Manual

MRMC Test Suite

Version 1.3

June 17, 2008

Authors:
Ivan S. Zapreev
Christina Jansen


Contents

1. Introduction

MRMC [KKZ05, JKO+07, Zap08] is a command-line tool for model checking discrete-, continuous- time Markov chains, and their reward extensions. It also support model checking of continuous-time Markov decision processes.

In order to keep MRMC bug free and to compare its performance to other model-checking tools (such as PRISM [KNP02], Ymer [You05b] and VESTA [SVA04]) we have developed a fully automated test suite featuring: internal, functional and performance tests.

The internal tests are targeted on testing, e. g., MRMC data structures, such as: sparse matrices, bit sets, sample vectors, and etc. The functional tests are used to assess the user-level behavior of the tool. This includes tests for the command-line interface, model-checking algorithms, and etc. Last but not least, the performance tests allow to evaluate the efficiency of implemented algorithms, such as: probabilistic bisimulation minimization, and "discrete event simulation" based model checking. Here, we consider several efficiency aspects: verification time, memory usage and etc.

The test suite contains well-known case studies: Wireless Group Communication Protocol (WGC) [MNS99, BCG02, MKL04], Simpel Peer-To-Peer Protocol (PTP) [KNP06], Workstation Cluster (WC) [HHK00, BKKT03, YKNP04, KNP02, KNP08b], Cyclic Server Polling System (CSP) [IT90, You05b, You05a, HKMKS00, SVA04, YKNP06, YS06], Randomized Mutual exclusion (RME) [PZ86], Crowds Protocol (CP) [RR98, KNP08a] and Synchronous Leader Election Protocol (SLE) [IR90, LP02, GSB94, FP04].

The test suite is freely distributed and can be obtained from:

http://www.mrmc-tool.org/

Note that, the test suite is intended to be used on a Linux platform only and its performance sub suite is not proven to work correctly under "Windows + Cygwin" or "Mac OS X".

This manual contains the description of the test suite for MRMC v1.3. The provided description is not complete, and is more-or-less a sorted collection of notes and various facts related to the test suite. It should simplify the process of acquaintance with the MRMC testing but the best understanding of the process can be only obtained through reading the test-suite scripts.

The rest of the document is organized as follows. Chapter 2 gives an overview of the test-suite. There, we discuss what and how we test, we also talk about the test-suite structure, its' configuration parameters and common-file types. Chapter 3 explains how the tests can be invoked, stopped and cleaned. In addition, we show how the test-run outputs should be interpreted. Chapters 4 and 5 provide additional information about *internal*, *functional* and *performance* tests. Chapter 6 contains contact information.

2. General details

In this chapter we are going to discuss some details about the test-suite's designation, top-level structure, configuration parameters, and most common file types.

2.1. What is that we are testing?

The MRMC test suite consists of three major parts that also have subdivisions:

1. ***internal*** – unit tests for the MRMC core.
2. ***functional*** – contains the tests for:
    - The command-prompt interface of MRMC.
    - Model-checking algorithms for:
        - Model checking PRCTL properties on DTMCs.
        - Model checking CSL properties on CTMCs:
            - Numerical algorithms.
            - Discrete event simulation algorithms.
        - Model checking PRCTL properties on DMRMs.
        - Model checking CSRL properties on CMRMs.
        - Model checking CSL properties on CTMDPs.
        - Probabilistic bisimulation for DTMCs, CTMCs, DMRMs, and CMRMs.
3. ***performance*** – contains the tests for:
    - `lumping` – Measures the effects of strong bisimulation minimization in model checking of DTMCs, CTMCs, DMRMs, and CMRMs. The latter two with state rewards only.
    - `simulations` – Compares the efficiency of the discrete event simulation engines of MRMC, Ymer, and VESTA, when model checking CTMCs.

Internal tests are simple *C* programs that include MRMC sources and manipulate with the tools data structures and/or algorithms. These programs provide some output that, when compared to the expected output, allow to check whether or not the MRMC interns are working properly.

Functional tests assess that MRMC, when invoked with certain command line options and run on certain input files, command-prompt commands and/or logical formulae, produces the expected output.

