Symbolic Execution for Randomized Programs

Artifact Overview

Zachary Susag[1], Sumit Lahiri[2], Justin Hsu[3], and Subhajit Roy[4]

[1,3]Cornell University

[2,4]IIT Kanpur

In this document you will find detailed instructions on how to acquire, build, and reproduce the experiments found in the original paper.

1 Getting Started

1.1 Acquiring & Installing Plinko

We provide a prebuilt Docker image of our tool, Plinko, which contains all the necessary depedencies, tools, benchmarks, and scripts to reproduce the results of all experiments done in the paper. As such, the first step to get Plinko running is to first acquire Docker for your host operating system. We have only tested the Docker image on MacOS and Linux running the amd64 architecture; however, we do not forsee issues on Windows systems due to the compartmentalized nature of Docker. We do know that the Docker will fail if run on Apple Silicon devices, unfortunately.

To install Docker, follow the instructions found at `https://docs.docker.com/get-docker/`. While we will provide all necessary commands to get Plinko up and running, if you are unfamiliar with Docker, we suggest skimming the official Docker guide found at `https://docs.docker.com/get-started/`.

You can check if docker is up and running on your system by running the following command from your terminal.

```
$ docker run hello-world
```

To acquire PLINKO's Docker image, create a container from the downloaded image, and get an interactive terminal from inside the container, execute the following in your terminal:

```
$ docker pull zsusag/plinko:latest
$ docker run --name oopsla535_plinko -it --ulimit='stack=-1:-1' zsusag/plinko:latest
```

This will place you in an interactive Bash terminal inside of the Docker container. All binaries should already be compiled. The tests described later assume at least 16GB of RAM installed on the system, with atleast a 4 core CPU along with 40GB or more disk space.

1.2 Kick-the-Tires Test

To make sure that everything is working properly, we will solve the Monty Hall problem as described in Section 2 of the paper. An annotated copy of the code found in Figure 1 can be found in `~/experiments/kick_the_tires/montyhall.cpp`. The goal of this experiment is to prove that the probability of winning the car regardless of the choice of the door, given that the contestant chooses to switch doors when asked, is 2/3.

To run the experiment through PLINKO, simply run the associated Python script found in the `~/experiments` directory:

```
# from terminal inside the docker.
$ cd ~/experiments
$ ./kick_the_tires.py
```

After a few seconds, the script will return a table displaying the name of the experiment, how long KLEE took to analyze the program, how long Z3 took to solve the query, the total amount of time elapsed, the number of paths explored in the program, and the number of random samples (or probabilistic symbolic variables) encountered in the program. The table should look like the following:

Case Study	KLEE (sec.)	Z3 (sec.)	Total (sec.)	Paths	Samples
Montyhall	2.37	0.02	2.39	4	2

Additionally, the output from Plinko will be displayed. In general, if the query is found to be true, Plinko will output "Property verified!". Otherwise, a counterexample and the computed probability of the predicate, ψ, being true will be displayed. In the case of the Monty Hall problem, you should see the following:

```
Num PSVs: 2
Property verified!
Z3 evaluation:   0.02s
```

2 Detailed Evaluation of Plinko

To reproduce the results from the paper we have created Python scripts to automatically run the various experiments performed in the paper. We assume that you run all the commands henceforth from the interactive terminal attached to the docker container that we have shown in section §1.2 of this document.

In the `~/experiments` directory, you will find a script for each table/figure found in Section 6 of the paper, namely Table 1 (`table1.py`), Table 2 (`table2.py`), Figure 6 (`figure6.py`), and Figure 8 (`figure8.py`). To reproduce the Storm comparison experiments (both Freivalds' Algorithm (results found on lines 878–903) and Figure 7), `storm_comparison.py` is used. All source code for the experiments can be found in the correspondingly named directory, also in the `~/experiments` directory.

We will now go through in detail how to use each script and how to interpret the results of them in the order they appear in the paper.

2.1 Table 1

Table 1 provides performance metrics for each of the case studies presented in Section 4 of the paper run on the stock version of Plinko. To reproduce this table in full, simply run the `table1.py` script from within the `~/experiments` directory:

```
# from the ~/experiments directory in docker container.
$ ./table1.py
```

Each case study is run with a timeout of 10 minutes, so at most this script will take 80 minutes, but in actuality should only take about 30 minutes. As the script is running, progress updates will be

printed to the terminal. When the script is completed, an ASCII version of the table will be written to `~/experiments/table1.txt` and should look like the following:

