```
def progam1(inpt):
        prob, _ = inpt
        z = 0
        flip = 0
        while (flip == 0):
                d = bernoulli.rvs(size=1, p=prob)[0]
                if d:
                        flip = 1
                else:
                        z = z + 1
        return z

assert(z > (1 - prob)/prob)
```

Fig. 1. Program listing for finding probabilistic assert violation.

```
int main()
{
        double prob;
        int d, z = 0, flip = 0;

        make_pse_symbolic(&flip, sizeof(flip), "flip_pse_sym", 0, 1);
        make_pse_symbolic(&prob, sizeof(prob), "prob_sym", 0, 1);
        klee_make_symbolic(&z, sizeof(z), "z_sym");
        klee_assume(z >= 0);

        std::random_device rd{};
        std::mt19937 rng{rd()};
        std::bernoulli_distribution rvs(prob);

        while (flip == 0) {
                int d = rvs(rng);
                if (d) {
                        flip = 1;
                } else {
                        // must be executed more
                        // for the assert to pass.
                        z += 1;
                }
        }

        return 0;
}
```

Fig. 2. `Translate()` of the program for testing the assert.

```
long long int termCount = 50000, unroll = 2500;
while (termCount-)
{
        flip = 0;
        z = 0;
        scanf("%Lf", &prob);

        ...

        while (flip == 0 && unroll-) {
                int d = rvs(rng);
                if (d) {
                        flip = 1;
                } else {
                        z += 1;
                }
                if (z > ((double)(1 - prob) / (prob)))
                        win++;
                flip_runs++;
        }
        program_runs++;
}
```

Fig. 3. Translate() of the program for testing the assert.

```
(flip_pse_sym <= 1),
(0 <= flip_pse_sym),
(prob_sym <= 1),
(0.000001 < prob_sym),
(0 <= z_sym),

FAIL : (z_sym * prob_sym - (1 - prob_sym) <= 0)
```

Fig. 4. Constraints without optimization step.

```
(flip_pse_sym <= 1),
(0 <= flip_pse_sym),
(prob_sym <= 1),
(0.000001 < prob_sym),
(0 <= z_sym),

FAIL : (z_sym * prob_sym - (1 - prob_sym) <= 0)
optimize : maximize(prob_sym)
```

Fig. 5. Constraints with optimization step.

Table 1. Comparing `prob(p)` value to cases Vs `assert` status for `optimization` over `prob` value

`Flip Runs`	`Program Runs`	`prob(p)`	`z (Last)`	`(1-p)/p`
49999	50000	0.9999990463	0	0.0000009537
49999	50000	0.9999995232	0	0.0000004768
49999	50000	0.9999998808	0	0.0000001192
49999	50000	0.9999999991	0	0.0000000009
49999	50000	1.0000000000	0	0.0000000000
50000	50000	0.9999847412	0	0.0000152590
50001	50000	0.9999694825	0	0.0000305184
50003	50000	0.9998779298	0	0.0001220851
50008	50000	0.9997558596	0	0.0002442000
50016	50000	0.9995117192	0	0.0004885193
50045	50000	0.9990234385	0	0.0009775161
50822	50000	0.9843750156	0	0.0158729998
53303	50000	0.9375000625	0	0.0666665956
57237	50000	0.8750001250	0	0.1428569796
99385	50000	0.5000005000	0	0.9999980000

Table 2. Comparing `prob(p)` value to cases where assert `failure` occurs for no `optimization` case

`Flip Runs`	`Program Runs`	`prob(p)`	`z (Last)`	`(1-p)/p`
49999	50000	1.0000000000	0	0.0000000000
12798121	50000	0.0039072461	121	254.9347362328
2118330	50000	0.0234384766	3	41.6648888947
3191133	50000	0.0156259844	27	62.9959681516
1592312	50000	0.0312509687	85	30.9990080819
802573	50000	0.0625009375	6	14.9997600036
404577	50000	0.1250008750	3	6.9999440004
199379	50000	0.2500007500	2	2.9999880000
133424	50000	0.3750006250	1	1.6666622222
99661	50000	0.5000005000	0	0.9999980000

Table 3. Comparing `prob(p)` value to cases where assert `passed` without `optimization` step.

`Flip Runs`	`Program Runs`	`prob(p)`	`z (Last)`	`(1-p)/p`
2551264	50000	0.0195322305	60	50.1974298071
6371788	50000	0.0078134922	169	126.9837458595
4280156	50000	0.0117197383	303	84.3261373592
1592312	50000	0.0312509687	85	30.9990080819
266112	50000	0.1875008125	7	4.3333102223