

```

def count_heads(prob : float, n : int):
    x = 0
    while(n > 0):
        d = bernoulli.rvs(size=1, p=prob)[0]
        x = x + d
        n = n - 1
    return x

```

Fig. 1. Program to *count* heads over n bernoulli trials.

```

if (prob <= 0.5) E(x) < 0.4 * n else E(x) >= 0.6 * n

```

Fig. 2. Assert we want to check for failure.

```

double prob, path_prob = 1, choice_prob = 1;
int sum = 0, n = 0;

klee_make_symbolic(&prob, f"prob_sym_{i}");
klee_make_symbolic(&n, f"n_symbolic");
for i in range(n):
    int d;
    klee_make_symbolic(&d, f"d_sym_{i}");

    d = bernoulli(prob);
    (d == 1) ? choice_prob = prob : choice_prob = (1 - prob);
    path_prob = path_prob * choice_prob;

    sum = sum + d;

klee_dump(path_prob)
klee_dump(sum) // E[heads in "n" runs]

```

Fig. 3. Program listing for n bernoulli trails experiment with transformation for KLEE.

We consider a independent *bernoulli* trials here of flipping a fair coin " n " times.

$$choice_prob = \begin{cases} p & \text{if } d \text{ value is 1 corresponding to getting a "heads"} \\ 1 - p & \text{if } d \text{ value is 0 corresponding to getting a "tails"} \end{cases}$$

concretely, on the i^{th} run \vec{d} can have a value as below, one-hot encoded w.r.t the outcome of *heads* or *tails*.

$$\vec{d}_i = encode(< 0, 0, 0, 1, 1, 0, 1, 1, 1, 0 >)$$

Based on the value of the \vec{d}_i , we get w_i value using *choice_prob*.

$$w_i = (p)^{x_i} * (1 - p)^{n - x_i} \quad (1)$$

where x_i denotes the number of *heads* in the i^{th} randomized run and for n runs. $n = 10$ for the case in the above example.

$$Objective_1 = maximize(\sum_{i=1}^k (p)^{x_i} * (1-p)^{n-x_i}) \quad | \quad \forall(i, j) [x_i \neq x_j] \quad (4)$$

Fig. 4. Optimization Expression for k randomized paths

$$w_i = (p)^{x_i} * (1-p)^{10-x_i} \quad (2)$$

We consider top " k " randomized runs now for the optimization query. The expression for optimization thus becomes

$$maximize(\sum_{i=1}^k w_i)$$

On substituting the value of w_i from (1).

$$maximize(\sum_{i=1}^k (p)^{x_i} * (1-p)^{n-x_i}) \quad (3)$$

After performing the optimization above, we get different values of \vec{d}_i . For i^{th} randomized run, we get a single one hot encoded \vec{d} vector. We show below the encoding for a few i values.

$$\vec{d}_1 = encode(< 0, 0, 0, 1, 1, 0, 1, 1, 1, 0 >)$$

$$\vec{d}_3 = encode(< 0, 1, 0, 1, 1, 0, 0, 0, 1, 0 >)$$

$$\vec{d}_4 = encode(< 1, 0, 1, 0, 1, 0, 1, 1, 0, 1 >)$$

....

We run the optimization by renaming the k pse variables set appropriately and then impose the *distinct* clause so that we don't run the optimization on the same *randomized* runs again.

$$\forall(i, j) [\vec{d}_i \neq \vec{d}_j] \quad (5)$$

We approximate the value of *expected* heads in the above program, with the following equations.

$$w_i = \prod_{j=1}^n choice_prob_i(j), \quad sum_i = \sum_{j=1}^n components(\vec{d}_i), \quad (6)$$

$$EV(heads) = (\sum_{i=1}^k w_i * sum_i), \quad Error = n * prob - EV(heads) \quad (7)$$

where both w_i and sum_i can both be computed from the corresponding \vec{d}_i expression we get from the *model* of the *optimization* query Eq 4

For $k = 5$ & $n = 10$ the two constraint sets and optimization expressions are as follows :

$$w_1 = \prod_{j=1}^{10} choice_prob_1(j), \quad sum_1 = \sum_{j=1}^{10} components(\vec{d}_1), \quad (8)$$

$$w_2 = \prod_{j=1}^{10} choice_prob_2(j), \quad sum_2 = \sum_{j=1}^{10} components(\vec{d}_2), \quad (9)$$

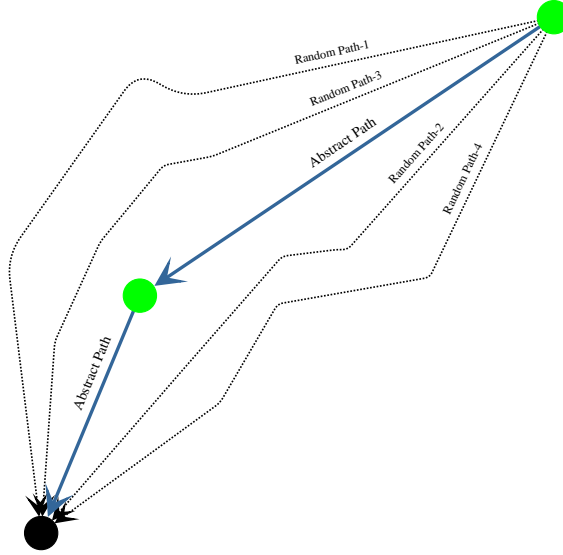


Fig. 5. Multiple *random* paths corresponding to different \vec{d} 's but a single *abstract* path.