Performance tests for `lumping` run MRMC on various case studies and collect time and memory statistics for verifying the Markov chains, and for minimizing plus verifying the lumped Markov chain. The latter is done for both formula-dependent and formula-independent lumping. The time statistics is based on the elapsed-time output of MRMC, whereas memory statistics is collected using the standard `ps` utility. The latter is periodically invoked during the test runs. For more information about experimental settings read Section 4.2 of [Zap08].

Performance tests for `simulations` allow to run MRMC v1.3, Ymer (v3.0) and VESTA (v2.0) on various case studies and collect time, memory, *confidence*, and *samples* statistics for verifying the CTMCs. The *confidence* statistics estimates the % of correct answers produced by the same tool on the same model with the same inputs. The *samples* statistics reflects the average number of states visited when verifying a given property with a given tool on a given model. The time and *samples* statistics are based on the tool outputs with one exception. By default, Ymer does not report on the number of sampled states. Therefore, we extended the tool with the required *print* statement (for more details see Appendix A). The memory statistics, for all tools, is collected the same way as it is done for the `lumping` tests. For more information about matching the tool parameters and other experimental settings read Sections 7.1 and 7.2 of [Zap08].

Note that, `lumping` and `simulations` sub-suites both use PRISM and its models for generating MRMC input files (Markov chains, labeling, rewards).

Extended information about the *performance* test-suite can be found in Chapter 5.

2.2. Top-level test-suite structure

After downloading the `MRMC_test_v1.3.zip` file, unpack it in the MRMC folder. As a result a directory `MRMC_HOME_DIR/MRMC_test_v1.3/` will be created. Further, for brevity, we assume that you rename it into `MRMC_HOME_DIR/test/`. Then the test-suite structure is as follows:

- `./TS_Manual.pdf` – The test-suite manual.
- `./LICENSE` – A copy of the GPL license.
- `./README` – The "read me" file.
- `./RELEASENOTES` – The release notes.
- `./settings.cfg` – The configuration script.
- `./test_all.sh` – The test-suite invocation script.
- `./clean_all.sh` – The test-suite "clean-up" script.
- `./stop.sh` – The test-run termination script.
- `./internal_tests/` – Unit tests of the MRMC core.
- `./functional_tests/` – Functional tests of MRMC.
- `./performance_tests/` – Performance tests of MRMC.

2.3. Configuring tests

The main configuration parameters of the MRMC test-suite can be set in the

`MRMC_HOME_DIR/test/settings.cfg`

configuration script. These parameters are subdivided into two groups:

General settings

- `MRMC_HOME_DIR` - The absolute name of the MRMC distribution directory.
- `MRMC` - The location of the MRMC binary. This setting does not need to be changed if `MRMC_HOME_DIR` is set correctly. Note that, when running MRMC on Windows, the binary name should be set to `mrmc.exe`.
- `VALGRING_HOME` - The absolute path to the `valgrind` executable [ABFH+08]. It is only required if tests are run under the `-valgrind` option. Note that in this case MRMC should be first recompiled with the `-O0 -ggdb -g` options, which are available in `MRMC_HOME_DIR/makefile.def`.
- `VALGRIND_LOG_FILES_DIR` - The absolute name of the folder for storing *log* filed produced by `valgrind`.
- `EXTRA_VALGRIND_PARAM` - Extra options for `valgrind`.

Performance-test settings The performance part of the test suite was developed for Linux platform only. It is not proven to work under Windows or Mac OS X.

- `PRISM` - The absolute path of the PRISM [KNP02] command line executable. This setting is required for generating performance-test models.
- `TMPDIR` - This setting should point to a local directory, which will be used for storing generated models.
- `YMER` - The absolute path of the Ymer [You05b] command line executable[2].
- `VASTA_JAR` - The absolute path of the VESTA [SVA04] jar file[2].
- `NUMBER_OF_PERFORMANCE_REPETITIONS` - The number of times every performance test is going to be repeated. If set to zero, no "elapsed-time" statistics is collected. At the same time the functional testing and the memory-usage statistics are collected only for the `lumping` sub suite.
- `MILLISECONDS` - The time units of the "elapsed-time" plots.
- `KILOBYTES` - The data units of the "memory-usage" plots.
- `CONFUNIT`- The data units of the "confidence" plots[2].
- `PERFORMANCE_TEST_TIMEOUT_SECS` - The timeout (in seconds) for each performance test invocation.

