Control-Flow Analysis

Motivating example: identifying loops
- majority of runtime
  - focus optimization on loop bodies!
  - remove redundant code, replace expensive operations)
    speed up program

Finding loops:
- easy...
- or harder
  (GOTOs)

Steps to Finding Loops
1. Identify basic blocks
2. Build control-flow graph
3. Analyze CFG to find loops

Identifying Basic Blocks
- Input: sequence of instructions \( \text{instr}(i) \)
- Identify leaders:
  - first instruction of basic block
- Iterate: add subsequent instructions to basic block until we reach another leader

Control-Flow Graphs
- Control-flow graph:
  - Node: an instruction or sequence of instructions (a basic block)
    - Two instructions \( i, j \) in same basic block
    - if execution of \( i \) guarantees execution of \( j \)
  - Directed edge: potential flow of control
  - Distinguished start node Entry
    - First instruction in program

Basic Block Partition Algorithm

```plaintext
leaders = 1 // start of program
for i = 1 to |n| // all instructions
    if instr(i) is a branch
        leaders = leaders \ targets of instr(i)
    worklist = leaders
while worklist not empty
    x = first instruction in worklist
    worklist = worklist - \{x\}
    block(x) = \{x\}
    for i = x + 1; i <= |n| && i not in leaders; i++
        block(x) = block(x) \{i\}
```
Basic Block Example

1. \( A = 4 \)
2. \( t_1 = A \times B \)
3. \( L_1: t_2 = \frac{t_1}{C} \)
4. If \( t_2 < W \) goto \( L_2 \)
5. \( M = t_1 \times k \)
6. \( t_3 = M + I \)
7. \( L_2: H = I \)
8. \( M = t_3 - H \)
9. If \( t_3 \geq 0 \) goto \( L_3 \)
10. Goto \( L_1 \)
11. \( L_3: \text{halt} \)

Leaders

Basic blocks

Control-Flow Edges

- Basic blocks = nodes
- Edges:
  - Add directed edge between \( B_1 \) and \( B_2 \) if:
    - Branch from last statement of \( B_1 \) to first statement of \( B_2 \) (\( B_2 \) is a leader), or
    - \( B_2 \) immediately follows \( B_1 \) in program order and \( B_1 \) does not end with unconditional branch (goto)

Steps to Finding Loops

1. Identify basic blocks
2. Build control-flow graph
3. Analyze CFG to find loops
   - Spanning trees, depth-first spanning trees
   - Reducibility
   - Dominators
   - Dominator tree
   - Strongly-connected components

Control-Flow Edge Algorithm

Input: \( \text{block}(i) \), sequence of basic blocks
Output: CFG where nodes are basic blocks

for \( i = 1 \) to the number of blocks
    \( x = \text{last instruction of block}(i) \)
    if \( \text{instr}(x) \) is a branch
        for each target \( y \) of \( \text{instr}(x) \),
            create edge \( \text{block}(i) \rightarrow \text{block}(y) \)
    if \( \text{instr}(x) \) is not unconditional branch,
        create edge \( \text{block}(i) \rightarrow \text{block}(i+1) \)
for \( v \) in \( V \)
    \( \text{InTree}(v) = \text{false} \)
    \( \text{InTree}(\text{root}) = \text{true} \)
    \( \text{Span}(\text{root}) \)

CFG Edge Example

Spanning Tree

procedure \( \text{Span}(v) \)
    for \( w \in \text{Sup}(v) \)
        if not \( \text{InTree}(w) \)
            add \( v \) to ST
            \( \text{InTree}(w) = \text{true} \)
            \( \text{Span}(w) \)
    for \( v \in \text{Do} \)
        \( \text{InTree}(v) = \text{false} \)
        \( \text{InTree}(\text{root}) = \text{true} \)
        \( \text{Span}(\text{root}) \)
CFG Edge Classification

Tree edge:
in CFG & ST

Advancing edge:
(v,w) not tree edge but w is descendant of v in ST

Back edge:
(v,w): v=w or w is proper ancestor of v in ST

Cross edge:
(v,w): w neither ancestor nor descendant of v in ST

Depth-first spanning tree

procedure DFST(v)
pren(v) = vnum++
InStack(v) = true
for w in Succ(v)
if not InTree(w)
add v \rightarrow w to TreeEdges
InTree(w) = true
DFST(w)
else if pre(v) < pre(w)
add v \rightarrow w to AdvancingEdges
else if InStack(w)
add v \rightarrow w to BackEdges
else
add v \rightarrow w to CrossEdges
InStack(v) = false

for v in V
if inTree = false
pre(v) = vnum
DFST(root)

Reducibility

Natural loops:
- no jumps into middle of loop
- entirely disjoint or nested

Reducible: hierarchical, “well-structured”
- flowgraph reducible if all loops in it natural

Dominance

Node d dominates node i ("d dom i")
if every path from Entry to i includes d

- Reflexive: a dom a
- Transitive: a dom b \land b dom c \Rightarrow a dom c
- Antisymmetric: a dom b, b dom a \Rightarrow b=a

Immediate dominance:
- a dom b \land \neg a \dom c \land \neg c \dom b \Rightarrow a dom c
- idom x:
  each node has unique idom
  relation forms tree

Reducibility Example

Some languages only permit procedures with reducible flowgraphs (e.g., Java)
- "GOTO Considered Harmful": introduces irreducibility
  - FORTRAN
  - C
  - C++

DFST does not find unique header in irreducible graphs

Dominance Example

Immediate and other dominators:
(excluding E nty)
- a idom b; a dom a, c, d, e, f, g
- b idom c; b dom b, d, e, f, g
- c idom d; c dom c, e, f, g
- d idom e, d idom d, f, g
- e idom f, e idom g, edom e
Dominance and Loops
- Redefine back edge as one whose head dominates its tail
- Slightly more restrictive definition
- Now we can (finally) find natural loops!
  - for back edge m → n, natural loop is subgraph of nodes containing m (loop header) and nodes from which m can be reached without passing through n + connecting edges

Strongly-Connected Components
- What about irreducible flowgraphs?
- Most general loop form = strongly-connected component (SCC):
  - subgraph S such that every node in S reachable from every other node by path including only edges in S
- Maximal SCC:
  - S is maximal SCC if it is the largest SCC that contains S.
  - Now: Loops = all maximal SCCs

SCC Example
- Entry
- B1
- B2
- B3
- Maximal strongly-connected component
- Strongly-connected component

Computing Maximal SCCs
- Tarjan’s algorithm:
  - Computes all maximal SCCs
  - Linear-time (in number of nodes and edges)
- CLR algorithm:
  - Also linear-time
  - Simpler:
    - Two depth-first searches and one "transpose": reverse all graph edges
  - Unlike DFST, neither distinguishes inner loops

Conclusion
- Introduced control-flow analysis
- Basic blocks
- Control-flow graphs
- Discussed application of graph algorithms: loops
  - Spanning trees, depth-first spanning trees
  - Reducibility
  - Dominators
  - Dominator tree
  - Strongly connected components

Next Time
- Dataflow analysis
  - Read ACD I Chapter 8, pp. 217-251
  - Photocopies should be available soon