Graph Traversal for Flow Analysis

CS502

Compiler

What graphs?

- During intra-procedural flow analysis, we propagate information through the flow graph
- During inter-procedural flow analysis, we need to also propagate information through the call graph
- The order in which we traverse the graph can have an impact on how fast the analysis converges, i.e. reaches a fixed point that gives us the final result
- To discuss graph traversal in flow analysis, we begin by reviewing several basic graph algorithms that are useful in compilers

Depth-first search

- Depth-first search (DFS) visits nodes in the following manner:

```
main {
  counter = 1;
  dfsearch(root);
}

function dfsearch(n) {
  n->visited = true;
  n->dfn = counter;
  for each successor, s, of n,
    if not s->visited {
      counter = counter+1;
      dfsearch(s);
    }
}
```

NOTE: if an edge exists from node a to node b, b is called a *successor* of a.

- The number n->dfn is called the depth-first number for node n.
- Notice that in routine dfsearch, the order in which we choose a successor in the statement “for each successor, s, …” is left unspecified. A different choice may lead to a different depth-first numbering

DFST

- Under a specific DF numbering, we can define a spanning tree for the given graph, called depth-first spanning tree, DFST.
- To draw the DFST under a specific DF numbering, we insert a pseudo statement in routine dfsearch(n), i.e.
    - if not s->visited {
    -   **draw edge (n, s)**
    -   counter = counter+1;
    -   dfsearch(s); }

Edge classification under a DFST

- Given a specific DFST for rooted graph G, we classify edges in G as follows:
  - tree edges: edges in the DFST
  - forward edges: all edges (a, b) such that b is a *descendant* of a in DFST and (a, b) is not a tree edge
  - retreating edges: all edges (a, b) such that a is a descendant of b in the DFST; and
  - cross edges: all edges not belonging to the above
- (In general, since dfn is non-unique, DFST is also non-unique. Tree classification is valid only for the specific DFST, in general.)

Traversal order

- When we apply a procedure to each node in a graph G, there are two issues:
  - The search order (the order in which we visit the nodes)
  - The traversal order (when do we apply the procedure).
- The following traversal schemes are particularly common:
  - Preorder traversal: each node is processed **before** its descendants, as defined by a specific DFST.
  - Postorder traversal: each node is processed **after** its descendants, as defined by a specific DFST.

Breadth-first search

- Sometimes breadth-first search is useful

```
bfsearch{
  counter = 1;
  root->visited = true;
  worklist = {root};
  while worklist is
nonempty do {
    remove first node, n,
from worklist;
    n->bfn = counter;
    counter = counter + 1;
 for each successor, s, of n,
     if not s->visited {
        s->visited = true;
       worklist = union
(worklist, s);
     }
  }  \\ while
}
```

In the above, the “visited" fields are assumed to have initial value of false.

topsort

- If a flow graph is a DAG, then the nodes are often processed by following a topological sort, or a topsort, traversal
- In a topsort traversal, a node is always processed before any of its successors.
- NOTE: In general, topsort is nonunique.

Relationship between traversals

- Given a DAG, G, clearly a post-order traversal is not a topsort traversal
- But a pre-order traversal may also not necessarily be a topsort traversal
  - See an example
- How to we find a topsort traversal then?

Reversed postorder numbering (rPost numbering)

```
main {
  counter = number_of_nodes;
  dfsearch(root);
}

function dfsearch(n) {
  n->visited = true;
 for each successor, s, of n {
    if not s->visited {
      dfsearch(s);
    }
    n->rpost = counter;
    counter = counter -1 ;
}
```

NOTE: n->rpost is called the reversed post-order number, or rpost number of n

Rpost number is assigned following a postorder traversal of G under a DFST

Properties of rPost numbering

- It maintains topological sort for any DAG, G
  - If (a, b) is an edge in G, we have rPostNum(a) < rPostNum(b)
  - We leave the proof of this property as a homework question
- It maintains the dominance relationship
  - If a dom b, then we have rPostNum(a) < rPostNum(b)
- In the following we introduce the concept of dominance relationship in a flow graph

Dominators and Postdominators

- In a flow graph G, if every path from the entry to b must contain a, then *a* is said to *dominate* b, written as *a dom b*.
  - By definition, we have a dom a.
- If every path from a to the exit must contain b, then a is said to be post-dominated by b, written as *b pdom a.*
  - *In most literature, a is defined as pdom a.*
  - *Hence, b pdom a in G if and only if b dom a in the reversed graph of G.*

The dominator tree

- The dominance relationship is transitive
- Hence the nodes in a flow graph G form a tree under the dominance relationship
  - Called the dominator tree, or the dom tree
  - (a, b) is an edge in the dom tree if and only if a is the *immediate dominator* of b
  - Node a strictly dominates b, if a dom b, and a ≠ b

A simple algorithm to compute dominators

- For each node n in flow graph G, initialize dom(n) = {n}? Would this work?
- Until no change, do {
    - For each n in G, do {
    - Old_dom = dom(n)
    - dom(n) ={n} $\cup(\bigcap_{all\ predecessor\ p} dom(p))$
    - If Old_dom ≠ dom(n), set change to true
    - }
- }
- Consider implementing this based on a worklist
- How do we prove that this will compute the exact DOM set for each node? [This will be a thought exercise. We will discuss again in the next lecture.]

Back edges

- If a $\rightarrow$ b is an edge such that b dominates a (a and b are not identical), a $\rightarrow$ b is called a back edge
- A back edge must be a retreating edge regardless what DFST we build for the given graph.
- A back edge uniquely defines a "natural loop"
  - Textbook gives an algorithm to find all instructions belonging to a natural loop
- Concepts of
  - Structured programs
  - Irreducible graphs
  - Strongly connected components
  - Cycles