prob = 0.435, n = 8 (8 coin flips), k = 240 (randomized paths)
 prob = 0.551, n = 5 (5 coin flips), k = 27 (randomized paths)

Fig. 6. Values for Assert failure.

$$w_3 = \prod_{j=1}^{10} \text{choice_prob}_3(j), \quad \text{sum}_3 = \sum_{j=1}^{10} \text{components}(\vec{d}_3), \quad (10)$$

$$w_4 = \prod_{j=1}^{10} \text{choice_prob}_4(j), \quad \text{sum}_4 = \sum_{j=1}^{10} \text{components}(\vec{d}_4), \quad (11)$$

$$w_5 = \prod_{j=1}^{10} \text{choice_prob}_5(j), \quad \text{sum}_5 = \sum_{j=1}^{10} \text{components}(\vec{d}_5), \quad (12)$$

$$\vec{d}_1 \neq \vec{d}_2 \neq \vec{d}_3 \neq \vec{d}_4 \neq \vec{d}_5 \quad (13)$$

$$EV(\text{heads}) = \left(\sum_{i=1}^k w_i * \text{sum}_i \right), \quad (14)$$

$$EV(\text{heads}) = (w_1 * \text{sum}_1 + w_2 * \text{sum}_2 + w_3 * \text{sum}_3 + w_4 * \text{sum}_4 + w_5 * \text{sum}_5) \quad (15)$$

$$\text{Objective}_2 = \text{maximize}(w_1 * \text{sum}_1 + w_2 * \text{sum}_2 + w_3 * \text{sum}_3 + w_4 * \text{sum}_4 + w_5 * \text{sum}_5) \quad (16)$$

We can make multiple optimization *objectives* be fulfilled in one query, but the best results are produced for the *Objective*₂ since it minimizes the *error* most.

$$\text{Objective}_2 = \text{maximize} \left(\sum_{i=1}^k w_i * x_i \right) \quad (17)$$

1 QUICKSORT EXAMPLE

Consider a pivot vector p_{abs}^{\rightarrow} that KLEE produces over a single abstract path. We consider a 10 element array as shown above. The *pivot* indices for p_{abs}^{\rightarrow} as chosen from a *random* distribution.

$$arr[10] = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 \end{bmatrix} \quad (18)$$

A possible assignment of the indexes choosen as *pivot* indices along this *abstract* path can be as shown below.

$$p_{abs}^{\rightarrow} = \begin{bmatrix} a_9 & a_8 & a_2 & a_3 & a_8 & a_8 \end{bmatrix} \quad (19)$$

For same forall values in $arr[10]$, we can have different random runs and on each run, the $pivot^{\rightarrow}$ vector will have a different assignment of values as indices. We are showing 5 different $pivot_{r_j}^{\rightarrow}$ vectors, each corresponding to a valid random run j .

$$pivot_{r_1}^{\rightarrow} = \begin{bmatrix} a_9 & a_4 & a_4 & a_2 & a_9 & a_9 \end{bmatrix} \quad (20)$$

$$pivot_{r_2}^{\rightarrow} = \begin{bmatrix} a_9 & a_9 & a_7 & a_7 & a_7 \end{bmatrix} \quad (21)$$

$$pivot_{r_3}^{\rightarrow} = \begin{bmatrix} a_9 & a_4 & a_1 & a_4 & a_9 & a_9 \end{bmatrix} \quad (22)$$

$$pivot_{r_4}^{\rightarrow} = \begin{bmatrix} a_9 & a_3 & a_1 & a_9 & a_6 & a_9 \end{bmatrix} \quad (23)$$

$$pivot_{r_5}^{\rightarrow} = \begin{bmatrix} a_9 & a_5 & a_3 & a_1 & a_9 & a_8 \end{bmatrix} \quad (24)$$

$$pivot_{r_6}^{\rightarrow} = \begin{bmatrix} a_9 & a_2 & a_1 & a_9 & a_6 & a_6 & a_9 \end{bmatrix} \quad (25)$$

$$pivot_{r_7}^{\rightarrow} = \begin{bmatrix} a_9 & a_9 & a_9 & a_5 & a_5 & a_9 & a_8 \end{bmatrix} \quad (26)$$

The probability mass associated with the i_{th} random run as is given below considering each *pivot* selection being an independent selection.

$$w_i = (1/n) * \prod_{j=1}^k 1/l_j * \prod_{j=1}^k 1/r_j \quad | \quad \text{for } k - \text{partitons} \quad (27)$$

$$\text{where } l_j = \text{count}(l_{terms}), \quad r_j = \text{count}(r_{terms}) \quad | \quad \text{in } j^{th} \text{ partition.} \quad (28)$$