A Artifact Appendix

A.1 Abstract

We consider our paper's artifact to be the set of benchmarks we used in the paper, as well as the results we got by running a particular version of Persevere (and its naive counterparts) on the benchmarks set. We do *not* consider the artifact of the paper to be Persevere itself, as it will evolve over time, and the results obtained by running the same benchmarks may differ in the future.

We have made Persevere publicly available on GitHub, as part of the GenMC tool. For any bugs, comments, or feedback regarding Persevere, please do not hesitate to contact us.

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

- **Algorithm:** Persevere.
- **Program:** C benchmarks publicly available at GenMC's repository.
- **Run-time environment:** docker.
- **Output:** Console.
- **Experiments:** Scripts that fully reproduce the paper's results are provided.
- **How much disk space required (approximately)?:** ~7 GB.
- **How much time is needed to prepare workflow (approximately)?:** Everything is already set up.
- **How much time is needed to complete experiments (approximately)?:** <7 hours (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)?:** 10.5281/zenodo.4067992

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 Persevere'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 in 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 v18.09.1+dfsg1-7.1 and v19.03.13+dfsg1-2 under Debian GNU/Linux.

2. Next, import the docker container containing Persevere:

```
[sudo] docker import persevere.docker persevere
```

3. Finaly, start up the image by issuing:

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

A.5 Experiment workflow

The results of the paper are reproduced using some bash scripts which print out some tables corresponding to the ones in our paper.

A.6 Paper claim-list

The major claims made in the paper regarding the implementation of Persevere are summarized in the beginning of Section 5. More specifically, these are the following:

1. **Section 5, Methodology, L.1069-1070**: Persevere explores exponentially fewer executions than P-Total and P-Partial.
2. **Section 5, Methodology, L.1075-1077:** We found bugs in commonly-used editors using both stress testing and Persevere.

Apart from these major claims above, some minor claims regarding the tools' performance are made in Section 5.1, and a claim regarding Persevere's configuration is made in the beginning of Section 5. Since the former claims are easily extracted from Table 1 in Section 5, and the latter is also exercised in the same table, we do not explicitly list them here. Rather, in Section A.7, we show how to reproduce Table 1 directly.

A.7 Evaluation and expected result

For the following sections we assume that the working directory is `~/popl21-benchmarks`.

As in the paper, we have set the timeout limit to 20 minutes. This can be changed by appropriately setting L.41 in `get-table-data.sh`. We use a clock symbol to denote a timeout, and an X symbol to denote a failure of some sort, although failures are not expected for these benchmarks.

A.7.1 Reproducing Table 1 (~ 5h 30m)

To reproduce the results of Table 1, please issue the following command:

```
./get-table1.sh
```

Note that the times produced might slightly differ from the ones reported due to the different LLVM version used to build Persevere, some optimizations in the tool, and the host machine.

Also note that, in the case of `nano-backup-fix`, Persevere explores 9 executions instead of 10. This is a typo in the paper and will be fixed in the revised version.

A.7.2 [Optional] Reproducing the editor bugs (< 1h)

In this section we describe how to reproduce some of the bugs in common text editors using Persevere and, although not strictly part of our artifact, also using stress testing. Note that, since the bugs depend on races happening (as well as on parameters of the host machines), reproducing them using stress testing might be difficult. That said, we include scripts similar to the ones we used for completeness, as they may facilitate further research.

Persevere To reproduce the bug in `nano` using Persevere, please issue the following command:

```
./nano-bug.sh
```

This should find an assertion violation in the respective test case, and print a violating graph along with an error message.

Note: We only provide a test case for `nano` in the artifact. Bugs in the other editors can be reproduces via stress testing. The test case for `nano` is located at `/root/genmc-tool/tests/wrong/fs/nano-backup-proc/` and its subdirectories. The parts of code extracted from nano are in `/root/genmc-tool/tests/wrong/fs/nano-backup-proc/nano.c`, while our model emulating nano's environment is in `/root/genmc-tool/tests/wrong/fs/nano-backup-proc/fake.h`.

[Optional] Stress testing As reproducing the bug using stress testing requires crashing a machine, the docker image contains a QEMU VM that can be used for that purpose. Thus, in order to be able to start the QEMU VM, the docker image needs to be run with extended privileges as follows:

```
[sudo] docker run -it --privileged persevere bash
```

After starting the docker image anew, you will need another terminal instance for the same container. For that, you will need the ID of the first container. The ID can be obtained by observing the session in the first terminal (it is of the form `root@<ID>`), or by issuing `docker ps`.

Having the ID, the following command from a separate terminal will start another session connected to the same container:

```
[sudo] docker exec -it <ID> bash
```

In the first terminal, type the following command to start the VM:

```
./run-qemu.sh
```

In the second terminal, the testing procedure will take place. This procedure involves editing a file in the VM (over SSH), crashing the VM, and inspecting the file upon rebooting the VM again. However, this procedure is semi-automatic and requires some steps to be taken manually as explained below.

After the testing procedure starts (with the command listed below), an editing session will pop up for a file containing one line of '\0's (the total file size is 128MiB). You will need to edit the file by erasing some '\0's, and replacing them with different ones. As the testing script checks whether the size of the file has changed upon a crash, the number of characters erased and inserted needs to match (e.g., replace exactly 2 characters). Note that the editing procedure might be slow due to the file size.

After you have edited the file, you need to exit `nano` by pressing `Ctrl-X`, and then pressing `Y` when prompted to save the changes, using the already provided filename. Please make sure to exit the file at least 32 seconds after the editing session has popped up, so that I/O load necessary to trigger the race has been started by the testing script.

Upon exiting `nano`, the testing script will immediately crash the VM, restart it, and inspect the file's state. If the error was triggered, it will print an appropriate message. Otherwise, it will restart the testing procedure again.

You can start the interactive testing by typing the following command:

```
./editor-interactive.sh
```

When prompted whether to continue connecting, type `yes`.

Upon successful reproduction of the bug, you can inspect the file's size in the first terminal. To log in, use the username `root` (no password is necessary), and inspect the file size by issuing:

```
ls -lh /usr/local/persevere/big-file.txt
```

The file size should be less than 128MiB, which is the initial size.

Note: Many tries might be necessary in order to trigger the bug.

A.8 Experiment customization

A.8.1 Running GenMC and Persevere

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 compiler, and `OPTIONS` include several options that can be passed to GenMC. Among these options, the most useful ones are probably 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 file that uses `pthreads` for concurrency.

To use Persevere, please invoke GenMC with the `-persevere` option. A shell alias `persevere` that boils down to `genmc -persevere` is provided as well. 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 the tables of our paper are located in the directory `~/popl21-benchmarks/benchmarks` (default directory when starting the image). Apart from the benchmarks located in the folder above, many more benchmarks can be found at GenMC's repository, a 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's folder.

For example, to run a simple test case with with Persevere, one can issue:

```
genmc -persevere ~/genmc-tool/tests/correct/fs/pord-app-sfl/variants/ordered-append0.c
```

A.8.3 Code Layout

GenMC's source code is located at `~/genmc-tool/src`. From the source code, some important parts of the code pertinent to Persevere can be found in the following files:

`PersistencyChecker.cpp`: Calculation of the pb relation and the [REC] axiom of Definition 4.2.

`GenMCDriver.hpp`: Functions being declared with a comment starting with "`Pers:`" are relevant to Persevere and Algorithm 3 of our paper.

`Execution.cpp`: Functions of the form `call*FS()` contain the implementation of the system calls, similar to Figure 7 of our paper.