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
1  t1 = A * B
2  repeat { t2 = t1 / C
3    if t2 < W goto L2
4    M = t1 * k
5    t3 = M + I
6   } L2:
7    H = I
8    M = t3 - H
9  } until (T3 ≥ 0)
```

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
1  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
     goto L1
     L3:  halt
```

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, \( \text{stat}(i) = i^{th} \) 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
  1. \( \text{leaders} = \{1\} \)
  2. for \( i = 1 \) to |\( n \)|
  3.   if \( \text{stat}(i) \) is a branch, then
  4.     \( \text{leaders} = \text{leaders} \cup \) all potential targets
  5.   end if
  6. worklist = leaders
  7. while worklist not empty
  8.   x = remove earliest statement in worklist
  9.   block(x) = \{x\}
  10.   for \( i = x + 1 \) to |\( n \)| and \( i \notin \text{leaders} \)
  11.     block(x) = block(x) \cup \{i\}
  12. end for
  13. end while

**Running example**

```
1  A = 4
2  t1 = A * B
3  L1:  t2 = t1 / C
      if t2 < W goto L2
      M = t1 * k
      t3 = M + I
6  L2:  H = I
      M = t3 - H
      if t3 ≥ 0 goto L3
5  goto L1
8  L3:  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 \( B_i \) 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
  1. for \( i = 1 \) to |\( \text{block} \)|
  2.   \( x = \text{last statement of block}(i) \)
  3.   if \( \text{stat}(x) \) is a branch, then
  4.     for each explicit target \( y \) of \( \text{stat}(x) \)
  5.       create edge from block \( i \) to block \( y \)
  6.   end if
  7.   if \( \text{stat}(x) \) is not unconditional then
  8.     create edge from block \( i \) to block \( i+1 \)
  9. end for
```
Result

\[ A = 4 \]
\[ t1 = A \times B \]

\[ \ldots \]
\[ t2 = t1/c \]
\[ \text{if } t2 < W \text{ goto } L2 \]
\[ M = t1 \times k \]
\[ t3 = M + I \]

\[ \ldots \]
\[ M = t3 - H \]
\[ \text{if } t3 \geq 0 \text{ goto } L3 \]
\[ \text{halt} \]

Discussion

- Sometimes 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"