Case Study	KLEE (sec.)	Z3 (sec.)	Total (sec.)	Paths	Samples	Concretizations
Freivalds'	2	17	19	2	2	(n) = (2)
Freivalds' (Multiple)	6	253	259	8	21	(n,k) = (3,7)
Reservoir Sampling	14	81	95	127	6	(n,k) = (13,7)
Reservoir Sampling	509	0	509	4096	12	(n,k) = (13,1)
Monotone Testing	5	377	382	36	1	(n) = (27)
Quicksort	14	109	123	120	10	(n) = (5)
Bloom Filter	12	360	372	83	8	(m,epsilon) = (3,0.39)
Count-min Sketch	3	132	135	2	8	(n,epsilon,gamma) = (4,0.5,0.25)

Additionally, log files produced by Plinko for each case study are written to the `table1_results` directory. Each log file should display the number of PSVs, whether the property was verified, and how long the Z3 evaluation took. All case studies should show that they have been verified with the exception of Bloom Filter, which produces a counter-example due to a bug in the implementation (as stated in the paper). In particular, that log file (`table1_results/bloom_filter.log`) should look like the following:

```
Num PSVs: 8
mergePathsWithProbs: 353.69s
sum_probs!0 -> (/ 44177.0 78125.0)


Z3 evaluation: 360.93s
```

2.2 Figure 6 (Psi Comparison)

Figure 6 presents performance results from our comparison against Psi on Freivalds' algorithm, Monotone Testing, and the Reservoir Sampling case studies. Due to the amount of data presented in this figure, instead of providing a script to run all the settings (which would take many, many hours), we have instead opted to allow the reviewer to select which case study to run and which concretizations to run the case study with. Running `figure6.py -h` will provide a short summary of how to use the script, but we will explain in greater detail below.

2.2.1 Freivalds' Algorithm

To produce the results of Figure 6a which uses Freivalds' Algorithm as a case study, run

```
# from the ~/experiments directory in docker container.
$ ./figure6.py freivalds <k>
```

where `<k>` is the setting of k, or the number of times Freivalds' algorithm should be run to reduce the false-positive probability. The value of n is fixed to be 2. For example, if you wanted to run Freivalds' algorithm with $k = 10$ on Psi, Plinko, and Plinko with the algebraic simplifications optimization turned on, run:

```
$ ./figure6.py freivalds 10
```

Progress statements will be printed as experiments are run. Upon the completion of the script, a table will be printed giving the timing results for each system. The table should look like the following:

```
Freivalds' (FirstOff): (n,k) = (2,10)
-----------------------------------
       System         Time (sec.)
-------------------- -------------
Psi                          20.08
Plinko                       84.77
Plinko (Algebraic)            5.16
```

Additionally, all log files can be found in the `figure6_results` directory.

If you would only like to run an individual system (e.g., you only want to test Psi), you can provide the `--psi` flag to the command:

```
$ ./figure6.py freivalds 10 --psi
```

To just run Plinko, use the `--plinko` flag, and to run Plinko (Algebraic), use the `--plinkoAlg` flag.

2.2.2 Monotone Testing

To produce the results of Figure 6b which uses Monotone Testing as a case study, run

```
$ ./figure6.py monotone-testing <n>
```

where `<n>` is the setting of n, or the cardinality of the domain of the function to be tested for monotonicity. For example, if you wanted to run Monotone Testing with $n = 5$ on both Psi and Plinko, run:

```
$ ./figure6.py monotone-testing 5
```

As with Freivalds' algorithm, progress statements will be printed,a table will be produced when the script has terminated, and log files will be written to the `figure6_results` directory. For the above command, the table should look like the following:

```
Monotone Testing: (n) = (5)
-----------------------------------
       System         Time (sec.)
-------------------- -------------
Psi                         600.00
Plinko                        2.05
```

Again, if you would only like to test Psi, use the `--psi` flag, and if you would only like to test Plinko, use the `--plinko` flag.

2.2.3 Reservoir Sampling

To produce the results of Figure 6c which uses Reservoir Sampling as a case study, run

```
$ ./figure6.py reservoir-sampling -n <n> -k <k>
```

where `<n>` is the setting of n to use and `<k>` is the setting of k to use. For example, if you wanted to run Reservoir Sampling with $n = 9$ and $k = 4$ on both Psi and Plinko, you would run the following command:

```
$ ./figure6.py reservoir-sampling -n 9 -k 4
```

As with the other two case studies, progress statements will be printed, a table will be produced when the script has terminated, and log files will be written to the `figure6_results` directory. For the above command, the table should look like the following:

```
Reservoir Sampling: (n,k) = (9,4)
---------------------------------
       System         Time (sec.)
------------------- -------------
Psi                         85.88
Plinko                       7.95
```

Again, if you would only like to test Psi, use the `--psi` flag, and if you would only like to test Plinko, use the `--plinko` flag.

