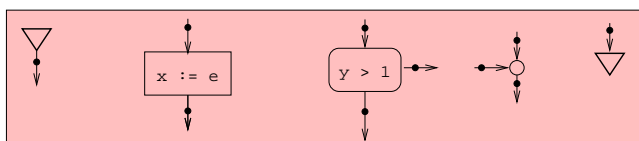


Program Analysis and Verification

Practice Questions in Data-flow Analysis

1. A typical data-flow analysis framework (for programs without procedures) comprises:

- An underlying complete lattice $L = (D, \leq)$ of data values, with $d_1 \sqcap d_2$ and $d_1 \sqcup d_2$ denoting the meet and join of data values d_1 and d_2 , and \perp and \top denoting the least and greatest element respectively.
- A “straight-line” transfer function $f_n : D \rightarrow D$ for each node n in the program. Following Cousot and Cousot, programs are viewed as flow-charts in which there are 5 kinds of program statements or nodes: *Entry* nodes, *assignment* nodes, *if* nodes, *join* nodes, and *exit* nodes, as shown in the diagram below.



The straight-line transfer function simply says how a data value d on an incoming edge is transformed to a data value d' on a given outgoing edge.

For a join node, the transfer function is simply the identity function. However, for a join node, which is the only kind of node with multiple incoming edges, we can also specify how, given data values for each incoming edge, how the transformed value on the (unique) outgoing edge is computed. Typically, this computation is either a meet or join of the incoming data values.

The framework is called a meet or join framework depending on whether this computation at join nodes takes the meet or join of the incoming data values.

2. We are interested in computing the meet-over-all-paths (MOP) values at each program point (i.e. at each edge of the program). A *path* in the program is a sequence of nodes $p = n_0 n_1 \cdots n_k$ starting at an entry node and with each (n_i, n_{i+1}) an edge in the program. The straight-line function f_p computed by the above path p is defined to be simply the composition of the straight-line functions for each node, i.e. $f_p = f_{n_k} \cdot f_{n_{k-1}} \cdots f_{n_0}$. Then the MOP value at a program point N is defined to be

$$\bigwedge_{\text{paths } p \text{ to } N} f_p(\perp).$$

For a join framework, the MOP value is defined with join instead of meet. In the sequel we assume a meet framework.

3. A data-flow framework induces in a natural way a system of equations

$$\begin{aligned}
 x_E &= f_E(-) && \text{initial value for an entry node } E \\
 x_N &= f_n(x_M) && \text{for an assignment node } n \text{ with incoming edge} \\
 &&& M \text{ and outgoing edge } N \\
 \dots &&& \text{etc.}
 \end{aligned}$$

These equations lead us to consider a (complete) lattice of vectors (each vector v is a map from $PP \rightarrow D$, where PP is the finite set of program points), where the ordering \leq is lifted in the natural way to the set D^{PP} . The transfer functions induce a map $F : (PP \rightarrow D) \rightarrow (PP \rightarrow D)$ defined in a natural way as $F(v)(N) = f'_n(v)$, where n is the unique node for which N is an outgoing edge, and where by $f'_n(v)$ we mean

- $f_n(-)$ if n is an entry node.
- $f_n(v(M))$ if n is an assignment node with incoming edge M .
- $f_n(v(M))$ if n is an “if” node with incoming edge M and N is the “true” (resp. “false”) branch, and f_n is the function for the “true” (resp. “false”) branch.
- $v(M_1) \sqcap \dots \sqcap v(M_k)$ where n is a join node and M_1, \dots, M_k are the incoming edges of n .

Note that solutions to the equations above are precisely the fixpoints of the function F .

When the transfer functions f_n are all monotonic (i.e. $d \leq d'$ implies $f_n(d) \leq f_n(d')$), it follows that F is also monotonic. In this case, by the Knaster-Tarski theorem F has (a complete lattice of) fixpoints. In particular, F has a maximal fixpoint which we refer to as the MFP solution.

4. **Claim 1** *When the given framework is monotonic, the MFP vector is an underapproximation of the MOP vector (i.e. $MFP \leq MOP$).*
5. The framework is called distributive if each f_n is distributive (i.e. $f_n(d_1 \cap d_2) = f_n(d_1) \cap f_n(d_2)$).

Claim 2 *A distributive function is also monotonic.*

Claim 3 *For a distributive framework, we have $MFP = MOP$.*

6. Computing the MFP value:

- (a) Kleene’s iteration algorithm can be described as follows: Begin with value (\top, \dots, \top) and repeatedly apply F on it. Stop and output the current value if you get the same value as in previous iteration.

Claim 4 *For a monotone framework with no infinite descending chains, Kleene’s iteration algorithm computes the MFP value.*

(b) Kildall's algorithm can be described as follows:

- Initialize value at each program point to \top .
- Mark initial nodes
- Choose a marked node n , unmark it, and propagate value to its successors based on f_n , by taking meet with existing value.
- Mark successor nodes for which new value is smaller than old value.
- Stop when no marked nodes are left.

Claim 5 *Kildall's algo terminates whenever L has no infinite descending chains.*

Claim 6 *For a monotone framework with no infinite descending chains, Kildall's algorithm computes the MFP value.*

Corollary 1 *For a distributive framework with no infinite descending chains, Kildall's algorithm computes the MOP value.*

Prove (or disprove!) the claims above.

- For programs with procedures (the inter-procedural case) we have seen two approaches – the functional approach and the call-strings approach, for computing the meet-over-all-valid-paths (MVP) for distributive frameworks. Speculate what would happen if the framework was only monotonic.
- Compute the MVP values for the program below using (a) the functional approach and (b) the call-strings approach (for the latter case stop with call-strings of length 4).

