SNS COLLEGE OF TECHNOLOGY

(Autonomous)

COIMBATORE-35

Optimization of Basic blocks &Loops in flow Graphs

Optimization of Basic blocks

‘Optimizations’ of Basic Blocks

Equivalent transformations: Two basic block are equivalent if they compute the same set of expressions.

-**Expressions:** are the values of the live variables at the exit of the block.

Two important classes of local transformations:

-structure preserving transformations:

- common sub expression elimination
- dead code elimination
- renaming of temporary variables
- interchange of two independent adjacent statements.

-algebraic transformations (countlessly many):

- simplify expressions
- replace expensive operations with cheaper ones.

Optimization of Basic blocks

10

Transformations on Basic Blocks

- A *code-improving transformation* is a code optimization to improve speed or reduce code size
- *Global transformations* are performed across basic blocks
- *Local transformations* are only performed on single basic blocks
- Transformations must be safe and preserve the meaning of the code
  - A local transformation is safe if the transformed basic block is guaranteed to be equivalent to its original form

3

11

Common-Subexpression Elimination

- Remove redundant computations

```
a  := b + c
b  := a - d
c  := b + c
d  := a - d
```

⇨

```
a  := b + c
b  := a - d
c  := b + c
d  := b
```

```
t1  := b * c
t2  := a - t1
t3  := b * c
t4  := t2 + t3
```

⇨

```
t1  := b * c
t2  := a - t1
t4  := t2 + t1
```

4

Dead Code Elimination

- Remove unused statements

```
b := a + 1
a := b + c
...
```

```
b := a + 1
...
```

Assuming **a** is *dead* (not used)

```
if true goto L2
```

```
b := x + y
...
```

Remove unreachable code

Renaming Temporary Variables

- Temporary variables that are dead at the end of a block can be safely renamed

```
t1 := b + c
t2 := a - t1
t1 := t1 * d
d := t2 + t1
```

```
t1 := b + c
t2 := a - t1
t3 := t1 * d
d := t2 + t3
```

Normal-form block

Interchange of Statements

- Independent statements can be reordered

t1 := b + c t2 := a - t1 t3 := t1 * d d := t2 + t3	⇒	t1 := b + c t3 := t1 * d t2 := a - t1 d := t2 + t3

Note that normal-form blocks permit all statement interchanges that are possible

Algebraic Transformations

- Change arithmetic operations to transform blocks to algebraic equivalent forms

```
t1 := a - a
t2 := b + t1
t3 := 2 * t2
```