2.3 Storm Comparison (Lines 863–919)

We additionally compare Plinko against Storm, a probabilistic model checker on Freivalds' algorithm and Reservoir Sampling. Due to the amount of data presented in this figure, instead of providing a script to run all the settings (which would take many, many hours), we have instead opted to allow the reviewer to select which case study to run and which concretizations to run the case study with. Running `storm_comparison.py -h` will provide a short summary of how to use the script, but we will explain in greater detail below.

As an aside, recall from the paper that we had to restrict the input domain of the matrices for Freivalds' algorithm, and the input array for reservoir sampling, to be $\{1, \ldots, N\}$. When we refer to N, we are referring to the maximum element of the input domain that you would like to test for Storm.

2.3.1 Freivalds' Algorithm

To test Freivalds' algorithm against Storm and Plinko, run the following command:

```
# from the ~/experiments directory in docker container.
$ ./storm_comparison.py freivalds <N>
```

where `<N>` is the maximum element that a value in the matrix can be. For this experiment, we restrict ourselves to just 2x2 matrices with Freivalds' algorithm run only once ($n = 2, k = 1$).

For example, if you wanted to restrict the input domain to be $\{1, 2\}$, and to test against Storm and Plinko, you would run the following command:

```
$ ./storm_comparison.py freivalds 2
```

We do note that regardless of the setting of N, Plinko will use full 32-bit `int`s for verification, not $\{1, \ldots, N\}$.

Progress statements will be printed as experiments are run. Upon the completion of the script, a table will be printed giving the timing results for both tools. The table should look like the following:

```
Freivalds' (AllOff): (n,k) = (2,1)
----------------------------------
       System         Time (sec.)
-------------------- -------------
Plinko                        3.63
Storm                         3.00
```

Additionally, all log files can be found in the `storm_comparison_results` directory.

If you would only like to run an individual system (e.g., you only want to test Storm), you can provide the `--storm` flag to the command:

```
$ ./storm_comparison.py freivalds 2 --storm
```

To just run Plinko, use the `--plinko` flag (when using this flag you can elude the setting of N)

2.3.2 Reservoir Sampling (Figure 7)

To test Reservoir Sampling against Storm and Plinko, run the following command:

```
$ ./storm_comparison.py reservoir-sampling -n <n> -N <N>
```

where `<n>` is the length of the input array from which the sample will be created from, and `<N>` is the maximum element that a value in the input array can be. For this experiment, we restrict ourselves to just the case where $k = 1$ (i.e., the produced sample contains but a single element).

For example, if you wanted to restrict the input domain to be $\{1, \ldots, 6\}$, and you wanted to set $n = 5$, you would run the following command:

```
$ ./storm_comparison.py reservoir-sampling -n 5 -N 6
```

As with Freivalds' algorithm, Plinko will always be run with an input domain of 32-bit `int`s regardless of the setting of N.

Upon the completion of the script, a table will be printed giving the timing results for both tools. The table should look like the following:

```
Reservoir Sampling: (n,k) = (5,1)
---------------------------------
       System         Time (sec.)
-------------------- -------------
Plinko                       2.02
Storm                       37.00
```

Additionally, all log files can be found in the `storm_comparison_results` directory.

If you would only like to run an individual system (e.g., you only want to test Storm), you can provide the `--storm` flag, and if you only want to test Plinko you can provide the `--plinko` flag.

2.4 Table 2

Table 2 provides performance metrics of the three variations of specifying that $A \times B \neq C$ in the query for Freivalds' algorithm when $n = 2$ and the elements of the matrices are drawn from the space of 32-bit

ints. To reproduce the results of Table 2, simply run the following command from the ~/experiments directory:

```
# from the ~/experiments directory in docker container.
$ ./table2.py
```

As the script is running, progress updates will be printed to the terminal. When the script is completed, an ASCII version of the table will be written to ~/experiments/table2.txt and should look like the following:

Case Study	KLEE (sec.)	Z3 (sec.)	Total (sec.)	Paths
AllOff	3	0	3	2
SomeOff	2	27	29	2
FirstOff	2	2	4	2

Additionally, log files produced by Plinko for each case study are written to the table2_results directory. Each log file should display the number of PSVs, whether the property was verified, and how long the Z3 evaluation took.

2.5 Figure 8

Figure 8 presents performance metrics comparing the default, unoptimized version of Plinko against Plinko with the formula sharing optimization (denoted as "Sharing") and Plinko with the algebraic simplifications optimization (denoted as "Algebraic") on Freivalds' algorithm, Reservoir Sampling, Monotone Testing, and Quicksort.

