CC Lecture 5

Prepared for: 7th Sem, CE, DDU

Prepared by: Niyati J. Buch

Optimizing Transformations

1. Compile time evaluation
2. **Elimination of common subexpression**
3. Dead code elimination
4. Frequency reduction
5. Strength reduction

Global Common Subexpression Elimination

- **Global common-subexpression elimination** takes as its scope a flowgraph representing a procedure.

- It solves the data-flow problem known as **available expressions**.

- An **expression exp** is said to be **available** at the entry to a basic block if along every control-flow path from the entry block to this block there is an evaluation of exp that is not subsequently killed by having one or more of its operands assigned a new value.

Global Common Subexpression Elimination

- In determining what **expressions are available**, we use
  - **EVAL(i)** to denote the set of expressions evaluated in block i that are still available at its exit
  - **KILL(i)** to denote the set of expressions that are killed by block i.

EVAL(i)

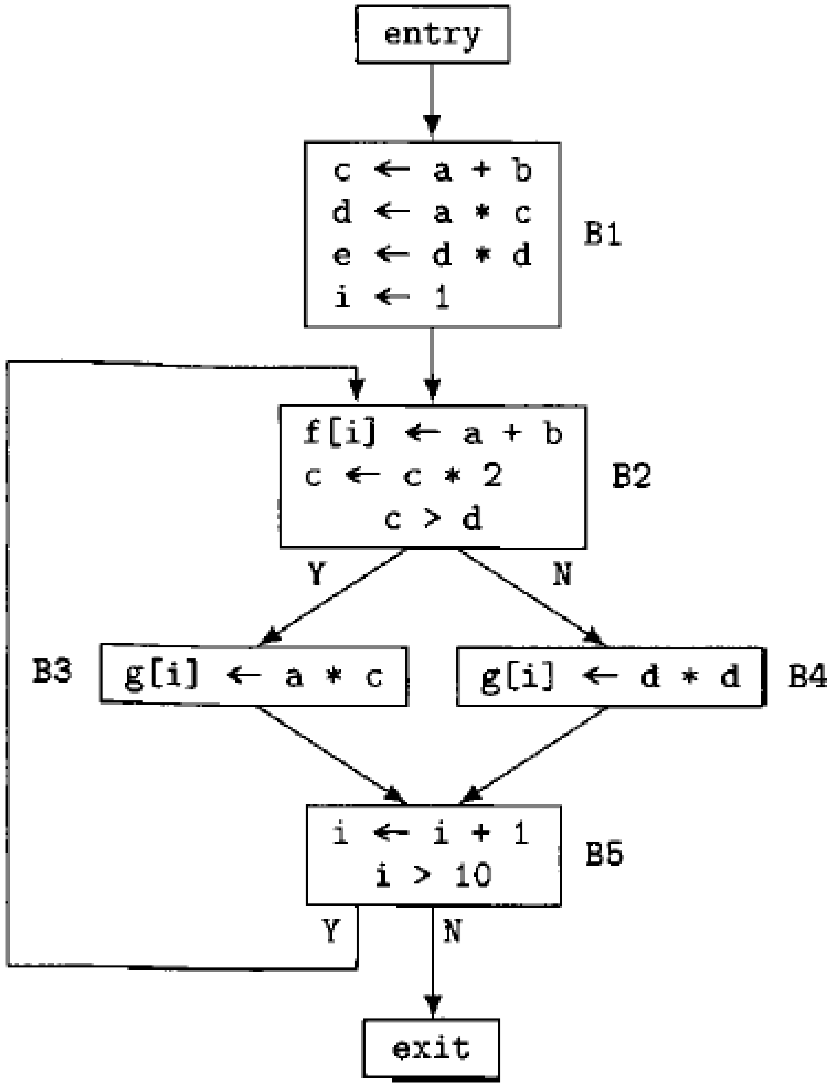
- To compute **EVAL(i)**,
  - we scan block i from beginning to end,
  - accumulating the expressions evaluated in it and
  - deleting those whose operands are later assigned new values in the block.

Note: An assignment such as a $\leftarrow$ a + b, in which the variable on the left-hand side occurs also as an operand on the right-hand side, does not create an available expression because the assignment happens after the expression evaluation.

