Control Flow Analysis

Overview

Control flow analysis discovers the hierarchical flow of control within each procedure. Typically *flowgraphs* are used for the static representation of procedures. There are two approaches to control flow analysis: one based on *dominators* and the other based on *interval analysis*. We focus on dominator-based analysis here.

Dominator-based analysis

Dominator-based control flow analysis is used together with **iterative dataflow analysis** later. Here are the steps:

1) identify *basic blocks*,
2) build a *flowgraph* of the basic blocks, and
3) build *dominance tree* to identify *loops*.

Basic blocks

A **basic block** is a *maximal straight-line sequence of code* which can be entered only at the top and exited only at the bottom. There is no branchings (jumps) into or out of the 'middle' of the basic block.

Basic blocks are identified using the notion of **leaders**, i.e. the first instructions in some basic block. The following are the leaders:

- any first instruction in the code is a leader.
- any instruction that is the target of a `jump` is a leader.
- any instruction that follows a `jump` is a leader.

Then, for each leader, its basic block consists of *itself and all instructions up to but not including the next leader or the end of the program*.

The `call` instruction may or may not be considered as a basic block boundary. Usually we don't need to consider it as a boundary (and this allows larger basic blocks). However, there are cases when we need to. For example, Fortran alternate returns necessitates calls to be considered as boundaries and C `setjmp()` and `longjmp()` also necessitates it.

Flowgraphs

A flowgraph is a directed graph $G = (V \cup \{s,t\}, E)$, where V is a set of basic blocks and E is a set of all directed edges between basic blocks. $\langle u, v\rangle \in E$ iff basic block v *can* immediately follow u in an execution. Two special nodes, s and t are entry and exit blocks added for the convenience in later dataflow analysis. A **branching node** is one with outdegree > 1 and a **join node** is one with indegree > 1.

A *strongly connected subgraph* of a flowgraph is called a **region**.

Dominators and postdominators

To determine the loops in a flowgraph, we first define a binary relation called **dominance** on flowgraph nodes.

A node u **dominates** node v, denoted by u dom v, if every possible execution path from the entry node to v includes u. Node u is called a **dominator** of node v. Node u is the **immediate dominator**, denoted by idom(v), of node v if there is no other dominator of v inside the u–v path.

$$u \text{ dom } v: \qquad \begin{tikzcd} s \arrow[r, squiggly] & \boxed{u} \arrow[r, bend left] \arrow[r, bend right] & v \arrow[r, squiggly] & t \end{tikzcd}$$

Also, a node u **postdominates** node v, denoted by u pdom v, if every possible path from v to the exit node t includes u.

$$u \text{ pdom } v: \qquad \begin{tikzcd} s \arrow[r, squiggly] & v \arrow[r, bend left] \arrow[r, bend right] & \boxed{u} \arrow[r, squiggly] & t \end{tikzcd}$$

Strict dominance and dominance frontier

A node u **strictly dominates** v if u dom v and $u \neq v$. Node u is an **ancestor** of v if there is a u–v path of dominator tree edges.

The **dominance frontier** of a node u, DF(u), is the set of all nodes $\{v\}$ such that u dominates a predecessor of v but u does not strictly dominates v. That is, DF(u) is the set of nodes where u's dominance stops.

$$\begin{tikzcd} & \boxed{u} \arrow[r, mapsto] & w_1 \arrow[dr] & & \\ s \arrow[ur, squiggly] \arrow[r, squiggly] & w_2 \arrow[rr] & & \underline{v} \arrow[r, squiggly] & t \end{tikzcd}$$

Finding dominators

The dominators of a node $v \in V$ are given by the maximal solution to the following equations:

$$\begin{aligned} \text{dom}(s) &= \{s\}, \\ \text{dom}(v) &= \bigcup_{u \in \Gamma^{-1}(v)} dom(u) \cup \{v\} \end{aligned}$$

Dominator tree

A *dominator tree* of a flow graph $G = (V, E)$ is a subgraph $T = (V, E')$ of G, where

$$E = \{\langle u, v\rangle : \ u \text{ idom } v\}.$$

Note that the resulting subgraph is a tree since each node has exactly one immediate dominator except for the entry node. Dominator tree can be computed using Lengauer-Tarjan algorithm.

Depth-first traversal of dominator trees

A depth-first traversal of dominator trees paritions the set of edges into one of four catetories: *tree edges*, *back edges*, *forward edges*, and *cross edges*.

A **back edge** in a flowgraph is one where *the head dominates its tail*. Back edges are critial in the identification of *natural loops*.

Loops in flowgraphs

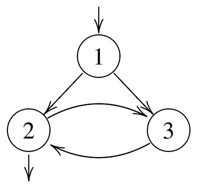
A **loop** in a flowgraph is a set L of nodes including a **header node** h with the following properties:

- From any node in L, there is a directed path leading to h.
- There is a directed path from h to any node in L.
- There is no edge from any node outside L to any node in L other than h.

Given a back edge $\langle v, h\rangle$, the **natural loop** of the back edge $\langle v, h\rangle$ is the subgraph consisting of 1) the node h (known as **loop header**) and 2) the set of nodes $\{u\}$ dominated by h where there exists a u–v path not passing through h. The header of this loop will be h. Simply, a natural loop is a loop with a *single loop header*.

Reducible flowgraph A flowgraph $G = (V, E)$ is **reducible** or **well-structured** iff E can be partitioned into disjoint sets, E_F of *forward edges* and E_B of *back edges*, such that (V, E_F) forms a DAG in which each node can be reached from the entry node, and the edges in E_B are all back edges. In other words, a flowgraph is reducible iff all the loops in it are *natural loops* which can be characterized by back edges. In reducible flowgraphs, any cycle of nodes have a unique header node.

Irreducible flowgraphs are graphs which has "cyclic core" as shown below, after node collapsing and edge deletions. Note that in the cycle, neither node 2 or node 3 dominates the other (the edges between nodes 2 and 3 are cross edges not back edges). Such graphs complicate the control flow analysis.



When common control-flow constructs such as **if-then**, **if-then-else**, **while-do**, **repat-until**, **for**, and **break** can only generate reducible flowgraphs. The use **goto** statements can introduce irreducible flowgraphs.

Loop-nest trees If L_1 and L_2 are two loops with headers h_1 and h_2, respectively, such that $h_1 \neq h_2$ and h_1 dominates h_2, then the nodes of L_2 are a proper subset of the nodes of L_1. We say that loop L_2 is **nested within** L_1, or that L_2 is a **inner loop**.

We can construct a **loop-nest** tree of loops in a program.

- Compute dominators of the flowgraph $G = (V, E)$.
- Construct the dominator tree $T = (V, E')$.
- Find all natural loops, and thus all the loop-header nodes.
- For each loop header h, merge all the natural loops of h into a single loop, loop[h].
- Construct the tree of loop headers (and implicitly loops), such that h_1 is above h_2 in the tree if h_1 dominates h_2.

The leaves of the loop-nest tree are the **innermost loops**.

Loop preheader Many loop optimizations will insert statements immediately before the loop executes. For example, *loop-invariant hoisting* moves a statement from inside the loop to immediately before the loop. **Loop preheaders** expediates the insertion of statements *before* the loop entry. For any loop entry nodes which has $n > 1$ indegree, we create one more node right before the loop entry node. e.g.