LANGUAGE PROCESSORS

UNIT 10: CODE OPTIMIZATION

David Griol Barres
dgriol@inf.uc3m.es
Computer Science Department
Carlos III University of Madrid
Leganés (Spain)

OUTLINE

- Introduction
- Code optimization
- Basic Blocks
- Where?
  - Local Optimizations
    - Constant folding
    - Constant propagation
    - Algebraic simplification and re-association
    - Strength Reduction
    - Other Local Optimizations
  - Global optimizations
    - Live Variable Analysis

Introduction

- Ideally, compilers should produce target code that is good as can be written by hand, but rarely that is the case.
- **OBJECTIVE:** Transform a piece of code to make it more efficient without changing its output (execution speed and memory requirements).
- One of the most interesting topics in compiler research.
- Optimization should preserve the meaning of programs.
- More an art than a science.

Code optimization

- Principles of design:
  - Correctness above all.
  - Application: Intermediate or target code.
  - Efficiency.
  - Control-flow analysis.

Basic Blocks

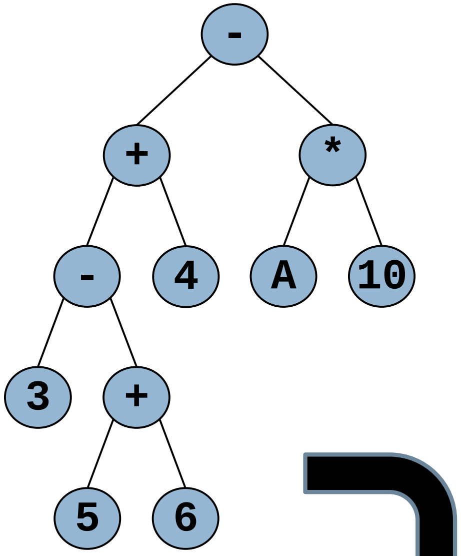
- A basic block is a segment of the code that has exactly one entry point and one exit point.
- A basic block begins in one of several ways:
  - The entry point into the function.
  - The target of a branch (often a label).
  - The instruction following a branch or a return.
- A basic block ends in any of the following ways:
  - A jump statement.
  - A conditional or unconditional branch.
  - A return statement.

Where?

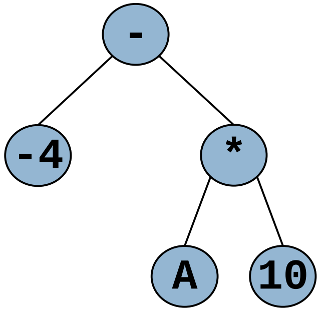
- Local optimizations (within a basic block)
  1. Constant folding
  2. Constant propagation
  3. Algebraic simplification and reassociation
  4. Operator strength reduction
  5. Copy propagation
  6. Dead code elimination
  7. Common subexpression elimination
  - …
- Global optimizations. Data flow analysis

Local Optimizations: Constant folding

- Expression: 3-(5+6)+4-A*10



- Result: -4-(A*10)

