CS 4120 Lecture 26 — Control-flow analysis — 28 October 2011

Lecturer: Andrew Myers

1 Loops, dominators, natural loops, control trees, dominator analysis

These topics are described in these slides.

2 Using dominators for SSA conversion

We have seen that SSA form is a convenient form for optimization and analysis of code. However, converting code to SSA poses difficulties. In the previous lecture we observed that conversion could be broken down into two steps:

1. Insert uses of ϕ-functions for various variables at the beginnings of basic blocks; that is, for a variable x that needs to use a ϕ-function, we insert $x = \phi(x, x)$.

2. When enough uses of ϕ have been inserted, each use of a variable is reached by just one definition. Therefore, we can give each definition its own unique variable name, and rename all the corresponding uses reached by that definition. Note that the definitions of a variable include the new definitions using ϕ that were inserted in step 1.

In Step 1 we don't want to use ϕ more often than necessary, because this will create unnecessary variable names and impede optimization. Previously we identified the *path convergence* criterion as a way to determine where ϕ-functions are needed: intuitively, ϕ is needed at a node when it is the earliest place that two paths from different definitions converge. However, the path convergence criterion does not naturally lead to an efficient algorithm for finding these locations.

The notion of dominators can be used to efficiently insert ϕ exactly where the path convergence criterion says it should. The intuition is that if a node n defines a variable x, the path convergence criterion will not demand that ϕ be used for x at any node dominated by n where the definition reaches. For example, in Figure 1, the nodes inside the colored boundary are all dominated by node n, and ϕ is not needed. Unless there is another definition of x inside the dominated region, the definition at n must be the only one that reaches the node. Thus, node m does not need a ϕ-function for x, even though it has two predecessors. On the other hand, node n' does need a ϕ-function, because it has a predecessor dominated by n, yet it is not itself dominated by n.

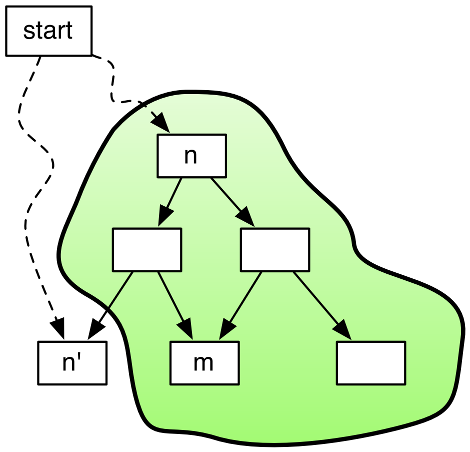

Figure 1: Dominators and SSA

We say that an edge crossing from a node dominated by n to a node not dominated by n lies on the *dominance frontier* for n. And we consider the destination node of that edge (such as n') to also lie on the dominance frontier. The nodes lying on the dominance frontier of n are exactly the nodes that to have the new definition $x = \phi(x, x)$ added to them.

Notice that adding this definition indeed adds a new definition to the control flow graph. And this new definition has its own dominance frontier that may induce additional definitions using ϕ. However, this *iterated dominance frontier* process will eventually converge on a set of ϕ definitions such that every node on the dominance frontier of every definition of a variable x starts with a corresponding definition $x = \phi(x, x)$.

3 Computing the dominance frontier

Let $DF(n)$ denote the dominance frontier of node n: the set of nodes not dominated by n, but with a predecessor dominated by n. Assuming we have computed the dominance relation, we can easily check whether any given node lies on the dominance frontier of node n. This observation leads to an obvious quadratic algorithm.

However, we can make the computation of dominance frontiers more efficient by observing that every node on the dominance frontier of n is either:

- a direct successor of n
- on the dominance frontier of some child c of n in the dominator tree.

Thus, to compute the dominance frontier of n, we recursively compute the dominance frontier of each of n's children in the dominator tree, then iterate over the childrens' dominance frontiers and the direct successors, checking whether each of these nodes is on the dominance frontier of n.