[2]This setting is required only for the `simulation` sub suite.

2.4. Common test-suite files

One of the most common kind of files in the test suite is the `test_list` file. These files contain lists of names which always correspond to the names of the same directory's sub-folders. These names should be interpreted either as test names or test sub-suite names.

Modifying `test_list` files one can easily prevent tests or sub suites from running. In order to do so, just place the "#" symbol on the line with the test (suite) name. For example, consider `./functional_tests/test_list` – the list of *functional*-test sub suits. This file contains the following data:

```
#############
# TEST LIST #
#############

#Tests for Discrete-time Markov Chains
dtmc

#Tests for Continuous-time Markov Chains
ctmc

#Tests for Discrete-time Markov Reward Models
dtmrm

#Tests for Continuous-time Markov Reward Models
ctmrm

#Tests for Continuous-time Markov Decision Processes
ctmdpi
```

The following modification of this file excludes the `dtmc` and `ctmdpi` sub-suites from the test runs:

```
#############
# TEST LIST #
#############

#Tests for Discrete-time Markov Chains
#dtmc

#Tests for Continuous-time Markov Chains
ctmc

#Tests for Discrete-time Markov Reward Models
dtmrm

#Tests for Continuous-time Markov Reward Models
ctmrm

#Tests for Continuous-time Markov Decision Processes
#ctmdpi
```

The test suite contains various permanent files, designated for this or that purpose. The most common file-name extensions of these files are:

- *.*info* – test case description
- *.*input* – MRMC commands
- *.*tra* – Markov Chains (MC)
- *.*lab* – MC labeling
- *.*rew* – MC state rewards
- *.*rewi* – MC impulse rewards
- *.*golden* – expected MRMC output
- *.*zip* – contain *.golden files

During test runs, the test suite produces various temporary files. The most common filename extensions of such files are listed below:

- *.*out* – actual MRMC output
- *.*diff* – `diff` *.*out* *.*golden*
- *.*results* – time statistics
- *.*memstat* – memory statistics

It is important to note that, *.*out* files are generated during every test run. If there is no difference between the *.*out* and the corresponding *.*golden* file then the former one is deleted. If some difference was detected then it is stored in the *.*diff* file and the test is marked as failed (`FAIL`) in the test script output, otherwise it is marked as passed (PASS).

To put it in a nutshell, if a test fails then its directory contains two new files: *.*out* - an actual test output; *.*diff* - a difference between the *.*out* and the *.*golden* file. Note that, Typically before `diff` is applied, the *.*out* and *.*golden* files are preprocessed by a `sed` script that filters out run-dependent data. That is why resulting *.*diff* files contain only the relevant difference between expected and actual outputs.

Fore more details about the test-suite files, consider reading Chapters 4 to 5.

3. Managing tests

In this chapter we briefly introduce the test-suite functionality by explaining how it can be invoked, stopped and cleaned. We also explain how to interpret test-run outputs. For more information about internal, functional and performance tests we refer to Chapters 4 and 5.

3.1. Running

In this section we are going to discuss two things: *(i)* how the MRMC test suite can be invoked; *(ii)* how to interpret test-run outputs. Since the test suite has many purposes, we split our explanations in several parts. First, we discuss how the test suite and its sub suites can be run. Then, we explain how the output of the *internal* and *functional* sub suites has to be interpreted. In the end, we separately talk about the outputs of the `lumping`- and `simulation`- *performance* sub suites.

