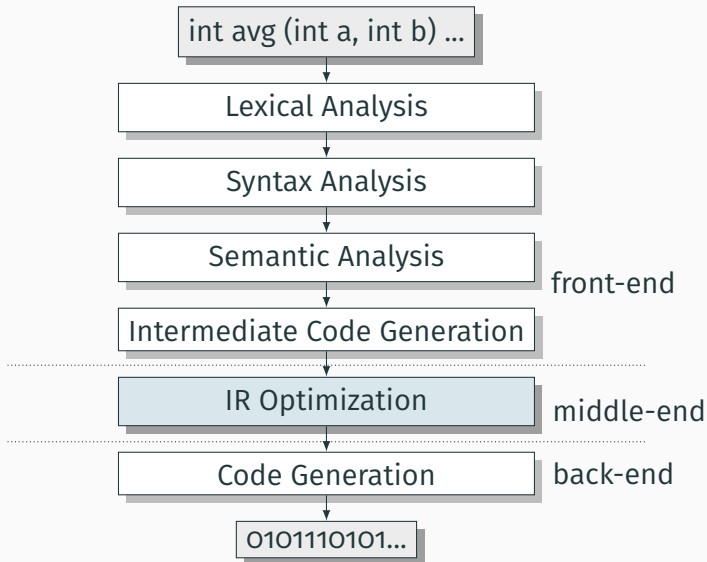
Ronghui Gu

Spring 2019

Columbia University

* Course website: <https://www.cs.columbia.edu/~rgu/courses/4115/spring2019>

IR Optimization



Goal

- Runtime
- Memory usage
- Power Consumption

Sources?

Optimizations from IR Generation

C code:

```
int x;  
int y;  
bool b1;  
bool b2;  
bool b3;  
b1 = x + x < y  
b2 = x + x == y  
b3 = x + x > y
```

Three-Address:

```
_t0 = x + x;  
_t1 = y;  
b1 = _t0 < _t1;  
_t2 = x + x;  
_t3 = y;  
b2 = _t2 == _t3;  
_t4 = x + x;  
_t5 = y;  
b3 = _t5 < _t4;
```

Optimizations from IR Generation

C code:

```
int x;  
int y;  
bool b1;  
bool b2;  
bool b3;  
b1 = x + x < y  
b2 = x + x == y  
b3 = x + x > y
```

Three-Address:

```
_t0 = x + x;  
_t1 = y;  
b1 = _t0 < _t1;  
_t2 = x + x;  
_t3 = y;  
b2 = _t2 == _t3;  
_t4 = x + x;  
_t5 = y;  
b3 = _t5 < _t4;
```

Optimizations from IR Generation

C code:

```
int x;  
int y;  
bool b1;  
bool b2;  
bool b3;  
b1 = x + x < y  
b2 = x + x == y  
b3 = x + x > y
```

Three-Address:

```
_t0 = x + x;  
_t1 = y;  
b1 = _t0 < _t1;  
  
b2 = _t0 == _t1;  
  
b3 = _t0 > _t1;
```

Optimizations from Lazy Coders

C code:

```
while (x < y + z) {  
    x = x - y;  
}
```

Three-Address:

```
_Lo:  
    _t0 = y + z;  
    _t1 = x < _t0;  
    bz _L1 _t1;  
    x = x - y;  
    jmp _Lo;  
_L1:
```

Optimizations from Lazy Coders

C code:

```
while (x < y + z) {  
    x = x - y;  
}
```

Three-Address:

```
_Lo:  
    _to = y + z;  
    _t1 = x < _to;  
    bz _L1 _t1;  
    x = x - y;  
    jmp _Lo;  
_L1:
```


Optimizations from Lazy Coders

C code:

```
while (x < y + z) {  
    x = x - y;  
}
```

Three-Address:

```
    _t0 = y + z;  
_Lo:  
    _t1 = x < _t0;  
    bz _L1 _t1;  
    x = x - y;  
    jmp _Lo;  
_L1:
```

Optimal? Undecidable!