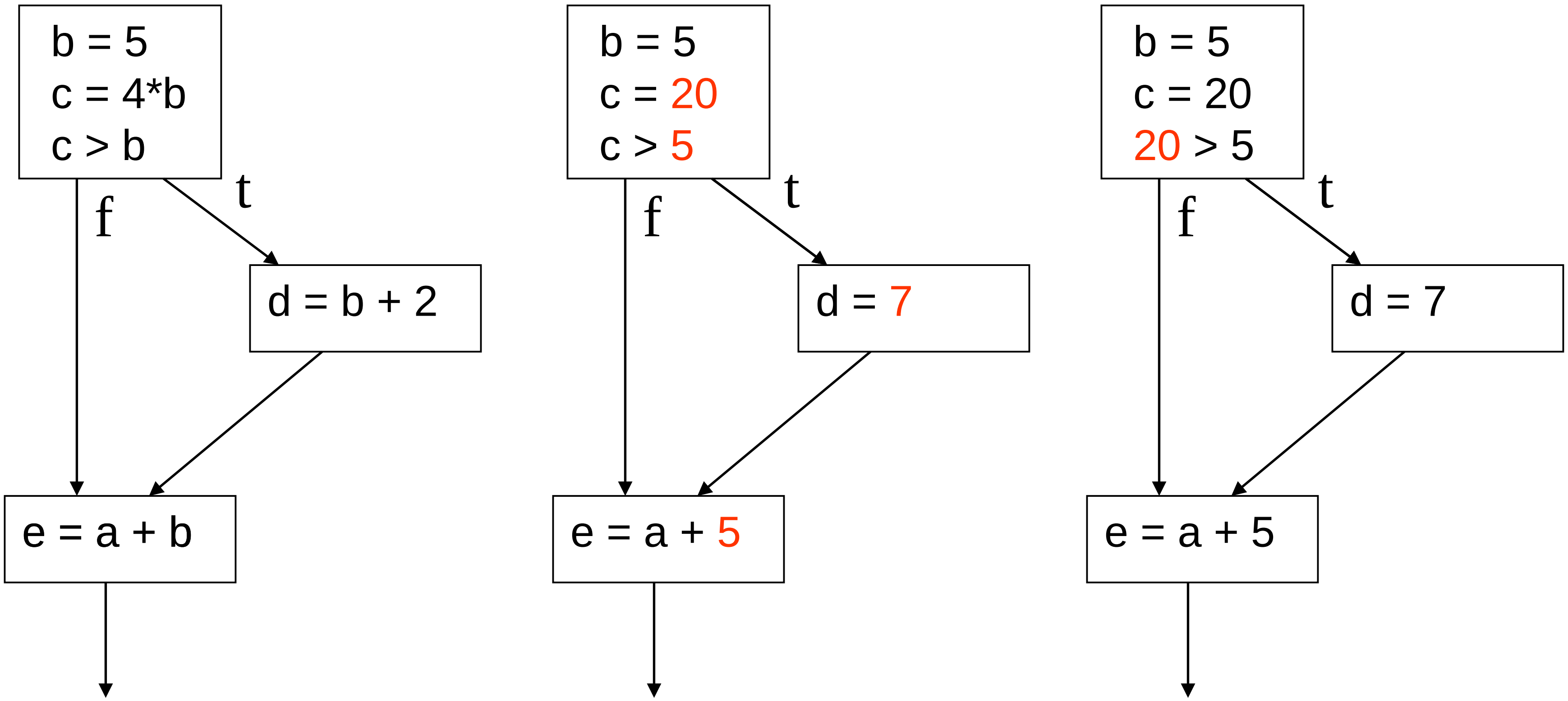


Evaluation at compile-time of expressions whose operands are known to be constant.

Local Optimizations: Constant propagation

- If a variable is assined a constant value:
  - The subsequent uses of that variable can be replaced by the constant as long as no intervening assignment has changed the value of the variable.

Local Optimizations: Constant propagation



b = 5
c = 4*b
c > b

f

t

d = b + 2

e = a + b

b = 5
c = 20
c > 5

f

t

d = 7

e = a + 5

b = 5
c = 20
20 > 5

f

t

d = 7

e = a + 5

Local Optimizations: Algebraic simplification and re-association

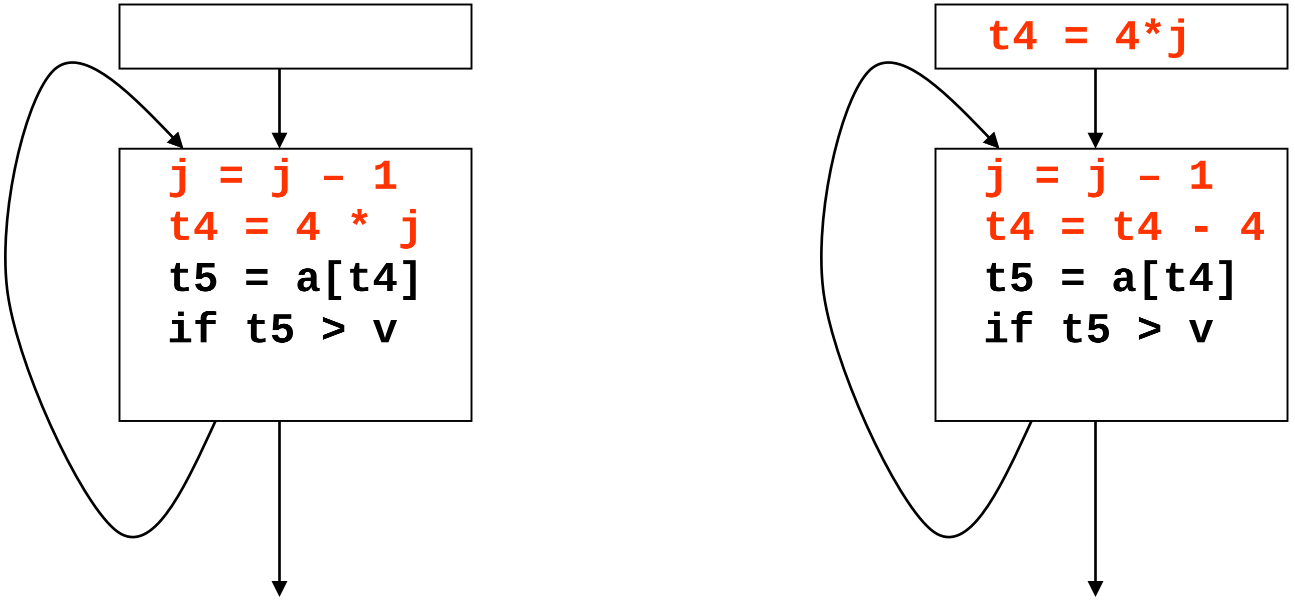
An expression **x op y** is redundant at a point p if it has already been computed at some point(s) and no intervening operations redefine **x** or **y**.

```
m = 2*y*z         t0 = 2*y          t0 = 2*y
                  m = t0*z          m = t0*z
n = 3*y*z         t1 = 3*y          t1 = 3*y
                  n = t1*z          n = t1*z
o = 2*y-z         t2 = 2*y
                  o = t2-z          o = t0-z
```

