EC3204: Programming Languages and Compilers

Lecture 16 — Optimization (1)

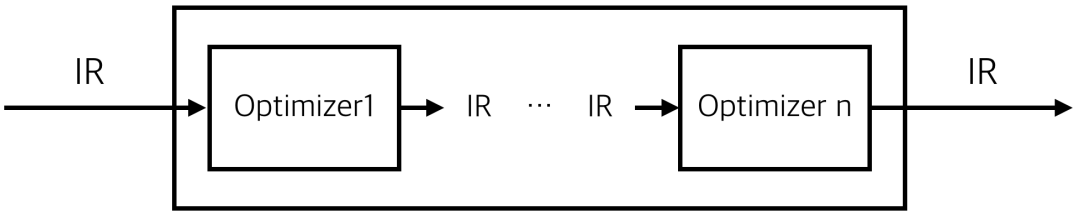
Introduction

Sunbeom So

Fall 2024

Middle-End: Optimizer

Converts the intermediate code (with many temporaries) into a more efficient yet semantically equivalent code.



Example:

```
t1 = 10
t2 = rate * t1
t3 = init + t2
pos = t3
```

original IR

```
t1 = 10
t2 = rate * 10
t3 = init + t2
pos = t3
```

```
t2 = rate * 10
t3 = init + t2
pos = t3
```

```
t2 = rate * 10
pos = init + t2
```

final IR

Commonly Used Optimizations

- Common subexpressions elimination
- Copy propagation
- Deadcode elimination
- Constant folding

Common Subexpression Elimination

- An occurrence of an expression E is called a **common subexpression** if E was previously computed and the values of the variables in E have not changed after the previous computation.

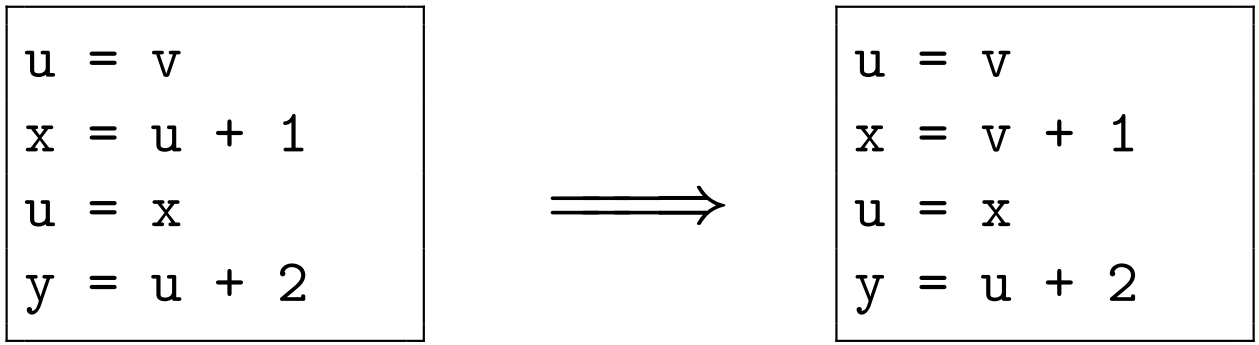
```
x = k * 2
...          // no defs to k
y = k * 2
```

- We can avoid recomputing E by replacing E by the variable that holds the previous value of E.

```
x = k *2
...          // no defs to k
y = x
```

Copy Propagation

- After the copy statement $u = v$, use v for u unless u is redefined.



Q. Each #instruction is the same. Why do we apply this?

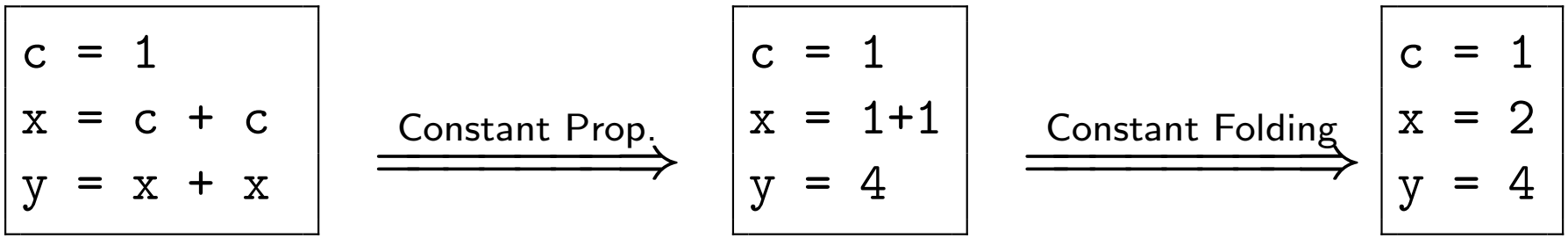
Deadcode Elimination

- A variable is **live** at a point in a program if its value is used eventually; otherwise it is **dead** at that point.
- A statement is said to be **deadcode** if it computes values that never get used.

