A Artifact Appendix

A.1 Abstract

We consider the paper's artifact to be the included version of WMC, as well as the tools and benchmarks we used in the paper. The results obtained for the same benchmarks by WMC in the future might differ, as the tool might evolve.

WMC's source code is publicly distributed with this artifact. For any bugs, comments, or feedback, please send an email to `evgeniy.moiseenko@jetbrains.com` or `michalis@mpi-sws.org`.

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

- **Algorithm:** WMC.
- **Program:** WMC, GenMC, HMC, HMC_{LBF}, CDSChecker, Nidhugg, rmem and C/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)?:** ~24h (see Section A.7).
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** See GenMC's webpage. 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.6821753.

A.3 Description

A.3.1 How delivered.

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

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 20.10.12 under Debian GNU/Linux.
2. Next, import the Docker container containing WMC:

```
[sudo] docker import wmc.docker wmc
```

3. Finally, start up the image by issuing:

```
[sudo] docker run -it wmc 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 WMC are summarized below:

1. **Section 5.1**: LB races are fairly races, and WMC only imposes a small overhead on GenMC.
2. **Section 5.2:** WMC's performance is on par with that of per-execution model checkers.
3. **Section 5.3:** Even when a lot of LB races are present, WMC usually explores a low number of duplicate executions. If these are memorized, the number of blocked executions does not outgrow the number of consistent executions.
4. **Section 5.4:** It is possible to automatically repair LB races, usually with one iteration of our script.

Apart from these major claims above, some minor claims regarding the tools' performance in particular benchmarks throughout Section 5. Since these claims easily follow from the tablesreproduced as part of claims 1-4, we do not explicitly list them here.

Finally, we do not distribute some benchmarks that we used that contain proprietary code. Section A.7 explicitly mentions this when it is the case.

A.7 Evaluation and expected result

In what follows, we assume that the working directory is `~/wmc-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 `~/wmc-benchmarks/logs`.

Columns starting with `E:`, `T:` show the number of executions explored and the time required, respectively.

As in the paper, we have set the timeout limit to 30 minutes. This can be changed by appropriately setting L.41 in `get-table-data.sh`. A `*` entry denotes a timeout, while a `+` denotes a failure of some sort, though failures are not expected for these benchmarks.

A.7.1 Reproducing Claim 1 (< 1h).

To reproduce the first case study of Section 5.1 issue:

```
./repair-libs.sh
```

Note that since the lock implementations used by Oberhauser et al. contain proprietary code, they are not included in the artifact. Since the only two races we found were in those benchmarks, the tests included in the artifact are expected to be LB-race-free.

To reproduce Table 1 issue:

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

A.7.2 Reproducing Claim 2 (< 14h).

To reproduce Tables 2 and 3 issue:

```
./get-table2.sh
./get-table3.sh
```

Note that there is a discrepancy for `szymanski(3)`, where all tools time out or run out of memory. We will update the paper accordingly.

A.7.3 Reproducing Claim 3 (< 2h).

To reproduce Table 4 issue:

```
./get-table4.sh
```

A.7.4 Reproducing Claim 4 (< 2h).

To run our repair script on all racy benchmarks of A.7.1 issue:

```
./repair-genmc.sh
```

The command above assumes that `get-table1.sh` has already been run and that the container has not been restarted in the meantime. Since we do not include the proprietary benchmarks of Oberhauser et al. this script will run on 28 instead of 30 racy benchmarks.

A.8 Experiment customization

A.8.1 Running GenMC/WMC.

A generic invocation of GenMC/WMC looks like the following:

```
wmc [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 (`wmc--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/WMC, respectively (default is MO). 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 `~/wmc-benchmarks/benchmarks`. 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 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/WMC's source code is located at `~/genmc-tool/src`. Important parts of the code pertinent to the explanation of Section 4 of our paper can be found in the following files:

`GenMCDriver.{hpp,cpp}`: The main verification driver.
`WKMODriver.{hpp,cpp}`: The required changes for WMC.
`Interpreter.h, Execution.cpp`: A large part of the interpreter infrastructure.
`ExecutionGraph.{hpp,cpp}`: Default structure for an execution graph.
`PromiseSet.{hpp,cpp}`: Calculation of a promise set for certification.

A more detailed explanation of GenMC's code layout can be found at this tool paper.