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
t1 = A * B
repeat {
t2 = t1 / C
  if (t2 ≥ W) {
    M = t1 * k
t3 = M + I
  }
  H = I
  M = t3 - H
} until (T3 ≥ 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 \( I_0, I_1, I_2, ..., I_n \) such that if \( I_i \) and \( I_{i+1} \) are two adjacent statements in this sequence, then
  - The execution of \( I_i \) is always immediately followed by the execution of \( I_{i+1} \)
  - The execution of \( I_{i+1} \) is always immediately preceded by the execution of \( I_i \)
- Edges between basic blocks represent potential flow of control
### CFG for running example

```
A = 4
υ1 = A * B
υ2 = υ1 / C
if ν2 < W goto L2
ν3 = ν1 * k
ν4 = ν3 + I
L2:
H = I
ν5 = ν3 - H
if ν3 ≥ 0 goto L3
L3:
halt
```

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

### Running example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A = 4</td>
</tr>
<tr>
<td>2</td>
<td>t1 = A * B</td>
</tr>
<tr>
<td>3</td>
<td>L1: ( t2 = t1 / C )</td>
</tr>
<tr>
<td>4</td>
<td>if ( t2 &lt; W ) goto L2</td>
</tr>
<tr>
<td>5</td>
<td>M = t1 * k</td>
</tr>
<tr>
<td>6</td>
<td>t3 = M + I</td>
</tr>
<tr>
<td>7</td>
<td>L2: H = I</td>
</tr>
<tr>
<td>8</td>
<td>M = t3 - H</td>
</tr>
<tr>
<td>9</td>
<td>if ( t3 \geq 0 ) goto L3</td>
</tr>
<tr>
<td>10</td>
<td>goto L1</td>
</tr>
<tr>
<td>11</td>
<td>L3: halt</td>
</tr>
</tbody>
</table>

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 \( B_i \) to \( B_j \) if
- There is a branch from the last statement of \( B_i \) to the first statement (leader) of \( B_j \)
- \( B_j \) immediately follows \( B_i \) in program order and \( B_i \) does not end with an unconditional branch
- Input: block, a sequence of basic blocks
- Output: The CFG
  ```
  for \( i = 1 \) to |block|
    \( x = \text{last statement of block}(i) \)
    if stat(\( x \)) is a branch, then
      for each explicit target \( y \) of stat(\( x \))
        create edge from block \( i \) to block \( y \)
      end for
    end if
    if stat(\( x \)) is not unconditional then
      create edge from block \( i \) to block \( i+1 \)
    end for
  end for
  ```
**Result**

\[
A = 4 \\
t_1 = A \times B
\]

L1:
\[
t_2 = \frac{t_1}{c} \\
\text{if } t_2 < W \text{ goto L2}
\]

\[
M = t_1 \times k \\
t_3 = M + I
\]

L2:
\[
H = I \\
M = t_3 - H \\
\text{if } t_3 \geq 0 \text{ goto L3}
\]

\[
\text{halt}
\]

**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”

**Statement level CFG**