Static Single-Assignment (SSA) Form

In pure functional languages, every variable is assigned a value exactly once. In ordinary imperative languages, a variable may be modified multiple times.

Recently, it was found that the single-assignment property can greatly simplify data-flow analysis and various compiler optimization algorithms.

Transforming a procedure to a pure single-assignment form, unfortunately, would require excessive $renaming$. A practical compromise is to settle for $static$ single assignment (SSA).

With SSA, the internal representation of a procedure is transformed such that each distinct static definition of a variable is given a distinct name. Where two distinct definitions of the same variable meet at a join point (through different execution paths), a ϕ function is introduced to $select$ from the two values and assign the result to an artificially introduced definition.

The ϕ function is normally left uninterpreted. In the program's implementation, the two joined definitions normally will have to be bound to the same memory location. So, the SSA essentially serves only as an aid to compiler analysis without known significance in the actual execution of the program.

Under SSA, *every static use of data has a unique reaching definition.*

Next we study an algorithm to translate a procedure to minimal SSA form, i.e. an SSA form with a minimal number of ϕ functions.

First, introduce the definition of $dominance\ frontier$ of a given flowgraph node x, written $DF(x)$:

$$DF(x) = \{y \mid (\exists z \in Pred(y) s.t. x\ dom\ z) \& x\ !sdom\ y\} \quad (1)$$

NOTE 1: Recall that dom is reflexive, i.e. $x\ dom\ x$.

NOTE 2: $!sdom$ means "does not strictly dominate".

Informally, and somewhat sloppily, one understands that $y \in DF(x)$ if x "almost" dominates y.

The textbook presents an algorithm to compute $DF(x)$ for all x, based on the following equations:

$$DF_{local}(x) = \{y \in Succ(x) \mid idom(y) \neq x\} \quad (2)$$

$$DF_{up}(x, z) = \{y \in DF(z) \mid idom(z) = x \& idom(y) \neq x\} \quad (3)$$

$$DF(x) = DF_{local}(x) \cup \bigcup_{z \in N(idom(z)=x)} DF_{up}(x, z) \quad (4)$$

The above equations are turned into code for computing $DF(x)$ for all x.

Next, the definition of *iterated dominance frontier* $DF^+()$ is defined as follows.

$$DF(S) = \bigcup_{x \in S} DF(x) \tag{5}$$

$$DF^+(S) = lim_{i \to \infty} DF^i(S) \tag{6}$$

where $DF^1(S) = DF$ and $DF^{i+1}(S) = DF(S \cup DF^i(S))$,

If S is the set of nodes that assign to variable x, plus the `entry` node, then $DF^+(S)$ is exactly the set of nodes that need ϕ-functions for x. (Why is this so?)