

# PSE Quant Sampling Algorithm

Sumit Lahiri

February 25, 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** Dependence Case : (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
```

---