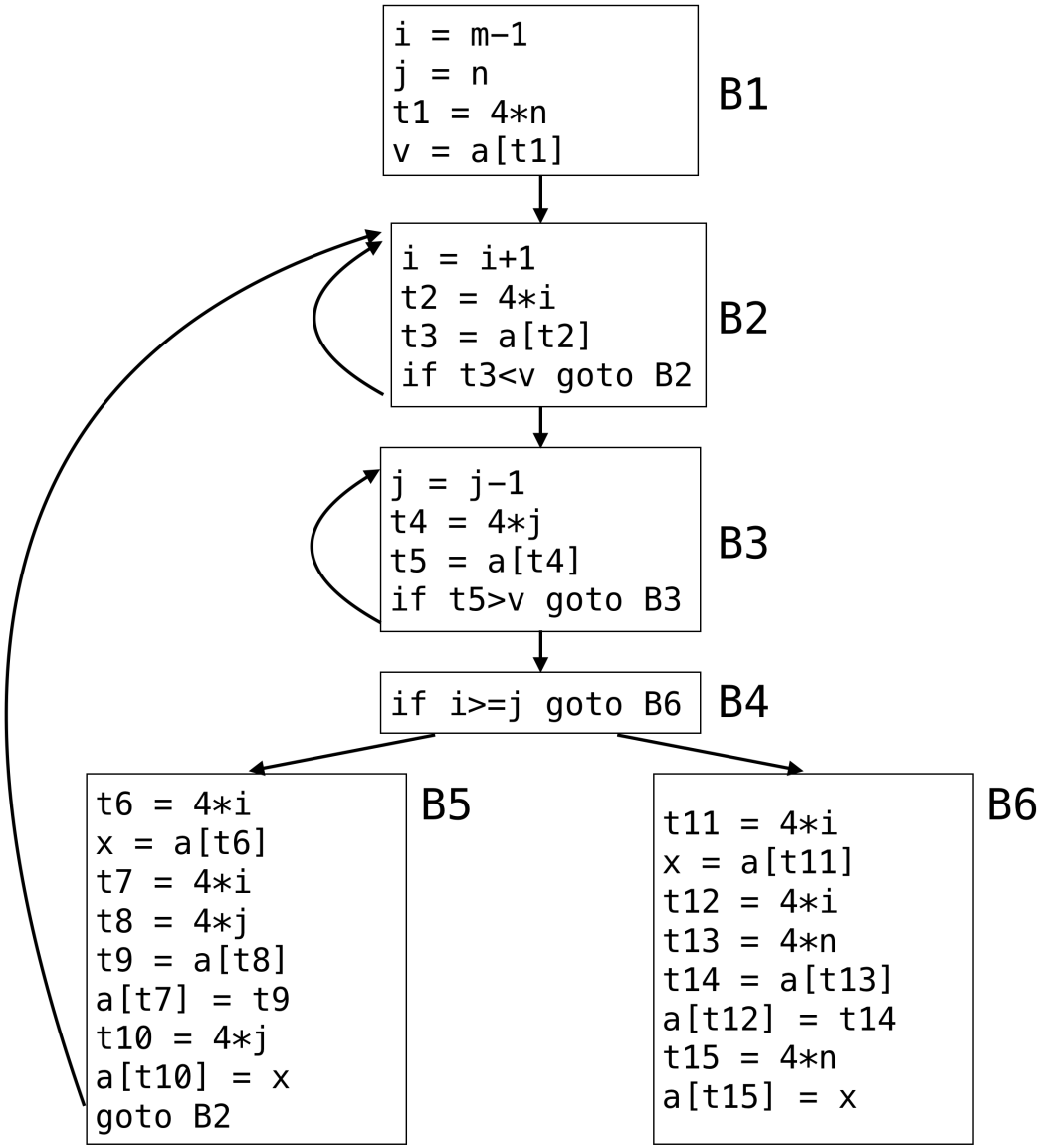
```
u = v       // deadcode
x = v + 1
u = x
y = u + 2
```