The only valid way for invoking MRMC testing is to use the script:

```
MRMC_HOME_DIR/test/test_all.sh
```

When run without any parameters, this script produces the following output:

```
>>test_all.sh
Usage: MRMC_HOME_DIR/test/test_all.sh  [options]
Options:
  -all: run all tests
  -internal: run internal tests
  -functional: run functional tests
  -performance: run performance tests
  -valgrind: turn on Valgrind (mrmc has to be
             compiled with '-O0 -ggdb -g' options)
```

From this it becomes clear that all available MRMC tests can be run by using:

```
>>test_all.sh -all
```

whereas for running *functional* and *performance* sub suites we should use:

```
>>test_all.sh -functional -performance
```

A distinctive feature of the test suite is that *functional* and *performance* tests can be run under the `Valgrind` profiling tool [ABFH+08]. This feature is very useful for MRMC developers, because it allows to track memory leaks and misuses. In order to run *functional* tests under `Valgrind` one has to:

1. Erase all MRMC binaries, by running `make clean` in the `MRMC_HOME_DIR` folder.
2. Modify the the `MRMC_HOME_DIR/makefile.def` file:
   a) Comment: `CFLAGS += -O3`
   b) Uncomment: `#CFLAGS += -O0 -ggdb -g`
3. Compile MRMC binaries, by running `make all` in the `MRMC_HOME_DIR` folder.
4. Invoke the *functional* test suite by running:

```
>>test_all.sh -valgrind -functional
```

The profiling-log files (one for each test) will be located in the folder, defined by the

`VALGRIND_LOG_FILES_DIR`

variable in `MRMC_HOME_DIR/test/settings.cfg`.

Note that:

- `Valgrind` can be supplied with various options by means of the script's variable: `EXTRA_VALGRIND_PARAM`.
- Our test scripts are designed for `Valgrind` version $3.3.0$ or higher.

3.1.1. Internal and functional tests

An example output of the *internal*- and *functional*-test run looks as follows:

```
>>test_all.sh -internal -functional
*****************************************************************
* NOTE: Running Internal Tests
*
*****************************************************************
* .:
        sample_01.................................................PASS
        sample_02.................................................PASS
        simulation_utils_01.......................................PASS
                                  ...
        test_lab_reader...........................................PASS
        test_label................................................PASS
        test_omega................................................FAIL
        test_sparse...............................................PASS
*****************************************************************
* NOTE: Running Functional Tests
*
*****************************************************************
* ./dtmc/pctl/syntax:
        pctl_general_input_01.....................................PASS
        pctl_general_input_02.....................................PASS
        pctl_general_input_03.....................................PASS
*****************************************************************
* ./dtmc/pctl/operators/basic:
```

```
        pctl_basic_01.........................................PASS
************************************************************
* ./dtmc/pctl/operators/long_run:
        pctl_steady_state_01..................................FAIL
        pctl_steady_state_02..................................PASS
        pctl_steady_state_03..................................PASS
        pctl_steady_state_04..................................PASS
                                    ...
```

Here, all tests except for `test_omega` (*internal* tests) and `pctl_steady_state_01` (*functional* tests), pass. In order to find out what caused the test failures, one can consider checking the **.diff* and/or **.out* files of the corresponding tests. The location of these files is defined by the test-run output.

We already know (see Section 2.2) that *internal* and *functional* tests are located (respectively) in the `internal_tests` and `functional_tests` sub folders of the directory: `MRMC_HOME_DIR/test/`. The remaining path to the test location can be constructed with the test name and the sequence of sub-suite names leading to the given test. This name sequence is provided in the test-run output right before each sub-suite tests are executed. For the `test_omega` test, the name sequence is indicated by the output: "`* .:`" which means that it is empty. For the `pctl_steady_state_01` test, the sequence is given by the output: "`* ./dtmc/pctl/operators/long_run:`". Therefore, the **.out* and **.diff* files for these two tests are located in the following directories:

```
./internal_tests/test_omega
./functional_tests/dtmc/pctl/operators/long_run/pctl_steady_state_01
```

3.1.2. Lumping-performance tests

Performance tests for `lumping` are designed to compare model-check time and memory consumption when running MRMC (on the same models, with the same input parameters and formulae to verify) in a simple model-checking mode, in "formula-independent" lumping mode, and in "formula-dependent" lumping mode. Note that, for the latter two the model-check time includes time required for lumping.