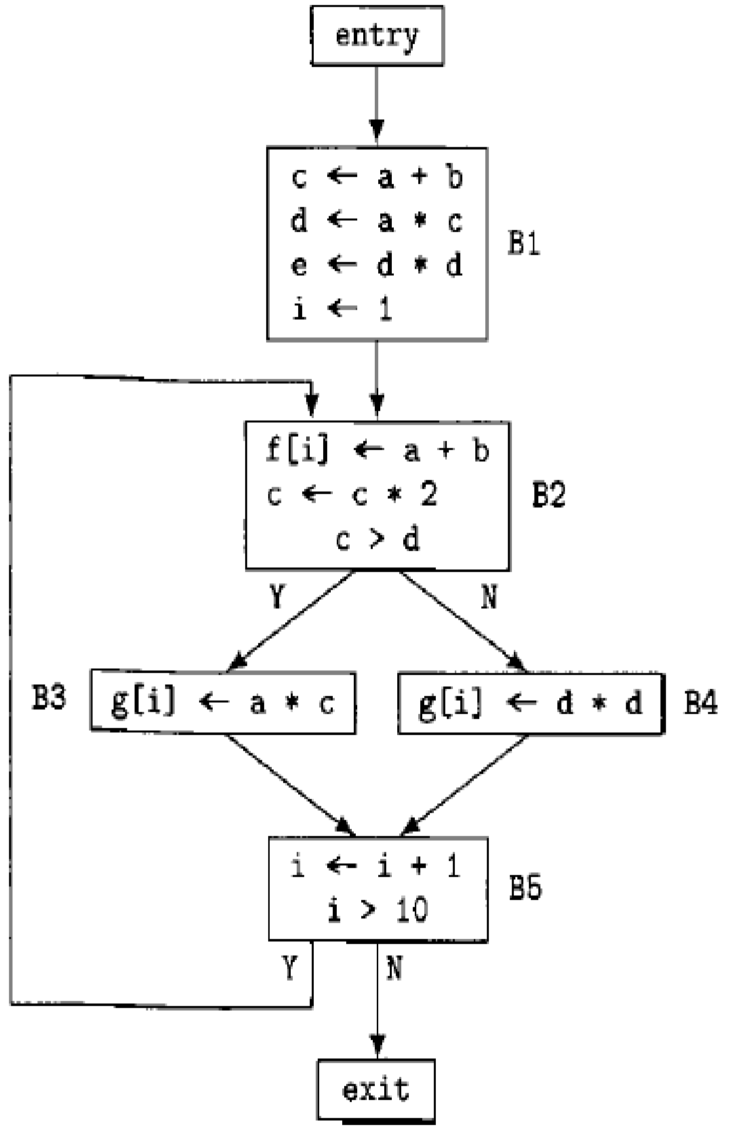
KILL(i)

- **KILL(i)** is the set of all expressions
  - evaluated in **other blocks** such that one or more of their operands are **assigned to in block** i,
  - or that are evaluated in **block** i and subsequently have an operand **assigned to in block** i.

Example: Given a flow graph



Find EVAL(i) sets for the basic blocks



- EVAL(entry) = ?
- EVAL(B1) = ?
- EVAL(B2) = ?
- EVAL(B3) = ?
- EVAL(B4) = ?
- EVAL(B5) = ?
- EVAL(exit) = ?

