A Artifact Appendix

A.1 Abstract

AnonymousTool of the submitted paper is GenMC. We consider the paper's artifact to be *the included version* of GenMC, as well as the tools and benchmarks we used in the paper. We stress that the results obtained for the same benchmarks by GenMC in the future might differ, as the tool will evolve.

GenMC is publicly available on Github. (The version of GenMC included in the artifact is not yet released, but it will be shortly.) For any bugs, comments, or feedback regarding GenMC, please send an email to `michalis@mpi-sws.org`.

A.2 Artifact check-list (meta-information)

- **Algorithm:** GenMC.
- **Program:** GenMC and C benchmarks.
- **Run-time environment:** Docker.
- **Output:** Console.
- **Experiments:** Scripts that fully reproduce the paper's results are provided.
- **How much disk space required (approximately)?:** ~3 GB.
- **How much time is needed to prepare workflow (approximately)?:** Everything is already set up.
- **How much time is needed to complete experiments (approximately)?:** <1 hour (see Section A.7).
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** See GenMC's webpage for its license. For all code in the artifact not belonging to GenMC, GPLv2 applies.
- **Data licenses (if publicly available)?:** GPLv2.
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.4722967.

A.3 Description

A.3.1 How delivered.

The artifact (available on Zenodo) consists of a Docker image containing binaries for all the model checking tools used, along with all the benchmarks used in the submitted version of our paper, and GenMC's source code. These should suffice to validate the claims made in the paper.

A.3.2 Hardware dependencies.

None in particular; having at least 4GB RAM is recommended but not strictly required. Depending on your operating system, Docker might impose some extra hardware restrictions (see Docker's webpage).

A.3.3 Software dependencies.

An operating system on which Docker can be installed (see Docker's webpage) and Docker itself.

A.4 Installation

1. Download and install Docker, in case it is not already installed. On a Debian GNU/Linux distribution, Docker can be installed and started with the following commands:

```
[sudo] apt install docker.io
[sudo] systemctl start docker
```

   We have tested the artifact with Docker 18.09.1 under Debian GNU/Linux.

2. Next, import the Docker container containing GenMC:

```
[sudo] docker import genmc.docker genmc
```

3. Finally, start up the image by issuing:

```
[sudo] docker run -it genmc bash
```

A.5 Experiment workflow

The results of the paper are reproduced using bash scripts that run benchmarks which validate our claims.

A.6 Paper claim-list

The most important claims made in the paper regarding the implementation of GenMC are summarized below:

1. **Section 3.1, Tunable range of errors**: GenMC's error-reporting when a system call fails can be refined.
2. **Section 3.1, Flat combining queue:** We found and repaired a bug in a flat-combining queue implementation.

3. **Section 4.1, Supporting LKMM:** GenMC supports LKMM.
4. **Section 5.1, Supporting system calls:** GenMC supports system calls like `read()` and `write()`.
5. **Section 5.2, Supporting barriers:** GenMC natively supports synchronization barriers in an optimized fashion that can yield exponential gains.
6. **Section 6.1, Symmetry Reduction:** GenMC provides a symmetry reduction mechanism.
7. **Section 6.2, Lock-aware partial order reduction:** GenMC supports LAPOR, that can explore exponentially fewer executions in programs with locks.
8. **Section 6.3, Detecting memory errors:** GenMC detects various kinds of memory errors.
9. **Section 6.4, Detecting persistency errors:** GenMC can reason about persistency properties of programs under `ext4`.

Apart from the claims above, other minor claims regarding GenMC's features are made throughout the paper (e.g., memory model support, spinloop handling, etc). In order to keep this appendix small, we refrain from providing detailed information on such minor claims; instead, we refer interested readers to GenMC's manual, which provides an overview of the tool's basic features.

A.7 Evaluation and expected result

In what follows, we assume that the working directory is `~/cav21-benchmarks`. Each subsection below first lists the command(s) that need to be run in order to validate the respective claim, and then provides some comments on the output. We provide a log file with the expected output for each script under `~/cav21-benchmarks/logs`.

A.7.1 Reproducing Claim 1 (< 5m).

```
./syscall.sh
./syscall-stop.sh
```

These commands run GenMC on an erroneous benchmark involving system calls. In the first case, GenMC only prints a warning regarding the failed `link()` operation, while in the second case this is treated as an error.