A typical output of the `lumping` sub suite is given in Figure 3.1. Here, we run performance testing on the well known Randomized Mutual Exclusion (`rme`) case study [PZ86]. This study provides an algorithm guaranteeing that for N processes trying to access a critical section, at any time t there is at most one process in the critical-section phase and every process can eventually enter the critical section. The `rme` test is located in:

`./performance_tests/lumping/dtmc_lumping/rme`

This location can be easily deduced from the lines 6 to 8 of Figure 3.1. Note that, the `rme` test consists of the following test cases:

`mrmc_RANDOMIZED_N04`, ... , `mrmc_RANDOMIZED_N06`.

These correspond to the model parameter N being equal to $4, \ldots, 6$.

Execution of every `lumping` test consists of running each of its test cases and then *generating overall statistics*. Execution of every test case consists of several stages: *generating a model*, *testing MRMC functionality*, *gathering statistics*. Below, we briefly introduce all of these stages using (to a certain extent) the output provided in Figure 3.1.

```
1  >>test_all.sh -performance
2  ****************************************************************
3  * NOTE: Running Performance Tests
4  *
5  Each test will be repeated '10' times.
6  - lumping
7  - dtmc_lumping
8  -- rme:
9      mrmc_RANDOMIZED_N04:
10       Generating the model .........DONE
11         Functional test:
12           rme01:
13             mrmc_RANDOMIZED_N04.pctl.rme01: +................PASS
14             mrmc_RANDOMIZED_N04.pctl.-ilump.rme01: +.........PASS
15             mrmc_RANDOMIZED_N04.pctl.-flump.rme01: +.........PASS
16         Performance test:
17           rme01: 0:+++ 1:+++ 2:+++ 3:+++ 4:+++ 5:+++ 6:+++ 7:+++
18                  8:+++ 9:+++ DONE
19     mrmc_RANDOMIZED_N05:
20       Generating the model .........DONE
21         Functional test:
22           rme01:
23             mrmc_RANDOMIZED_N05.pctl.rme01: +................PASS
24             mrmc_RANDOMIZED_N05.pctl.-ilump.rme01: +.........PASS
25             mrmc_RANDOMIZED_N05.pctl.-flump.rme01: +.........PASS
26         Performance test:
27           rme01: 0:+++ 1:+++ 2:+13
28                                  ...
```

Figure 3.1.: An example run of the `lumping` sub-suite output.

Generating models

First, for each test case, the MRMC model is generated from the PRISM model. In every test-run output this stage is indicated by:

```
Generating the model .........DONE
```

See for example lines 10 and 20 of Figure 3.1. These lines contain model-generation statements for the test cases: `mrmc_RANDOMIZED_N04` and `mrmc_RANDOMIZED_N05`. The PRISM's **.log.out* file, containing data about the model-generation process, is named after the test case, and is located in the test-case folder, e. g., for `mrmc_RANDOMIZED_N03` it is:

```
./rme/mrmc_RANDOMIZED_N03/mrmc_RANDOMIZED_N03.prism.log.out
```

The generated MRMC models are stored in the folder defined by the `TMPDIR` variable of the `./test/settings.cfg` script (see Section 2.3). It is desirable that this folder is located on the hard drive of the machine that runs the tests. Otherwise, test runs can be affected by the network-speed fluctuations.

./rme/option_list	./rme/input_list
pctl pctl -ilump pctl -flump	rme01.input

Table 3.1.: The option_list and input_list files of the rme test.

Testing MRMC functionality

Functional testing is performed on every test-case model with the same input data as for the further performance testing. The reason to do so is that, before performance testing, we want to be sure that MRMC produces correct results.

For every test (such as rme) the list of used MRMC command-line parameters[1] is located in the option_list file and the list of *.*input* files, containing MRMC commands and verification formulae[2], is located in the input_list file.

For the rme test, these files are: ./rme/option_list and ./rme/input_list. The content of the files is given in Table 3.1. For every test case, we run MRMC on all various combinations of command-line options and the inputs given in these files. This is reflected in lines 12–15 and 22–25 of Figure 3.1.