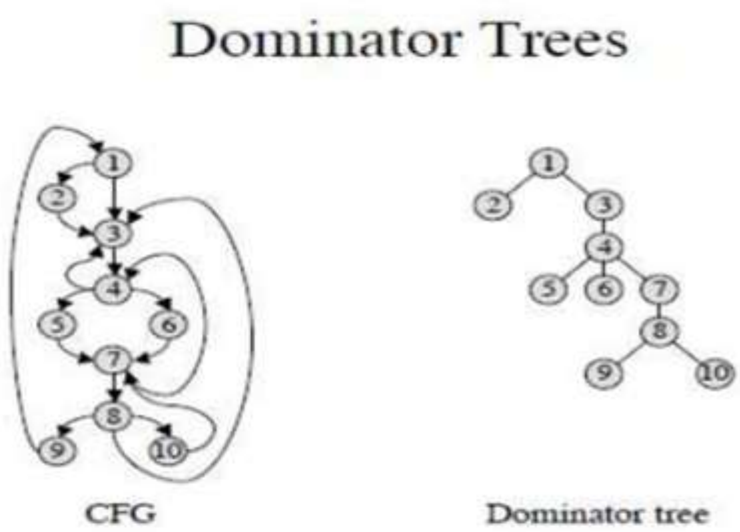
```
t1 := 0
t2 := b
t3 := t2 << 1
```

Loop in flow graphs

Loop in flow graphs

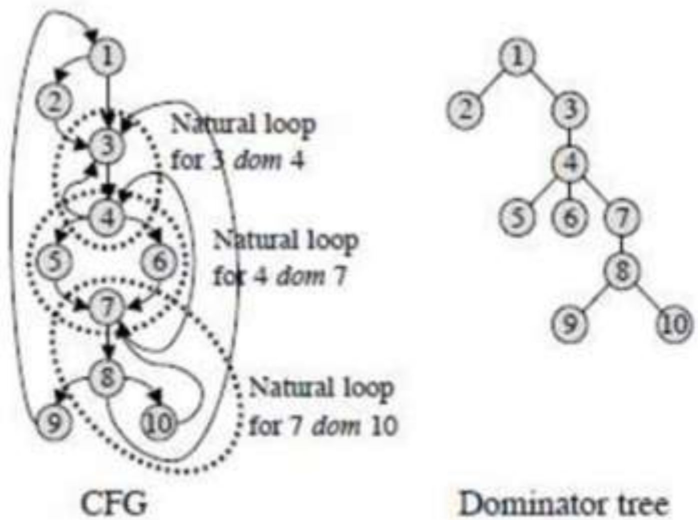
i. Dominators
ii. Natural loops
iii. Inner loops
iv. Pre-Headers
v. Reducible flow graphs

Loop in flow graph



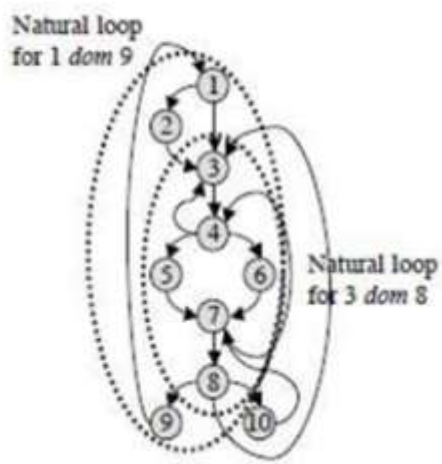
Loop in flow graph

NATURAL INNER LOOPS EXAMPLE

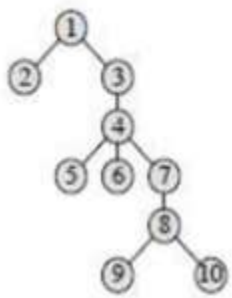


Loop in flow graph

NATURAL OUTER LOOPS EXAMPLE

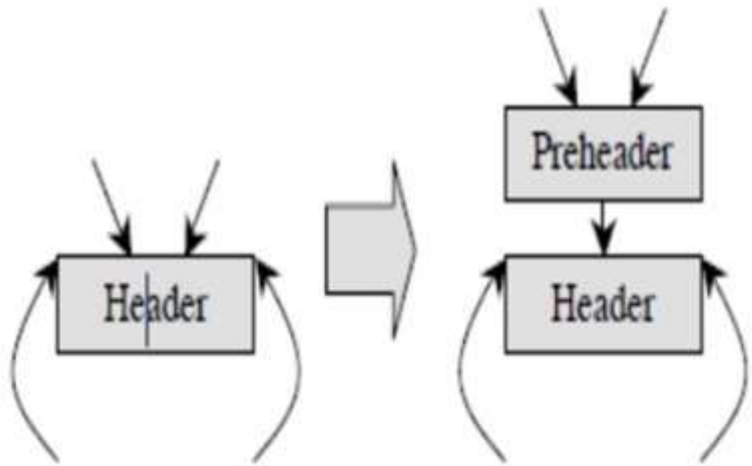


CFG

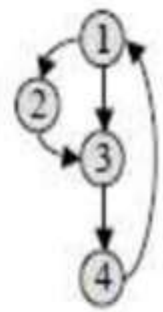


Dominator tree

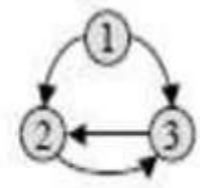
Loop in flow graph-Pre header



Loop in flow graph-Reducible Graph



Example of a reducible CFG



Example of a nonreducible CFG

Summarization