CS 5363, Lecture Notes
Static Single Assignment Form
Fall 2013

1 Reading / Source Material

- Cooper, Section 9.3

2 Objectives

1. determine the set of dominators, set of strict dominators, immediate dominator, and dominance frontier of a a node in a control flow graph
2. find the dominator tree for a control flow graph
3. explain how SSA serves as a factored use-def chain for efficient global optimizations
4. translate a small program in three-address code into SSA
5. translate a small program (not exhibiting critical edges or mutually dependent ϕ-functions) from SSA into normal three-address code
6. identify critical edges in a control flow graph

3 Vocabulary

static single assignment form, ϕ-functions, dominate, dominance frontier, immediate dominator, minimal SSA, pruned SSA, semi-pruned SSA, join node, critical edge

4 Outline

1. dominance
   (a) definitions

   dominate In a control flow graph, node A *dominates* node B iff every path from the start node to B must go through A.

   strictly dominates In a control flow graph, A node A *strictly dominates* node B iff A dominates B and A is not B.

   immediate dominator In a control flow graph, A node A *immediately dominates* node B iff A strictly dominates node B and there is no node C such that A strictly dominates C and C strictly dominates B.

   dominator tree The *dominator tree* for a control flow graph is the tree containing the same nodes as the control flow graph, whose root is the start node, and where every other node's parent is its immediate dominator in the control flow graph.

(b) Remarks:

- Every node dominates itself.
- The strictly dominates relation can be derived as the transitive closure of the immediate dominates relation.
- The dominates relation can be derived as the reflexive closures of the strictly dominates relation.
- The start node has 1 dominator (itself) and 0 strict dominators.
- Every node, except for the start node has exactly one immediate dominator.

(c) Dataflow Equation:

$$\textsc{Dom}(b_1) = \begin{cases} \{b_1\} & \text{if } b_1 \text{ is the entry node} \\ \left(\bigcap_{b_2 \in preds(b_1)} \textsc{Dom}(b_2)\right) \cup \{b_1\} & \text{otherwise} \end{cases}$$

2. static single assignment form

(a) definition and purpose

i. single name per static location

ii. combine data and control flow information into one data structure

A. variable names are isomorphic with defining instructions

B. the name of the variables used as an operand implicitly links the instruction using a variable to the instruction defining the variable

C. transitively following these links yields a "use-def" chain

D. it is sometimes said that SSA provides an implicit factored use-def chains

E. minimizes redundancy versus using a side-structure for explicit def-use information

- efficient to use, efficient to maintain as the program is transformed

iii. join nodes and ϕ-functions

A. join node = control flow graph node with multiple predecessors

B. multiple instances of an original source program may reach the join node on different paths

C. ϕ-functions

- appear at the top of join nodes
- define values in join-node based on values in predecessors, at runtime, based on which predecessor control actually came from

(b) dominance and SSA

i. an SSA variable is never overwritten, so we can use it anywhere we can ensure that it is defined

ii. an SSA variable is always defined when executing code in the region dominated by the definition

iii. the dominated region is effectively the scope of the SSA variable

(c) translation into SSA

i. dominance frontier
   A. The *dominance frontier* of a node A is the set of nodes that are not themselves strictly dominated by A but are successors of nodes dominated by A.
ii. insert ϕ-functions at dominance frontier (join nodes)
iii. rename variables to establish single assignment

(d) translation out of SSA form
i. replace ϕ-functions with register-to-register moves in predecessors
ii. but, semantics of ϕ-functions are that they all ”happen” simultaneously as the execution transitions into the join node from its predecessor.
iii. critical edges and lost-copy problem
   A. A critical edge in a control flow graph is an edge whose source is a node with multiple successors and whose destination is a node with multiple predecessors (i.e., a join node).
   B. The presence of critical edges will break naive translation out of SSA form by inserting moves into predecessor blocks, because the moves may be executed even when the edge to the join node is not taken.
   C. simple solution: ”break the critical edge”
      - add a new (initially empty) node, replace the critical edge with an edge from the critical edge’s source to the new block and an edge from the new block to the critical edge’s successor.
      - Neither of the new edges will be critical.
iv. mutually dependent ϕ-functions and swap problem
   A. the operands of ϕ

Warning: textbook definition of semi-pruned is based on “non-local” values that are live across block boundaries. This doesn’t mean that the value only exists within a single block, which is what would normally be meant by “local.”