Note that, when running functional part of the lumping performance tests, the output "-" indicates that the test run was terminated due to the time-out[3], whereas "+" indicates that the run is terminated normally. Also, the test output can contain a line similar to the following one:

```
mrmc_RANDOMIZED_N05.pctl.rme01: +...............????
```

This means that the test run is finished but the results are still being analyzed. After a short while, ???? will change into FAIL or PASS. The latter indicate whether the functional test failed or passed.

Gathering statistics

For performance testing, every test case on every combination of inputs is run several times. The number of repetitions is stated in the very beginning of the performance-suite output, see e. g. line 5 in Figure 3.1, and is defined by the

NUMBER_OF_PERFORMANCE_REPETITIONS

variable of the ./test/settings.cfg script. Each of performance runs is indicated in the Performance test: section of the test-case output, e. g. see lines 17 and 27 in Figure 3.1.

For a better usability, our scripts report the time progress (in tenths of a second) of every test run. For instance, line 27 in Figure 3.1 indicates that the current test run has been being executed for about 1.3 seconds. Note that, the reported time is not exact. The time-out script wakes up every 0.1 second in order to check for the possible time-out and to collect the

[1] See Section 5.1 of the MRMC manual.
[2] See Chapters 6 and 7 of the MRMC manual.
[3] See Section 2.3 for more details.

```
1                                ...
2
3       ------------ Collecting statistics and preparing data ------------
4       mrmc_RANDOMIZED_N03:
5         rme01:
6           mrmc_RANDOMIZED_N03.pctl.rme01 ..........DONE
7           mrmc_RANDOMIZED_N03.pctl.-ilump.rme01 ..........DONE
8                                ...
9       Converting the statistics into the gnuplot data files:
10        rme01.memory.dat:
11          Reading data file: rme.param
12          Reading data file: rme01.pctl.mvsz.memory.statistics
13          Reading data file: rme01.pctl.mrss.memory.statistics
14                               ...
15      ---------------------------------------------------
16          Writing gnuplot-data file: rme01.memory.dat
17          Generating:
18             rme01.memory.mvsz.eps
19             rme01.memory.mrss.eps
20             rme01.memory.avsz.eps
21             rme01.memory.arss.eps
22          --==WE ARE DONE==--
23                               ...
```

Figure 3.2.: Producing statistical results for the `rme` test.

memory-usage data. This script also prints the time-progress information. Therefore, the actual time interval between the time-sampling moments is *at least* 0.1 second. One might want to take this into account when setting the value of `PERFORMANCE_TEST_TIMEOUT_SECS`.

Generating overall statistics

For a given test, after all performance-test runs are finished, the statistical data is collected and the results are stored in the from of **.eps* plots and **.dat* (text) files. The latter ones contain statistical data used to produce the corresponding **.eps* plots. Figure 3.2 shows a part of the statistics-generation log for the `rme` test. In this output, lines 18 to 21, one can see that the overall memory statistics is represented by four plots:

- `rme01.memory.mvsz.eps` – maximum used virtual-memory size (MVSZ),
- `rme01.memory.mrss.eps` – maximum used resident-set size (MRSS),
- `rme01.memory.avsz.eps` – average used virtual-memory size (AVSZ),
- `rme01.memory.arss.eps` – average used resident-set size (ARSS).

Another type of plot we produce is the "model-check" time statistics. For the `rme` test it is present in the `rme01.performance.eps` file. Note that, the resulting statistical data is always stored in the test directory, e. g. `./lumping/dtmc_lumping/rme` for the case of the `rme` test.

For more details about the resulting-statistics files consider reading Section 5.1.

3.1.3. Simulation-performance tests

Performance tests for `simulations` are designed to compare simulation-based model-checking algorithms implemented in MRMC, Ymer and VESTA. Here, we collect four types of statistic:

- "model-check time" – the same as for the `lumping` tests.
- "memory-consumption" – the same as for the `lumping` tests.
- "actual confidence levels" – the % of correct answers, produced by the tools when model checking given properties on given models.
- "number of used observations" – the number of states sampled in order to verify various model-checking formulae.

Note that, all the test models and tool parameters were made sure to be equivalent. For more details, read Section 7.1 of [Zap08].