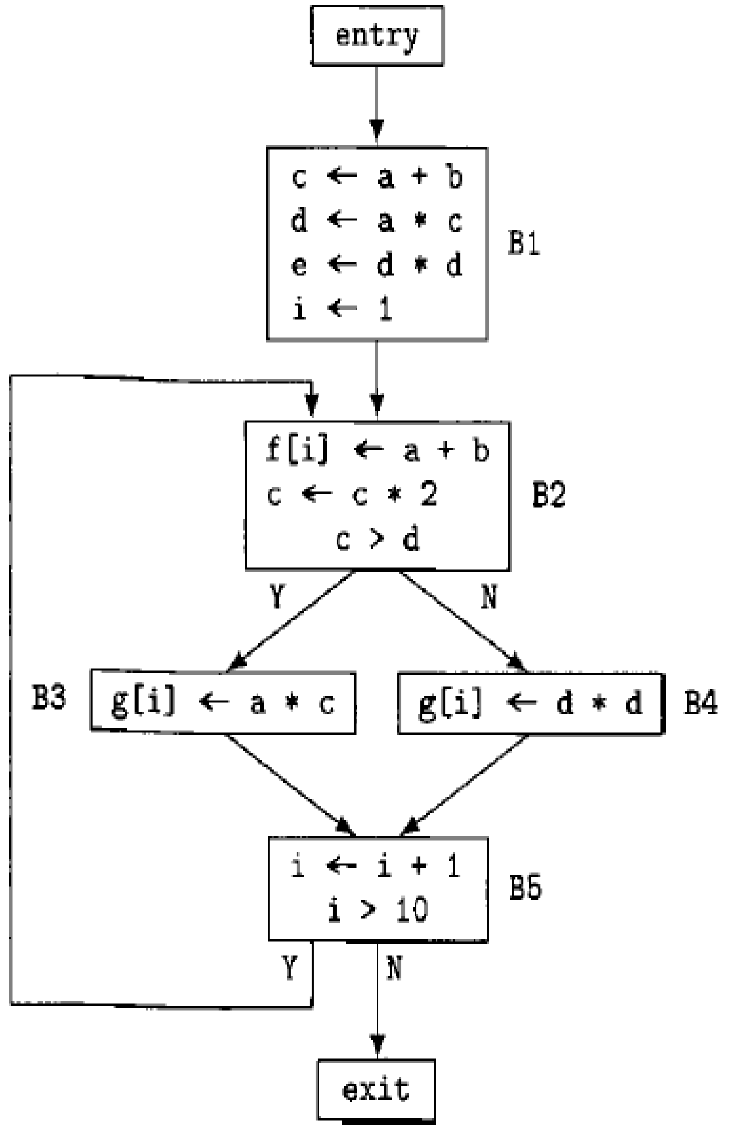
To compute **EVAL(i)**,

— we scan block i from beginning to end,

— accumulating the expressions evaluated in it and

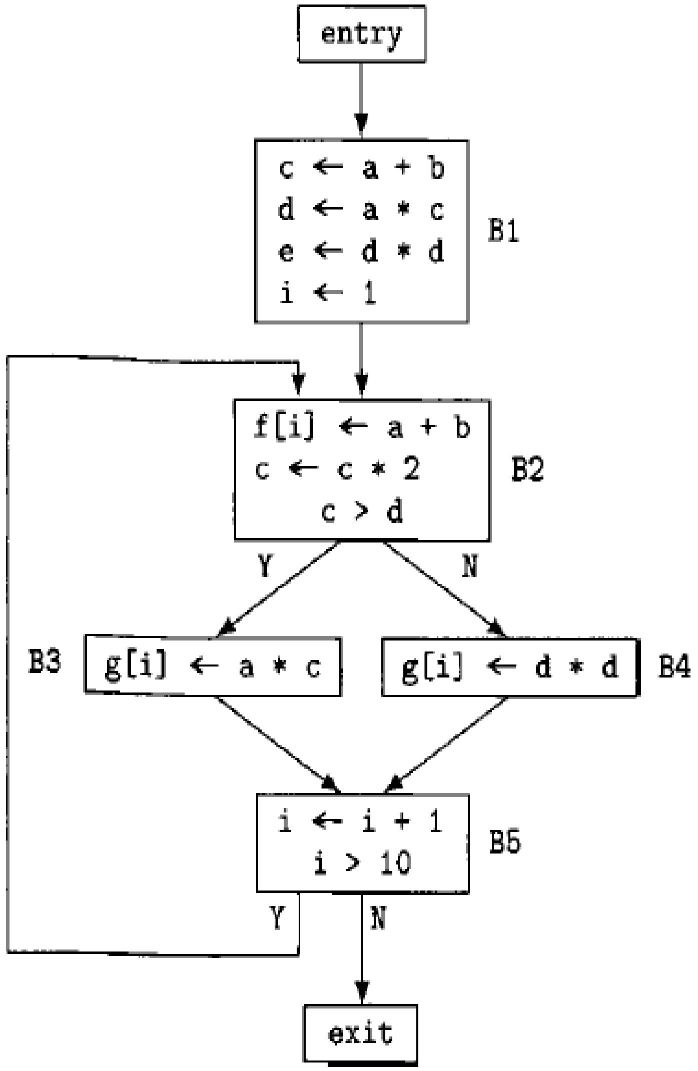
— deleting those whose operands are later assigned new values in the block.

The EVAL(i) sets for the basic blocks



- EVAL(entry) = ∅
- EVAL(B1) = {a+b, a*c, d*d}
- EVAL(B2) = {a+b, c>d}
- EVAL(B3) = {a*c}
- EVAL(B4) = {d*d}
- EVAL(B5) = {i<10}
- EVAL(exit) = ∅