Soundness: semantics-preserving

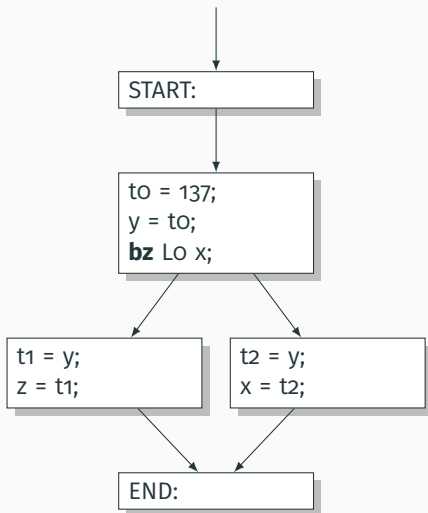
IR optimization v.s. code optimization:

$$x * 0.5 \Rightarrow x \gg 1$$

Local optimization v.s. global optimization

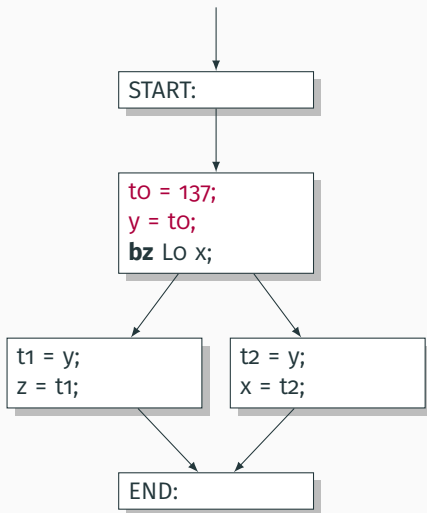
Local Optimization

```
int main() {  
    int y;  
    int z;  
    y = 137;  
    if (x == 0)  
        z = y;  
    else  
        x = y;  
}
```



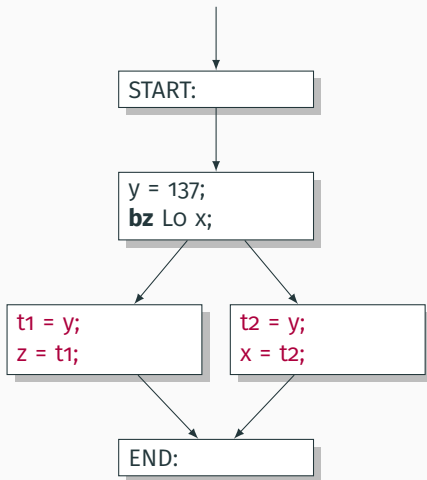
Local Optimization

```
int main() {  
    int y;  
    int z;  
    y = 137;  
    if (x == 0)  
        z = y;  
    else  
        x = y;  
}
```



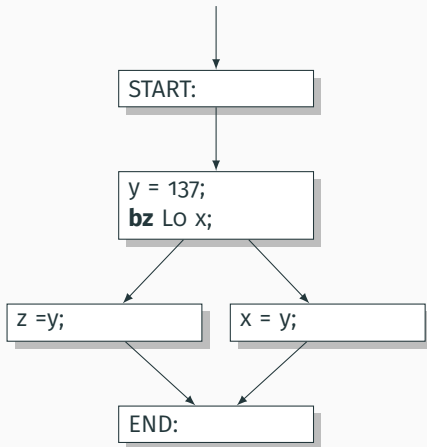
Local Optimization

```
int main() {  
    int y;  
    int z;  
    y = 137;  
    if (x == 0)  
        z = y;  
    else  
        x = y;  
}
```



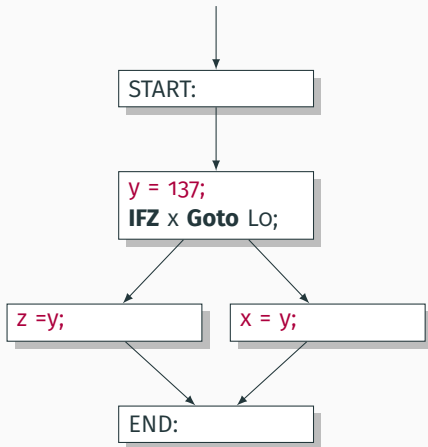
Local Optimization

```
int main() {  
    int y;  
    int z;  
    y = 137;  
    if (x == 0)  
        z = y;  
    else  
        x = y;  
}
```



