Errata

• Reading questions: please post!

• A Simple, Fast Dominance Algorithm. Cooper, Harvey, and Kennedy

• Finding Dominators in Practice. Georgiadis, Tarjan, and Werneck

• Homework 1: On website.

• Class discussion board: please use!
What does this do?

instr 1: enter 0
instr 2: br [4]
instr 3: enter 0
instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]
instr 6: write m#16
instr 7: ret 16
instr 8: mul n#24 m#16
instr 9: sub n#24 1
instr 10: param (9)
instr 11: param (8)
instr 12: mod (9) 2

instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]
instr 15: call [1]
instr 16: ret 16
instr 17: call [3]
instr 18: ret 16
instr 19: entrypc
instr 20: enter 0
instr 21: param 5
instr 22: param 1
instr 23: call [1]
instr 24: ret 0
Control Flow

• Basic block
  • Straight line sequence of code
  • Single entry point at beginning
  • Single exit at end
• Control flow graph (CFG)
  • Directed graph of basic blocks
  • Edges represent control transfer
Basic Blocks

instr 1: enter 0
instr 2: br [4]
instr 3: enter 0
instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]
instr 6: write m#16
instr 7: ret 16
instr 8: mul n#24 m#16
instr 9: sub n#24 1
instr 10: param (9)
instr 11: param (8)
instr 12: mod (9) 2
instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]
instr 15: call [1]
instr 16: ret 16
instr 17: call [3]
instr 18: ret 16
instr 19: entrypc
instr 20: enter 0
instr 21: param 5
instr 22: param 1
instr 23: call [1]
instr 24: ret 0
Basic Blocks

instr 1: enter 0
instr 2: br [4]
instr 3: enter 0
instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]
instr 6: write m#16
instr 7: ret 16
instr 8: mul n#24 m#16
instr 9: sub n#24 1
instr 10: param (9)
instr 11: param (8)
instr 12: mod (9) 2

instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]
instr 15: call [1]
instr 16: ret 16
instr 17: call [3]
instr 18: ret 16
instr 19: entrypc
instr 20: enter 0
instr 21: param 5
instr 22: param 1
instr 23: call [1]
instr 24: ret 0
Basic Blocks

instr 1: enter 0
instr 2: br [4]
instr 3: enter 0
instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]
instr 6: write m#16
instr 7: ret 16
instr 8: mul n#24 m#16
instr 9: sub n#24 1
instr 10: param (9)
instr 11: param (8)
instr 12: mod (9) 2

instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]
instr 15: call [1]
instr 16: ret 16
instr 17: call [3]
instr 18: ret 16
instr 19: entrypc
instr 20: enter 0
instr 21: param 5
instr 22: param 1
instr 23: call [1]
instr 24: ret 0
Basic Blocks

instr 1: enter 0
instr 2: br [4]
instr 3: enter 0
instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]
instr 6: write m#16
instr 7: ret 16
instr 8: mul n#24 m#16
instr 9: sub n#24 1
instr 10: param (9)
instr 11: param (8)
instr 12: mod (9) 2
instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]
instr 15: call [1]
instr 16: ret 16
instr 17: call [3]
instr 18: ret 16
instr 19: entrypc
instr 20: enter 0
instr 21: param 5
instr 22: param 1
instr 23: call [1]
instr 24: ret 0
Basic Blocks

instr 1: enter 0
instr 2: br [4]
instr 3: enter 0
instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]
instr 6: write m#16
instr 7: ret 16
instr 8: mul n#24 m#16
instr 9: sub n#24 1
instr 10: param (9)
instr 11: param (8)
instr 12: mod (9) 2
instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]
instr 15: call [1]
instr 16: ret 16
instr 17: call [3]
instr 18: ret 16
instr 19: entrypc
instr 20: enter 0
instr 21: param 5
instr 22: param 1
instr 23: call [1]
instr 24: ret 0
Basic Blocks

instr 1: enter 0
instr 2: br [4]

instr 3: enter 0

instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]

instr 6: write m#16
instr 7: ret 16

instr 8: mul n#24 m#16
instr 9: sub n#24 1

instr 10: param (9)
instr 11: param (8)

instr 12: mod (9) 2

instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]

instr 15: call [1]
instr 16: ret 16

instr 17: call [3]

instr 18: ret 16

instr 19: entrypc

instr 20: enter 0

instr 21: param 5

instr 22: param 1

instr 23: call [1]

instr 24: ret 0
Basic Blocks

instr 1: enter 0
instr 2: br [4]
instr 3: enter 0
instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]
instr 6: write m#16
instr 7: ret 16
instr 8: mul n#24 m#16
instr 9: sub n#24 1
instr 10: param (9)
instr 11: param (8)
instr 12: mod (9) 2

instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]
instr 15: call [1]
instr 16: ret 16
instr 17: call [3]
instr 18: ret 16
instr 19: entrypc
instr 20: enter 0
instr 21: param 5
instr 22: param 1
instr 23: call [1]
instr 24: ret 0
Basic Blocks

instr 1: enter 0
instr 2: br [4]
instr 3: enter 0
instr 4: cmpeq n#24 0
instr 5: blbc (4) [8]
instr 6: write m#16
instr 7: ret 16
instr 8: mul n#24 m#16
instr 9: sub n#24 1
instr 10: param (9)
instr 11: param (8)
instr 12: mod (9) 2

instr 13: cmpeq (12) 0
instr 14: blbc (13) [17]
instr 15: call [1]
instr 16: ret 16
instr 17: call [3]
instr 18: ret 16
instr 19: entrypc
instr 20: enter 0
instr 21: param 5
instr 22: param 1
instr 23: call [1]
instr 24: ret 0
Control Flow Graph

1: enter 0

3: enter 0

4: cmpeq n#24, 0
5: blbc (4) [8]

8: mul n#24 m#16
9: sub n#24 1
10: param (9)
11: param (8)
12: mod (9) 2
13: cmpeq (12) 0
14: blbc (13) [17]

15: call [1]
16: ret 16

6: write m#16
7: ret

17: call [3]
18: ret 16

T
F

Wednesday, April 3, 13
Tail Merging

1: enter 0

4: cmpeq n#24, 0
5: blbc (4) [8]

6: write m#16
7: ret

8: mul n#24 m#16
9: sub n#24 1
10: param (9)
11: param (8)
12: mod (9) 2
13: cmpeq (12) 0
14: blbc (13) [17]

15: call [1]
16: ret 16

T
F

17: call [1]
18: ret 16
Tail-Call Optimization

1: enter 0
4: cmpeq n#24, 0
5: blbc (4) [8]

T
8: mul n#24 m#16
9: sub n#24 1
10: param (9)
11: param (8)
12: mod (9) 2
13: cmpeq (12) 0
14: blbc (13) [17]
F
6: write m#16
7: ret

15: move m#16 (8)
16: move n#24 (9)
17: move m#16 (8)
18: move n#24 (9)
Tail Merging Again

1: enter 0

4: cmpeq n#24, 0
5: blbc (4) [8]

6: write m#16
7: ret

8: mul n#24 m#16
9: sub n#24 1
10: param (9)
11: param (8)
12: mod (9) 2
13: cmpeq (12) 0
14: blbc (13) [15]
15: move m#16 (8)
16: move n#24 (9)
Dead Code Elimination

1: enter 0

4: cmpeq n#24, 0
5: blbc (4) [8]

T
F

8: mul n#24 m#16
9: sub n#24 1
15: move m#16 (8)
16: move n#24 (9)

6: write m#16
7: ret

Wednesday, April 3, 13
CFGs

- Enable structural transformations of program
  - Simplify / optimize code
  - Enable further optimization (e.g., dead code)
- Also drives program analysis
  - Data flow analysis
  - Loop identification
Where is the loop?

1: enter 0

4: cmpeq n\#24, 0
5: blbc (4) [8]

8: mul n\#24 m\#16
9: sub n\#24 1
15: move m\#16 (8)
16: move n\#24 (9)

6: write m\#16
7: ret
Loop-driven optimization

1: enter 0

4: cmpeq n#24, 0
5: blbc (4) [8]

T

8: mul n#24 m#16
9: sub n#24 1
10: mul k#40 l#32
11: add (8) (10)
15: move m#16 (11)
16: move n#24 (9)

F

6: write m#32
7: ret
Code motion

1: enter 0
10: mul k#40 l#32

4: cmpeq n#24, 0
5: blbc (4) [8]

8: mul n#24 m#16
9: sub n#24 1
11: add (8) (10)
15: move m#16 (11)
16: move n#24 (9)

6: write m#32
7: ret
How do we find loops?

- We use an idea called *dominance*.
- Node M dominates node N if every path from the entry to N must contain M.
- The entry dominates all nodes.
Dominators

- A: \{A\}
Dominators

- A: \{A\}
- B: \{A, B\}

Wednesday, April 3, 13
Dominators

- A: \{A\}
- B: \{A, B\}
- C: \{A, B, C\}
Dominators

- A: \{ A \}
- B: \{A, B\}
- C: \{A, B, C\}
- D: \{A, B, D\}
Dominators

- A: \{ A \}
- B: \{ A, B \}
- C: \{ A, B, C \}
- D: \{ A, B, D \}
- E: \{ A, B, E \}
Dominators

- A: \{ A \}
- B: \{ A, B \}
- C: \{ A, B, C \}
- D: \{ A, B, D \}
- E: \{ A, B, E \}
- F: \{ A, B, E, F \}
Dominators

