

PSE Quant Sampling Algorithm

Sumit Lahiri

March 4, 2021

We try to formulate a way to compute path probabilities using symbolic execution and testing based technique.

```
int main(void)
{
    int a; // uninitialized
    int d = std::uniform_distribution<rd_seed>(0, 650);

    // forall variable : (INT_MIN to INT_MAX)
    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 : Dependence Case
    if (c > d) c = d;

    // Probabilistic query : assert(P(c != d) < 0.5)
    // Optimize here :
    //     Optimal value of forall (a) such that P(c != d) is close to 0.5
    return 0;
}
```

Algorithm 1 Candidates : (Testing Based Estimation)

```
1: for each  $p \in Paths$  do
2:    $c := ConstraintSet(p)$  ▷ Path Constraints for p
3:    $m := Optimize(query, c)$  ▷ solution for the path constraints
4:    $concreteSet = \{\}$ 
5:   for each  $v \in ForallVars(p)$  do ▷ ForallVars p → forall
6:      $concreteSet.append(\{key : v, val : m[v]\})$  ▷ Candidate Values
7:   end for each
8:    $executeCV(program, concreteSet)$ 
9: end for each
```

Algorithm 2 executeCV : PSE Sampled Normal Execution

```
1: function EXECUTECV( $P : program, C : concreteSet$ )
2:   for each  $v \in ForallVars(p)$  do
3:      $value(v) := concreteSet(v)$  ▷ Use values from ConcreteSet
4:   end for each
5:   ... ▷ proceed with normal execution
6: end function
```

For the sample program given above, we first resort to using **symbolic execution** to generate **path constraints** for all the feasible paths that this program can take and then convert the **path constraints** into an **formal logic** optimization problem that gives an **assignment** to **forall** variables such that it leads to optimum violation of the *query*.

```
def generateCandidates(k: int): # Candidates Algorithm
    opt = z3.Optimize()
    a = z3.Int("a_sym")
    d = z3.Int("d_prob_sym")

    opt.add(d >= 0)
    opt.add(d <= 650)
    opt.add(a > 50)
    opt.add(z3.Not(d > 60))
    opt.add(a + 75 > d)

    opt.maximize(a - d - 75) # Query to optimize
    n = 0
    while opt.check() == z3.sat and n != k:
        m = opt.model()
        n += 1
        print("%s = %s" % (a, m[a]))
        print("%s = %s" % (d, m[d]))
        opt.add(a != m[a])
```

We now explore a slightly different example which is more involved in terms of the constraints and query that the user can pose at the end of the symbolic execution.

```

int a, b, c, d;

// forall variables : (INT_MIN to INT_MAX)
klee_make_symbolic(&a, sizeof(a), "a_sym");
klee_make_symbolic(&b, sizeof(b), "b_sym");
klee_make_symbolic(&c, sizeof(c), "c_sym");

// PSE variables
make_pse_symbolic<int>(&d, sizeof(d), "d_prob_sym", 0, 500);

// PSE variable : Random Sampling
std::default_random_engine generator;
std::uniform_int_distribution<int> distribution(0, 500);

if (a + b > c + d)
{
    if (a > b) {
        a = 100;
        b = 500;
    } else {
        a = 500;
        b = 100;
    }
} else {
    if (c > d) {
        a = 100;
        c = 100;
        b = 600;
        d = distribution(generator);
    } else {
        a = 600;
        c = 600;
        b = 100;
        d = distribution(generator);
    }
}

if (a + c > b + d)
{
    d = distribution(generator);
}

```

Below we show a sample of the type of the queries that a user can make in

the context of the example shown above.

```

assert(a + b + c + d <= 1100);

Case : 1
[Query Parse] : P(a + b + c + d <= 1100) >= 0.5
Query : assert fails atleast half of the times.

Case : 2
[Query Parse] : P(a + b + c + d <= 1100) <= 0.5
Query : assert fails atmost half of the times.

Case : 3
[Query Parse] : P(a + b + c + d <= 1100) == 0.5
Query : assert fails exactly half of the times.

```

Based on the query posed, we get candidate models by converting the code into a optimization query and solve this optimization problem using any off-the-shelf SMT Solver to get model values for the forall variables that contribute to maximum violation of the query constraint.

```

Path : [And(d_prob_sym >= 0, d_prob_sym <= 500),
a_sym + b_sym > c_sym + d_prob_sym,
a_sym > b_sym, 500 + d_prob_sym < 100 + c_sym]
Model : 1
    a_sym = 402
    b_sym = 0
    c_sym = 401
Model : 2
    a_sym = 404
    b_sym = -1
    c_sym = 402
Path : [And(d_prob_sym >= 0, d_prob_sym <= 500),
a_sym + b_sym > c_sym + d_prob_sym,
a_sym > b_sym, Not(500 + d_prob_sym < 100 + c_sym)]
Model : 1
    a_sym = 1
    b_sym = 0
    c_sym = 0
Model : 2
    a_sym = 0
    b_sym = -1
    c_sym = -2
Path : [And(d_prob_sym >= 0, d_prob_sym <= 500),
a_sym + b_sym > c_sym + d_prob_sym,
Not(a_sym > b_sym), 100 + d_prob_sym < 500 + c_sym]
Model : 1

```

```

a_sym = -199
b_sym = -199
c_sym = -399
Model : 2
a_sym = -200
b_sym = -197
c_sym = -398
Path : [And(d_prob_sym >= 0, d_prob_sym <= 500),
a_sym + b_sym > c_sym + d_prob_sym,
Not(a_sym > b_sym), Not(100 + d_prob_sym < 500 + c_sym)]
...
```

Here $P(\text{condition})$ represents the sum path probabilities for a given condition to hold with some deterministic value as posed in the queries shown above.

We now do a transformation pass over the program and instrument the count of the given condition failing. In the transformation pass, we make the program take values that we find as candidate models upon following Algorithm 1

```

int a, b, c, d, assert_satisfy = 0, termCount = 35000;

// forall variables take values from stdin.
// pass values that conform to a candidate model
scanf("%d", &a); // (INT_MIN to INT_MAX)
scanf("%d", &b); // (INT_MIN to INT_MAX)
scanf("%d", &c); // (INT_MIN to INT_MAX)

while(termCount--) {
    ...
    (code remains same)
    (run the code in a while loop)
    ...
    // Added by transformation pass
    // We take a count of how many times the
    // posed query is satisfied.
    if (a + b + c + d - 1100 <= 0) {
        assert_satisfy++;
    }
}
```