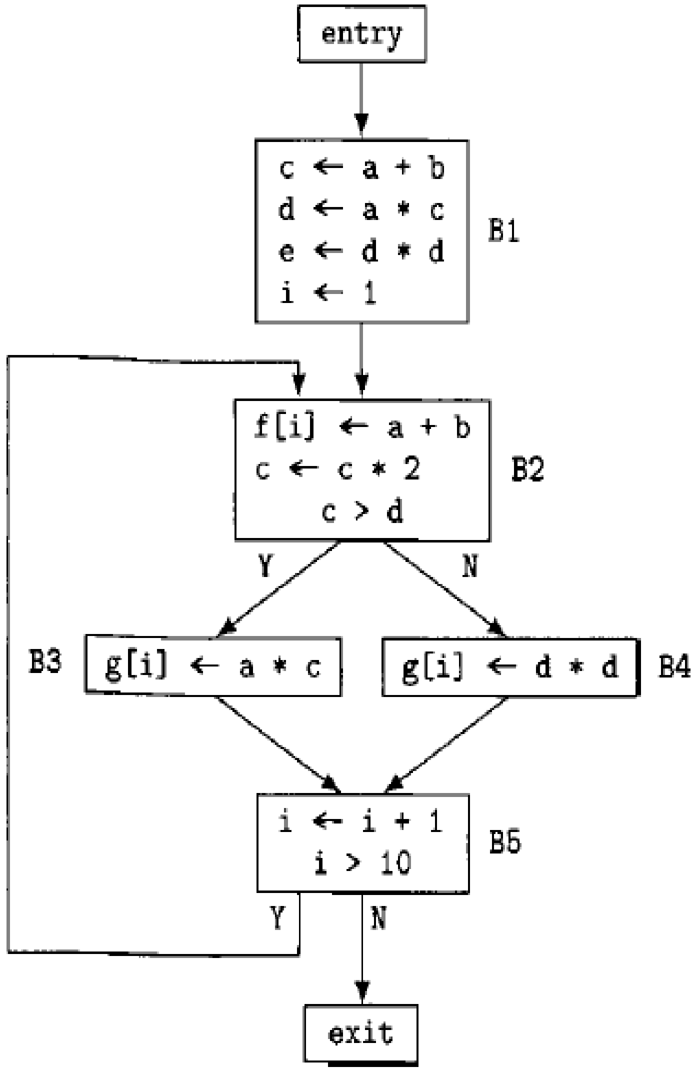
Find the KILL(i) sets for the basic blocks



- KILL(entry) = ?
- KILL(B1) = ?
- KILL(B2) = ?
- KILL(B3) = ?
- KILL(B4) = ?
- KILL(B5) = ?
- KILL(exit) = ?

- **KILL(i)** is the set of all expressions
  - evaluated in **other blocks** such that one or more of their operands are **assigned to in block i**,
  - or that are evaluated in **block i** and subsequently have an operand **assigned to in block i**.

The KILL(i) sets for the basic blocks



- KILL(entry) = Ø
- KILL(B1) = {c*2, c>d ,a*c, d*d, i+1, i>10}
- KILL(B2) = {a*c, c*2}
- KILL(B3) = Ø
- KILL(B4) = Ø
- KILL(B5) = {i +1}
- KILL(exit) = Ø

The equation system for the data-flow analysis can be constructed as follows:

- This is a forward-flow problem.
- We use **in(i)** and **out(i)** to represent the sets of expressions that are available on entry to and exit from block i, respectively.
- An **expression is available on entry** to block i if it is available at the exits of all predecessor blocks, so the path-combining operator is set intersection.
- An **expression is available at the exit** from a block if it is either evaluated in the block and not subsequently killed in it, or if it is available on entry to the block and not killed in it.

The system of data-flow equations is:

- **$out(i) = U - KILL(i)$ for all $i \neq entry$**
- **$U = \cup\, EVAL(i)$** //union for all i
- **$U = \{a+b, a*c, d*d, c>d, i>10\}$**
- **$in(i) = \cap\, out(j)$ $j \in Pred(i)$**

An Iterative Algorithm for Computing Available Expressions

```
for each block $B \neq B1$ do $\{OUT[B] = U - e\_kill[B];\}$
/* You could also do $IN[B] = U$;*/
/* In such a case, you must also interchange the order of */
/* $IN[B]$ and $OUT[B]$ equations below */
$change = true$;
while $change$ do { $change = false$;
  for each block $B \neq B1$ do {
        $IN[B] = \bigcap_{P \text{ a predecessor of } B} OUT[P]$;
        $oldout = OUT[B]$;
        $OUT[B] = e\_gen[B] \bigcup (IN[B] - e\_kill[B])$;
    if ($OUT[B] \neq oldout$) $change = true$;
  }
}
```

For all blocks, calculate out(i)

- out(i) = U - KILL(i)   for all i ≠ entry
- U = ∪ EVAL(i) //union for all i   U={a+b, a*c, d*d, c>d, i>10}

- KILL(entry) = Ø
- KILL(B1) = {c*2, c>d ,a*c, d*d, i+1, i>10}
- KILL(B2) = {a*c, c*2}
- KILL(B3) = Ø
- KILL(B4) = Ø
- KILL(B5) = {i +1}
- KILL(exit) = Ø