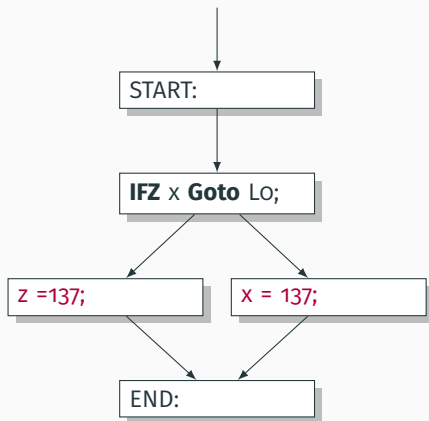
Global Optimization

```
int main() {  
    int y;  
    int z;  
    y = 137;  
    if (x == 0)  
        z = y;  
    else  
        x = y;  
}
```



Global Optimization

```
int main() {  
    int y;  
    int z;  
    y = 137;  
    if (x == 0)  
        z = y;  
    else  
        x = y;  
}
```



Local Optimization

Common Subexpression Elimination

```
v1 = a op b
```

```
. . .
```

```
v2 = a op b
```

If values of **v1**, **a**, and **b** have not changed, rewrite the code:

```
v1 = a op b
```

```
. . .
```

```
v2 = v1
```

Common Subexpression Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = _t0;  
_t1 = a + b;  
c = _t1;  
_t2 = a + b;  
param _t2  
call f;
```

Common Subexpression Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = _t0;  
_t1 = a + b;  
c = _t1;  
_t2 = a + b;  
param _t2  
call f;
```

Common Subexpression Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = _t0;  
_t1 = a + b;  
c = _t1;  
_t2 = _t1;  
param _t2  
call f;
```

Common Subexpression Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = _t0;  
_t1 = a + b;  
c = _t1;  
_t2 = c;  
param _t2  
call f;
```

Copy Propagation

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_to = 4;  
a = _to;  
_t1 = a + b;  
c = _t1;  
_t2 = c;  
param _t2  
call f;
```

Copy Propagation

If we have

v1 = v2

then as long as **v1** and **v2** have not changed, we can rewrite

a = ... v1 ...

as

a = ... v2 ...

Copy Propagation

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = _t0;  
_t1 = a + b;  
c = _t1;  
_t2 = c;  
param _t2  
call f;
```

Copy Propagation

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = 4;  
_t1 = a + b;  
c = _t1;  
_t2 = c;  
param _t2  
call f;
```

Copy Propagation

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = 4;  
_t1 = 4 + b;  
c = _t1;  
_t2 = c;  
param _t2  
call f;
```

Copy Propagation

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = 4;  
_t1 = 4 + b;  
c = _t1;  
_t2 = c;  
param _t2  
call f;
```

Copy Propagation

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = 4;  
_t1 = 4 + b;  
c = 4 + b;  
_t2 = c;  
param _t2  
call f;
```

Copy Propagation

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = 4;  
_t1 = 4 + b;  
c = 4 + b;  
_t2 = c;  
param _t2  
call f;
```

Copy Propagation

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = 4;  
_t1 = 4 + b;  
c = 4 + b;  
_t2 = 4 + b;  
param _t2  
call f;
```

Dead Code Elimination

An assignment to a variable v is called **dead** if its value is **never** read anywhere.

Dead Code Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = 4;  
_t1 = 4 + b;  
c = 4 + b;  
_t2 = 4 + b;  
param _t2  
call f;
```

Dead Code Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_to = 4;  
a = 4;  
_t1 = 4 + b;  
c = 4 + b;  
_t2 = 4 + b;  
param _t2  
call f;
```

Dead Code Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
a = 4;  
_t1 = 4 + b;  
c = 4 + b;  
_t2 = 4 + b;  
param _t2  
call f;
```

Dead Code Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t1 = 4 + b;  
c = 4 + b;  
_t2 = 4 + b;  
param _t2  
call f;
```

Dead Code Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
c = 4 + b;  
_t2 = 4 + b;  
param _t2  
call f;
```

Dead Code Elimination

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t2 = 4 + b;  
param _t2  
call f;
```