Constant Folding

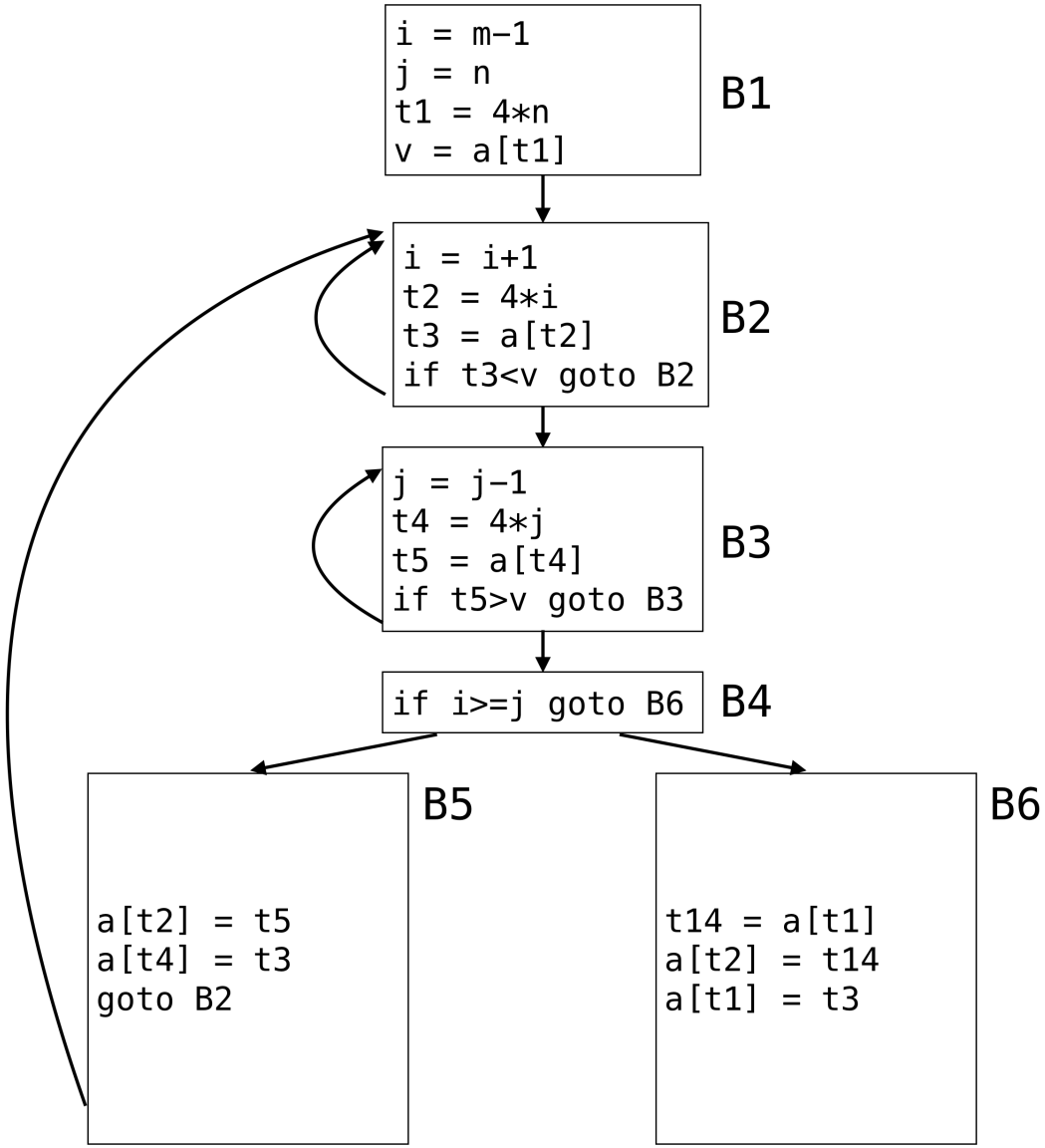
Decide that the value of an expression is a constant and use the constant instead.



Example: Original Program



Example: Optimized Program



Static analysis is Needed

To optimize a program, we need static analysis that derives information about the flow of data along program execution paths. Examples:

- Do the two textually identical expressions evaluate to the same value along any possible execution path of the program?
  If so, we can apply common subexpression elimination.
- Is the result of an assignment not used along any subsequent execution path?
  If so, we can apply deadcode elimination.

Summary

Introduction to code optimization:

- Code transformation to have better performance.
- Execution of transformed code must produce the same results with respect to all possible executions of the original code.

Next class: static analysis (data-flow analysis) for code optimization