A typical output of the `simulations` sub suite is given in Figure 3.3. Here, we run per-

```
1  >>test_all.sh -performance
2  ****************************************************************
3  * NOTE: Running Performance Tests
4  *
5  Each test will be repeated '3' times.
6  - simulations
7  - ctmc
8  -- cps:
9      CYCLIC_POLLING_N03:
10       Generating the model: .........DONE
11       Simulating the test:
12           cps01: 0:m+y+y+v+ 1:m+y+y+v+ 2:m+y+y+v+ DONE
13           cps02: 0:m+y+y+ 1:m+y-y+ 2:m+y+y+ DONE
14           cps03: 0:m+v+ 1:m+v+ 2:m+v+ DONE
15           cps04: 0:m+ 1:m+ 2:m+ DONE
16     CYCLIC_POLLING_N06:
17       Generating the model: .........DONE
18       Simulating the test:
19           cps01: 0:m+y+y+v+ 1:m+y+y+v+ 2:m+y+y+v+ DONE
20           cps02: 0:m-y+y- 1:m+y+y+ 2:m+y+y- DONE
21           cps03: 0:m+v+ 1:m+v+ 2:m+v+ DONE
22           cps04: 0:m+ 1:m+ 2:m+ DONE
23                        ...
```

Figure 3.3.: An example run of the `simulations` sub-suite output.

formance testing on the well known Cyclic Server Polling System (`cps`) case study [IT90, You05b, You05a, HKMKS00, SVA04, YKNP06, YS06]. The case study describes a polling system consisting of N equivalent stations and a server. Each station has a single-message buffer and the stations are attended by a single server in a cyclic order. The server starts by polling the first station. If this station has a message in its buffer (*busy*), the server starts serving the station. Once the station has been served, or if there was no message in the buffer

(*idle*), the server start polling the next station. After polling all stations, the server returns to polling the first station and thus beginning a new cycle. The polling and service times are exponentially distributed with rates $\gamma = 200$ and $\mu = 1$. The arrival rate of messages at a station is equal for all stations and is exponentially distributed with rate $\lambda = \frac{\mu}{N}$.

The `cps` test is located in:

```
./performance_tests/simulations/ctmc/cps
```

This location, the same way as it was done in Section 3.1.2, can be easily deduced from the output provided in Figure 3.3. Note that, the `cps` test consists of the following test cases:

`CYCLIC_POLLING_N03`, ... , `CYCLIC_POLLING_N18`.

These correspond to the model parameter $N \in \{3,\ 6,\ 9,\ 12,\ 15,\ 16,\ 17,\ 18\}$.

Execution of every `simulations` test consists of running each of its test cases and then *generating overall statistics*. Execution of every test case consists of several stages: *generating a model*, *gathering statistics*. Below, we briefly introduce all of these stages using (to a certain extent) the output provided in Figure 3.3.

Generating models

The model-generation part of the `simulations` tests is the same as for the `lumping` tests. Note that, the MRMC models are generated from the PRISM models. Ymer directly accepts PRISM models and VESTA uses its own input models, that were made sure to be equivalent to the used PRISM models.

Gathering statistics

The parameters influencing the number of repetitions of each test run and its timeout are the same as for the `lumping`-performance tests. One of the main differences from the `lumping` tests is that we do not just run MRMC but also Ymer and VESTA. For the rest, we still allow for having tool runs on different command-line options (per tool) and inputs (per test).

Let us consider the example run in Figure 3.3. It is easy to see, line 5, that every tool run (on a given test case, with selected command-line options and inputs), will be repeated 3 times. Moreover, for every test case, after the model is generated (e. g. line 10), the simulation tests are invoked. These tests are performed in a "per input" (`csp01`, `csp02`, `csp03`, and `csp04`) manner. For example, on line 12 we can see that for the test case `CYCLIC_POLLING_N03` on the input `cps01` we perform three repetitions marked from 0 to 2. In each repetition we consequently run MRMC – denoted by the letter "`m`", Ymer – denoted by the letter "`y`" and VESTA – the letter "`v`". Ymer is run twice because the command-line options for the first and second invocations differ. Unlike for `lumping` tests, the "`+`" output indicates that the tool produced proper model-checking results, otherwise we have "`-`". The latter check is required for collecting the "actual confidence levels" statistics.