For Comparison

C code:

```
int a;  
int b;  
int c;  
a = 4;  
c = a + b;  
f(a + b);
```

Three-address code:

```
_t0 = 4;  
a = _t0;  
_t1 = a + b;  
c = _t1;  
_t2 = a + b;  
param _t2  
call f;
```

Optimized code:

```
_t2 = 4 + b;  
param _t2  
call f;
```

Other Types of Local Optimization

Arithmetic implication:

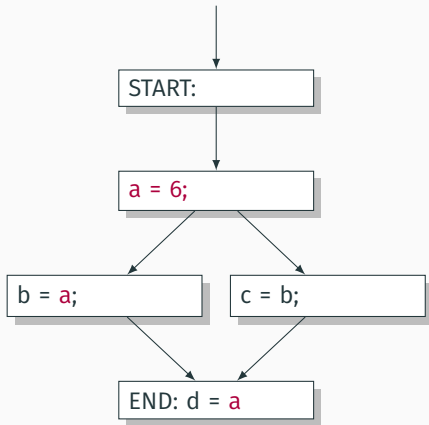
- e.g., rewrite $x = 4 * a$ as $x = a \ll 2$

Constant folding:

- e.g., rewrite $x = 4 * 5$ as $x = 20$

Global Optimization

Global Constant Propagation

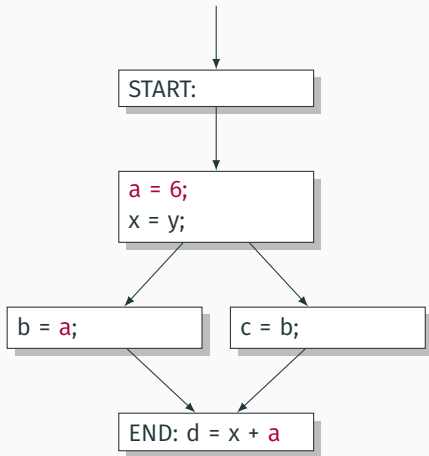


Global Optimization

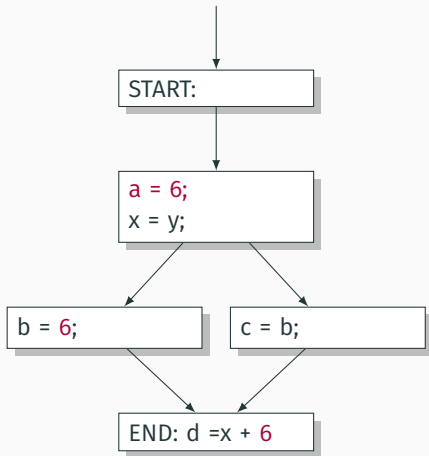
Global Constant Propagation

Replace each variable that is known to be a **constant** value with the constant.

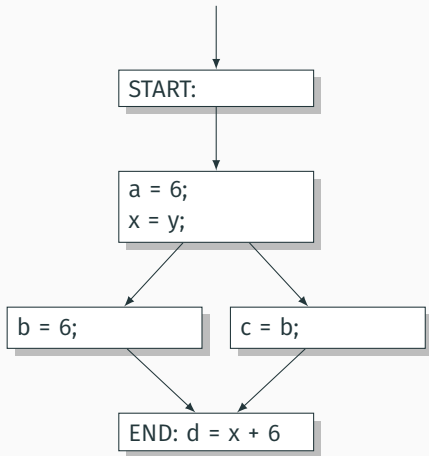
Global Constant Propagation



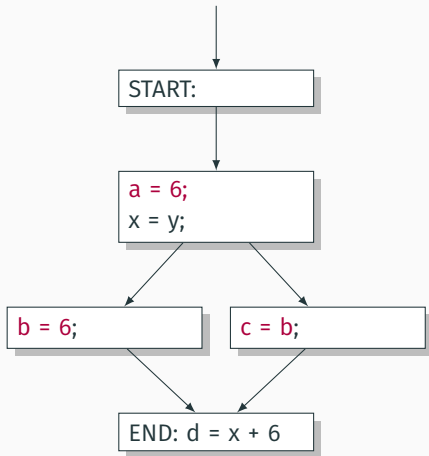
Global Constant Propagation



Global Dead Code Elimination



Global Dead Code Elimination



Global Dead Code Elimination