A.7.2 Reproducing Claim 2 (< 10m).

```
./fcqueue-broken.sh
./fcqueue-fixed.sh
```

The runs GenMC on the broken flat combining queue implementation (and GenMC will report an error), and the second command runs GenMC on the repaired implementation. (The compilation warnings can safely be ignored.)

Note that the error reported by GenMC is different than the one reported in the paper (although the underlying cause is the same). The reason for this discrepancy is that the test case we used for the submitted version was slightly different. We plan to update the paper accordingly.

A.7.3 Reproducing Claim 3 (< 5m).

```
./compare-results.sh
```

Runs the herd simulator and GenMC on a set of litmus tests under LKMM and compares the output of the two tools.

A.7.4 Reproducing Claim 4 (< 1m).

System call support is used in the test of Section A.7.1, and thus we do not provide a different test here.

A.7.5 Reproducing Claim 5 (< 5m).

```
./barrier.sh
./barrier-no-opt.sh
```

These run GenMC on a benchmark similar to the one described in Section 5.2 of the paper. GenMC explores 1 execution with the barrier optimization enabled, and 24 (i.e., 4!) complete executions with the optimization disabled.

A.7.6 Reproducing Claim 6 (< 5m).

```
./fcqueue-fixed-sr.sh
```

Runs GenMC on the flat combining queue benchmark of Section A.7.2 with symmetry reduction enabled. The running time of GenMC depends on the hardware used but should be substantially lower than the one of `fcqueue-fixed.sh`.

A.7.7 Reproducing Claim 7 (< 5m).

```
./locks.sh
./locks-no-opt.sh
```

These run GenMC on a benchmark utilizing coarse-grained locking with LAPOR enabled and disabled, respectively. GenMC explores 1 execution with LAPOR and 24 (i.e., 4!) executions without it.

A.7.8 Reproducing Claim 8 (< 5m).

```
./memory.sh
```

Runs GenMC on a benchmark similar to the one of Section 6.3 in the paper. A memory race is detected.

A.7.9 Reproducing Claim 9 (< 5m).

```
./pers.sh
```

Runs GenMC on a benchmark where an assertion related to a persistency property of the program is violated. An error is reported.

A.8 Experiment customization

A.8.1 Running GenMC.

A generic invocation of GenMC looks like the following:

```
genmc [OPTIONS] -- [CFLAGS] <file>
```

Where `CFLAGS` are options that will be passed directly to the C/C++ compiler, and `OPTIONS` include several options that can be passed to GenMC (`genmc -help` prints a full list). Among these options, the most useful ones are arguably the `-unroll=N` switch, which unrolls a loop `N` times, and the `-wb` and `-mo` options, that enable the WB and the MO variant of GenMC, respectively (default is WB). Lastly, `file` should be a C/C++ file that uses `pthreads` for concurrency.

More information regarding the usage of the tool, as well usage examples, can be found at GenMC's manual.

A.8.2 Available benchmarks.

The benchmarks we used for our paper are located under `~/cav21-benchmarks/benchmarks`. Apart from the benchmarks located in the folder above, many more benchmarks can be found at GenMC's repository, an (updated) local copy of which can be found at `~/genmc-tool`.

In the above repository under `tests` (and the relevant sub-directories), there is a separate folder for each benchmark, that contains the "core" of the test case, as well as the expected results for the test case, some arguments necessary for the test case to run, etc. In order to actually run a test case, one can run the tool with one of the test case variants, which are located in a folder named 'variants', in turn located within the respective test case folder.

For example, to run a simple store buffering test with GenMC, please issue:

```
genmc ~/genmc-tool/tests/correct/litmus/SB/variants/sb0.c
```

A.8.3 Code Layout.

GenMC's source code is located at `~/genmc-tool/src`. Important parts of the code pertinent to the explanation of Section 3 of the paper can be found in the following files:

`GenMCDriver.{hpp,cpp}`: The main verification driver.
`Interpreter.h, Execution.cpp`: A large part of the interpreter infrastructure.
`ExecutionGraph.{hpp,cpp}`: Default structure for an execution graph.
`*Calculator*`: Relation calculation on an execution graph.