Listing 1: C++ code using listings

```
int main(void)
{
    int a, d;
    // forall variable
    klee_make_symbolic(&a, sizeof(a), "a_sym");
    // PSE variable : Uniformly distributed [0 to 650]
    make_pse_symbolic<int>(&d, sizeof(d), "d_prob_sym", 0, 650);
    int c = a + 100;

    // case 1 -> Pure Forall Predicate
    if (a > 50) {
        c = a + 75;
    }   else    {
        c = a - 75;
    }

    // case 2 -> Pure PSE Predicate
    if (d > 60)
        d = 250;

    // case 3 -> Complex Case
    if (c > d)
        c = d;

    return 0;
}
```

Algorithm 1 Complex Case : (Testing Based Estimation)

```
for each p ∈ Paths do
    c := ConstraintSet(p)                          ▷ Path Constraints for p
    m := Solve(c)                                  ▷ solution for the path constraints
    forallConcreteSet = { }
    for each v ∈ ForallVars(p) do
        concreteSet.append({key : v, val : m[v]})  ▷ forall values
    end for each
    executeCV(program, concreteSet, c)
end for each
```

Algorithm 2 Complex Case : (k-samples)

for each $p \in Paths$ **do**
    c := ConstraintSet(p) ▷ Path Constraints for p
    m := Solve(c) ▷ solution for the path constraints
    forallConcreteSet = { }
    for each $v \in ForallVars(p)$ **do**
        concreteSet.append({key : v, val : m[v]}) ▷ forall values
    end for each
end for each

Algorithm 3 PSE Sampling Algorithm

function PSESAMPLE($\mathtt{x}, de, \varphi, \sigma, P, I$) δGenerate an unused name for a probabilistic symbolic variable
    $P[\delta] = de$
    $I[\delta] = \{\delta\}$
    $\sigma[\mathtt{x}] = \delta$
    return (φ, σ, P, I)
end function