Due to the amount of data presented in this figure, instead of providing a script to run all the settings (which would take many, many hours), we have instead opted to allow the reviewer to select which case study to run and which concretizations to run the case study with. Running `figure8.py -h` will provide a short summary of how to use the script, but we will explain in greater detail below.

2.5.1 Freivalds' Algorithm

To produce the results of Figure 8a which uses Freivalds' Algorithm as a case study, run

```
# from the ~/experiments directory in docker container.
$ ./figure8.py freivalds <k>
```

where <k> is the setting of k, or the number of times Freivalds' algorithm should be run to reduce the false-positive probability. The value of n is fixed to be 2. For example, if you wanted to run Freivalds' algorithm with $k = 9$ on Plinko, Plinko (Sharing), and Plinko (Algebraic), run:

```
$ ./figure8.py freivalds 9
```

Progress statements will be printed as experiments are run. Upon the completion of the script, a table will be printed giving the timing results for each system. The table should look like the following:

```
Freivalds' (FirstOff): (n,k) = (2,9)
----------------------------------
       System         Time (sec.)
-------------------- -------------
Plinko                      18.82
Plinko (Algebraic)           4.15
Plinko (Sharing)           100.88
```

Additionally, all log files can be found in the `figure8_results` directory.

If you would only like to run an individual system (e.g., you only want to test Plinko), you can provide the `--plinko` flag to the command:

```
$ ./figure8.py freivalds 9 --plinko
```

To just run Plinko (Sharing), use the `--plinkoSharing` flag, and to run Plinko (Algebraic), use the `--plinkoAlg` flag.

2.5.2 Reservoir Sampling

To produce the results of Figure 8b which uses Reservoir Sampling as a case study, run

```
$ ./figure8.py reservoir-sampling -n <n> -k <k>
```

where <n> is the setting of n to use and <k> is the setting of k to use. For example, if you wanted to run Reservoir Sampling with $n = 10$ and $k = 4$ on all variants of Plinko, you would run the following command:

```
$ ./figure8.py reservoir-sampling -n 10 -k 4
```

As with the other scripts, progress statements will be printed, a table will be produced when the script has terminated, and log files will be written to the `figure8_results` directory. For the above command, the table should look like the following:

```
Reservoir Sampling: (n,k) = (10,4)
----------------------------------
       System         Time (sec.)
-------------------- -------------
Plinko                      14.61
Plinko (Algebraic)          69.62
Plinko (Sharing)            14.11
```

Again, if you would only like to test Plinko, use the `--plinko` flag, if you would only like to test Plinko (Algebraic), use the `--plinkoAlg` flag, and if you would only like to test Plinko (Sharing) use the `--plinkoSharing` flag.

2.5.3 Monotone Testing

To produce the results of Figure 6c which uses Monotone Testing as a case study, run

```
$ ./figure8.py monotone-testing <n>
```

where <n> is the setting of n, or the cardinality of the domain of the function to be tested for monotonicity. For example, if you wanted to run Monotone Testing with $n = 20$ on all variants of Plinko, run:

```
$ ./figure8.py monotone-testing 20
```

As with the other scripts, progress statements will be printed,a table will be produced when the script has terminated, and log files will be written to the `figure8_results` directory. For the above command, the table should look like the following:

```
Monotone Testing: (n) = (20)
---------------------------------
       System         Time (sec.)
-------------------- -------------
Plinko                       18.07
Plinko (Algebraic)           31.75
Plinko (Sharing)             13.16
```

Again, if you would only like to test Plinko, use the `--plinko` flag, if you would only like to test Plinko (Algebraic), use the `--plinkoAlg` flag, and if you would only like to test Plinko (Sharing) use the `--plinkoSharing` flag.

2.5.4 Quicksort

To produce the results of Figure 6d which uses randomized Quicksort as a case study, run

```
$ ./figure8.py quicksort <n>
```

where `<n>` is the setting of n, or the length of the array to be sorted. For example, if you wanted to run Quicksort with $n = 3$ on all variants of Plinko, run:

```
$ ./figure8.py quicksort 3
```

As with the other scripts, progress statements will be printed,a table will be produced when the script has terminated, and log files will be written to the `figure8_results` directory. For the above command, the table should look like the following:

```
Quicksort: (n) = (3)
---------------------------------
       System         Time (sec.)
------------------- -------------
Plinko                       2.07
Plinko (Algebraic)           2.09
Plinko (Sharing)             2.06
```

Again, if you would only like to test Plinko, use the `--plinko` flag, if you would only like to test Plinko (Algebraic), use the `--plinkoAlg` flag, and if you would only like to test Plinko (Sharing) use the `--plinkoSharing` flag.