CS615 Autumn 2006 Hints to Quiz 7 Solution

How do we execute a program in an abstract domain? As studied in class, starting from an abstract precondition (a, b, c, d, e, f), the best abstract postcondition after executing a program statement is obtained by computing the postcondition of $\gamma((a, b, c, d, e, f))$ in the concrete domain, and then abstracting this concrete postcondition (recall the operator $\alpha(F(\gamma(\hat{S})))$ we studied in class). Also, when iterating through a loop, in the concrete domain, we keep collecting sets of states at every program location in the loop using the union operator (*lub* operator in concrete domain). In the abstract domain, we have to use the corresponding *lub* operator instead of the union operator. You may even want to replace the *lub* operator in the abstract domain by widening, but then your invariants may not be strong. However, what you'll get will still be a correct invariant.

Let the current abstract element at program location `L0` be (a, b, c, d, e, f). The abstract element we'll get at `L3` by pushing (a, b, c, d, e, f) through the body of the loop is obtained by concretizing $(\max(31, a)$, b, c, $\min(99, d)$, $e, f)$ (this is to take care of the conditions `(y < 100)` and `(x > 30)`, computing the postcondition of this concretization under `L2: x := x - y`, and then abstracting the concrete postcondition. Similarly, the abstract element at `L7` is obtained by concretizing $(a, \min(30, b), c, \min(99, d), e, f)$ (to take care of the conditions `(y < 100)` and `(x <= 30)`), computing the postcondition of this under `L6: x = x + y`, and abstracting the concrete postcondition. Finally, to compute the abstract precondition of statement `L8: y := y + 1`, we must compute the *lub* (in the abstract domain) of the postconditions obtained at `L3` and `L7`. By pushing this through `L8: y := y + 1`, i.e. by concretizing, computing the concrete postcondition and abstracting, we can obtain the abstract postcondition at `L9`. Before the next iteration of the while loop starts, we must compute the *lub* of this abstract postcondition at `L9` with the abstract precondition at the beginning of the current iteration, i.e. (a, b, c, d, e, f). If you use the widening operator here, instead of the *lub* operator, you will converge faster.

In the interests of time, I am not showing the abstract execution below. However, you can work this out by computing the *lub*s for the first few iterations and then start widening. When widening, it is advantageous to use threshold widening keeping as thresholds some important constants that appear in the condition checks in program (or are derived from it), e.g. 101 (from the check on y at the beginning of the while loop), 31 (from the check on x in the if-then-else conditions), $1 + \sum_{i=1}^{99} i$ (from the maximum value that x can take at `L7` if statement `L6` was executed in each iteration), and $31 - \sum_{i=1}^{99} i$ (from the minimum value that x can take at `L7` if statement `L2` was executed in each iteration.

Also, after you have obtained the abstract loop invariant at `L0` using the above method, you can tighten it further by concretizing it, and then pushing the concretized version through the loop body in the concrete domain a few times (at least once) and then abstracting the resulting postcondition. This is a consequence of the fact that the concretization of an abstract loop invariant is always a postfix point in the concrete lattice, and therefore computing its postcondition in the concrete domain can only (and very often significantly) help you in tightening the loop invariant in the concrete domain. Once you obtain this tighter loop invariant in the concrete domain, you can abstract it to get a loop invariant in the abstract domain.