redundant

Local Optimizations: Strength Reduction

- Replaces an operator by a "less-expensive" one:
- Example: Induction Variables in control loop iterations



```
j = j - 1
t4 = 4 * j
t5 = a[t4]
if t5 > v
```

```
t4 = 4*j
```

```
j = j - 1
t4 = t4 - 4
t5 = a[t4]
if t5 > v
```

Local Optimizations: Strength Reduction

```
while (i <=  limit - 2)
```

```
L1:
    t1 = limit – 2
    if (i > t1) goto L2
    body of loop
    goto L1
L2:
```

```
t := limit - 2
   while (i <= t)
```

```
    t1 = limit – 2
L1:
    if (i > t1) goto L2
    body of loop
    goto L1
L2:
```

Other Local Optimizations

- **Copy Propagation:** Generalization of the constant propagation.
  - Example: a=b → Replace the occurrences of a with b.

- **Dead Code elimination:** Eliminate instructions that are never used.

- **Common subexpression elimination:** Instructions that produce the same result (eliminate or unify code for not computing again the same result).

Global optimizations

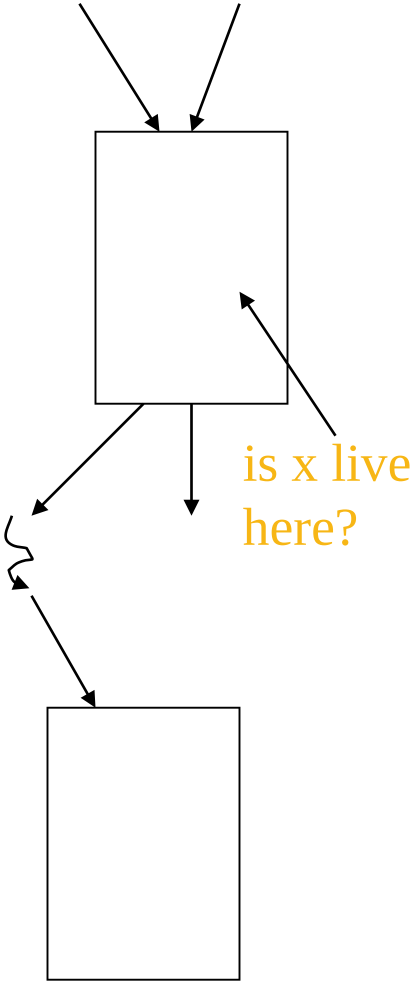
- Apply similar optimizations across basic blocks. Usually one function at a time (Data-flow analysis).

- Each block is a node in the flow graph of a program

Live Variable Analysis

A variable ***x*** is *live* at a point p if there is some path from *p* where ***x*** is used before it is defined.

Want to determine for some variable *x* and point *p* whether the value of *x* *could* be used along some path starting at p.



Global Live Variable Analysis

- **Code motion:** Unify code common to one or more basic blocks to reduce the code size and re-evaluations.

- **Machine optimizations:** Take into account specific machines features → code optimized for that machine.

- **Register allocation:** Minimize traffic between registers and memory → Register coloring.