Control flow graphs
Moving beyond basic blocks

• Up until now, we have focused on single basic blocks
• What do we do if we want to consider larger units of computation
  • Whole procedures?
  • Whole program?
• Idea: capture control flow of a program
  • How control transfers between basic blocks due to:
    • Conditionals
    • Loops
Representation

- Use standard three-address code
- Jump targets are labeled
- Also label beginning/end of functions
- Want to keep track of *targets of jump statements*
  - Any statement whose execution may immediately follow execution of jump statement
  - *Explicit* targets: targets mentioned in jump statement
  - *Implicit* targets: statements that follow conditional jump statements
    - The statement that gets executed if the branch is not taken
Running example

A = 4

\[ t_1 = A \times B \]

repeat { 
  \[ t_2 = \frac{t_1}{C} \]
  if \( (t_2 \geq W) \) { 
    \[ M = t_1 \times k \]
    \[ t_3 = M + I \]
  }
  \[ H = I \]
  \[ M = t_3 - H \]
} until \( (T3 \geq 0) \)
Running example

1    A = 4
2    t1 = A * B
3    L1: t2 = t1 / C
4    if t2 < W goto L2
5    M = t1 * k
6    t3 = M + I
7    L2:  H = I
8    M = t3 - H
9    if t3 ≥ 0 goto L3
10   goto L1
11   L3:  halt
Control flow graphs

• Divides statements into *basic blocks*

• Basic block: a maximal sequence of statements $l_0, l_1, l_2, \ldots, l_n$ such that if $l_j$ and $l_{j+1}$ are two adjacent statements in this sequence, then

  • The execution of $l_j$ is always immediately followed by the execution of $l_{j+1}$
  
  • The execution of $l_{j+1}$ is always immediately preceded by the execution of $l_j$

• Edges between basic blocks represent potential flow of control
How do we build this automatically?
Constructing a CFG

- To construct a CFG where each node is a basic block
  - Identify *leaders*: first statement of a basic block
  - In program order, construct a block by appending subsequent statements up to, but not including, the next leader
- Identifying leaders
  - First statement in the program
  - Explicit target of any conditional or unconditional branch
  - Implicit target of any branch
Partitioning algorithm

- **Input:** set of statements, \( \text{stat}(i) = \text{ith statement in input} \)
- **Output:** set of **leaders**, set of basic blocks where \( \text{block}(x) \) is the set of statements in the block with leader \( x \)
- **Algorithm**
  
  \[
  \text{leaders} = \{1\} \quad // \text{Leaders always includes first statement}
  \]
  
  \[
  \text{for } i = 1 \text{ to } |n| \quad // |n| = \text{number of statements}
  \]
  
  \[
  \quad \text{if } \text{stat}(i) \text{ is a branch, then}
  \]
  
  \[
  \quad \text{leaders} = \text{leaders} \cup \text{all potential targets}
  \]
  
  \[
  \text{end for}
  \]

  \[
  \text{worklist} = \text{leaders}
  \]

  \[
  \text{while worklist not empty do}
  \]
  
  \[
  \quad x = \text{remove earliest statement in worklist}
  \]

  \[
  \quad \text{block}(x) = \{x\}
  \]

  \[
  \quad \text{for } (i = x + 1; i \leq |n| \text{ and } i \notin \text{leaders}; i++)
  \]

  \[
  \quad \text{block}(x) = \text{block}(x) \cup \{i\}
  \]

  \[
  \quad \text{end for}
  \]

  \[
  \text{end while}
  \]
Running example

1. \( A = 4 \)
2. \( t_1 = A \times B \)
3. \( L1: \ t_2 = t_1 / C \)
4. \( \text{if } t_2 < W \text{ goto } L2 \)
5. \( M = t_1 \times k \)
6. \( t_3 = M + I \)
7. \( L2: \ H = I \)
8. \( M = t_3 - H \)
9. \( \text{if } t_3 \geq 0 \text{ goto } L3 \)
10. \( \text{goto } L1 \)
11. \( L3: \text{halt} \)

Leaders =
Basic blocks =
Running example

1 A = 4
2 t1 = A * B
3 \textbf{L1: } t2 = t1 / C
4 \textbf{if } t2 < W \textbf{ goto L2}
5 M = t1 * k
6 t3 = M + I
7 \textbf{L2: } H = I
8 M = t3 - H
9 \textbf{if } t3 \geq 0 \textbf{ goto L3}
10 \textbf{ goto L1}
11 \textbf{L3: } \text{halt}

Leaders = \{1, 3, 5, 7, 10, 11\}
Basic blocks = \{ \{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8, 9\}, \{10\}, \{11\} \}
Putting edges in CFG

• There is a directed edge from $B_1$ to $B_2$ if
  • There is a branch from the last statement of $B_1$ to the first statement (leader) of $B_2$
  • $B_2$ immediately follows $B_1$ in program order and $B_1$ does not end with an unconditional branch

• Input: $block$, a sequence of basic blocks

• Output: The CFG

  $$\text{for } i = 1 \text{ to } |block|$$
  $$\text{\hspace{1cm}} x = \text{last statement of } block(i)$$
  $$\text{\hspace{1cm}} \text{if } \text{stat}(x) \text{ is a branch, then}$$
  $$\text{\hspace{2cm}} \text{for each explicit target } y \text{ of } \text{stat}(x)$$
  $$\text{\hspace{3cm}} \text{create edge from block } i \text{ to block } y$$
  $$\text{\hspace{2cm}} \text{end for}$$
  $$\text{\hspace{1cm}} \text{if } \text{stat}(x) \text{ is not unconditional then}$$
  $$\text{\hspace{2cm}} \text{create edge from block } i \text{ to block } i+1$$
  $$\text{\hspace{1cm}} \text{end for}$$
A = 4
\[ t1 = A \times B \]

L1: \[ t2 = \frac{t1}{c} \]
if \( t2 < W \) goto L2

M = t1 \times k
\[ t3 = M + I \]

L2: \[ H = I \]
M = t3 - H
if \( t3 \geq 0 \) goto L3

goto L1

L3: halt
Discussion

- Some times we will also consider the *statement-level* CFG, where each node is a statement rather than a basic block.
- Either kind of graph is referred to as a CFG.
- In statement-level CFG, we often use a node to explicitly represent *merging* of control.
- Control merges when two different CFG nodes point to the same node.
- Note: if input language is *structured*, front-end can generate basic block directly.
- “GOTO considered harmful”
A = 4

t1 = A * B

L1: t2 = t1/c

if t2 < W goto L2

M = t1 * k

t3 = M + I

L2: H = I

M = t3 - H

if t3 ≥ 0 goto L3

L3: halt

goto L1