- A: \{ A \}
- B: \{ A, B \}
- C: \{ A, B, C \}
- D: \{ A, B, D \}
- E: \{ A, B, E \}
- F: \{ A, B, E, F \}
- G: \{ A, B, E, G \}
Dominators

- A: \{A\}
- B: \{A, B\}
- C: \{A, B, C\}
- D: \{A, B, D\}
- E: \{A, B, E\}
- F: \{A, B, E, F\}
- G: \{A, B, E, G\}
- H: \{A, B, E, F, H\}
Computing Dominators

- Maximal solution to:

\[
\text{DOM}(n_0) = \{n_0\}
\]

\[
\text{DOM}(n) = \left( \bigcap_{p \in \text{preds}(n)} \text{DOM}(p) \right) \cup \{n\}
\]

- From Cooper, et al.
Iterative Solution

for all nodes, $n$

$$\text{DOM}[n] \leftarrow \{1 \ldots N\}$$

Changed $\leftarrow$ true

while (Changed)

Changed $\leftarrow$ false

for all nodes, $n$, in reverse postorder

new_set $\leftarrow \left( \bigcap_{p \in \text{preds}(n)} \text{DOM}[p] \right) \cup \{n\}$

if (new_set $\neq \text{DOM}[n]$)

$$\text{DOM}[n] \leftarrow \text{new_set}$$

Changed $\leftarrow$ true
Step 1
Step 1

- A: \{ A \}
- B: \{A, B, C, D, E, F, G, H\}
- C: \{A, B, C, D, E, F, G, H\}
- D: \{A, B, C, D, E, F, G, H\}
- E: \{A, B, C, D, E, F, G, H\}
- F: \{A, B, C, D, E, F, G, H\}
- G: \{A, B, C, D, E, F, G, H\}
- H: \{A, B, C, D, E, F, G, H\}
Step 2

A -> B -> C -> D -> E -> F -> G -> H
Step 2

- A: \{ A \}
- B: \{A, B\}
- C: \{A, B, C\}
- D: \{A, B, D\}
- E: \{A, B, E\}
- F: \{A, B, E, F\}
- G: \{A, B, E, G\}
- H: \{A, B, E, F, H\}
Dominators

Tree representation
Immediate Dominator

Tree representation

A
   /\  
B   C  E  D
   /\  /\  /\  
F  G H  I  J

Wednesday, April 3, 13
Immediate Dominator

Tree representation

- \text{idom}(B) = A
- \text{idom}(C) = B
- \text{idom}(D) = B
- \text{idom}(E) = B
- \text{idom}(F) = E
- \text{idom}(G) = E
- \text{idom}(H) = F
Efficient Algorithms

- For m edges, n nodes
  - Simple: $O(mn^2)$
  - Cooper, et al.: $O(mn^2)$ with better const & space
  - Lengauer-Tarjan:
    - Standard: $O(m \cdot \alpha(m,n))$
    - Simple: $O(m \log(2+m/n) \cdot n)$
  - Georgiadis: $O(n^2)$
for all nodes, b /* initialize the dominators array */
    doms[b] = Undefined
doms[start node] = start node
Changed = true
while (Changed)
    Changed = false
    for all nodes, b, in reverse postorder { // (except start node)
        new_idom = first (processed) predecessor of b /* (pick one) */
        for all other predecessors, p, of b
            if doms[p] != Undefined /* i.e., if doms[p] already calculated */
                new idom intersect(p, new idom)
        if doms[b] != new_idom
            doms[b] = new_idom
            Changed = true
function intersect(b1, b2) returns node
    finger1 = b1
    finger2 = b2
    while (finger1 != finger2)
        while (finger1 < finger2)
            finger1 = doms[finger1]
        while (finger2 < finger1)
            finger2 = doms[finger2]
    return finger1
Shootout (Georgiadis)
Back to loops

- **Back-edge**: An edge where the target dominates the source.

- **Loop header**: The target (y) of one or more back edges of the form \((x \rightarrow y)\).

- **Loop node**: Predecessor z of x such that there exists a path from z to x that does not include y.

- **Loop consists of header and set of loop nodes.**
Example

A → B → C → D → E

H → G → E → D → B
Example

• Back edges

A

B

C

D

E

F

G

H
Example

- Back edges
  - F → B
Example

- Back edges
  - $F \rightarrow B$
  - $G \rightarrow B$
Example

- Back edges
  - F → B
  - G → B
- Loop header: B
Example

- Back edges
  - F → B
  - G → B
- Loop header: B
- Loop nodes:
Example

- Back edges
  - F → B
  - G → B
- Loop header: B
- Loop nodes:
  - \{ B, C, D, E, F, G \}