Remember that every input, e. g. `cps01`, contains a particular formulae that is to be model checked. In our case, we verify CSL formulae but not all of the considered tools support this logic to the full extent. Thus, it is possible that on a particular input we can only run some of the tools, but not all of them. For example, it is the case with the input `cps03`. For this input we can only run MRMC and VESTA, but not Ymer.

Generating overall statistics

For a given test, after all performance-test runs are finished, the statistical data is collected and the results are stored in the from of **.eps* plots and **.dat* (text) files. The latter ones contain statistical data used to produce the corresponding **.eps* plots. Figure 3.4 shows a part of the statistics-generation log for the `cps` test. In this output, lines 3, 27, 37, and 40 divide the output into four parts and show in which order the statistical data is generated. For every input name `INP` and a set of tools run on this input we produce four plots:

- `INP.memory.mvsz.eps` – the "memory-consumption" statistics (`MVSZ` only), the same as for the `lumping` tests.
- `INP.performance.eps` – the "model-check time" statistics, the same as for the `lumping` tests.
- `INP.confidence.eps` – the "actual confidence levels" statistics.
- `INP.sample.eps` – the "number of used observations" statistics.

The resulting statistical data is always stored in the `statistics` sub folder of the test directory, e. g. `./simulations/ctmc/cps/statistics` for the case of the `cps` test.

For more details about the resulting-statistics files consider reading Section 5.2.

3.2. Stopping

The internal- and/or functional- test runs can be terminated by simply pressing *Ctrl-C* in the console where they were invoked. The performance tests run MRMC in the background. Therefore, in order to halt these tests, it is not enough to terminate the test scripts by pressing *Ctrl-C*. If performance tests are to be stopped, the `MRMC_HOME_DIR/test/stop.sh` script shall be used. Just run it during the performance-test execution from another console. A typical output of this script looks as follows:

```
>> stop.sh
+++++++++++++++++ Stopping tests +++++++++++++++
* Iteration 1: Some unstopped processes detected.
1. Killing the main script, PID: 18525
2. Killing the test scripts, PID: 27071
3. Killing the performance test scripts,
   PID: 27373 27088 27083 27077
4. The MRMC processes is/are not running
5. The YMER processes is/are not running
6. Killing the JAVA processes, PID: 5081
7. Killing the PRISM processes, PID: 5546

* Iteration 2: Everything is stopped.
++++++++++++++++++++ Done ++++++++++++++++++++
```

Note that, this script will terminate all Java applications and/or MRMC, PRISM instances running on the same machine. Yet, we assume that this script is sufficiently safe, since performance testing should be done on a stand-alone machine dedicated specifically for the testing purpose.

```
                              ...

------------ Collecting PERFORMANCE statistics and preparing data ------------
CYCLIC_POLLING_N03:
  cps01:
    CYCLIC_POLLING_N03.cps01.mrmc.common ..........DONE
    CYCLIC_POLLING_N03.cps01.ymer.common ..........DONE
    CYCLIC_POLLING_N03.cps01.ymer.--pestimate ..........DONE
    CYCLIC_POLLING_N03.cps01.vesta.common ..........DONE
  cps02:
                              ...

Converting the statistics into the gnuplot data files:
  statistics/cps01.performance.dat:
    Reading data file: cps.param
    Reading data file: statistics/cps01/cps01.mrmc.common.performance.statistics
    Reading data file: statistics/cps01/cps01.ymer.common.performance.statistics
    Reading data file: statistics/cps01/cps01.ymer.--pestimate.performance.statistics
    Reading data file: statistics/cps01/cps01.vesta.common.performance.statistics
-------------------------------------------------
    Writing gnuplot-data file: statistics/cps01.performance.dat
    Generating:
        statistics/cps01.performance.*.eps
    --==WE ARE DONE==--
                              ...

------------ Collecting MEMORY statistics and preparing data ------------
CYCLIC_POLLING_N03:
  cps01:
    CYCLIC_POLLING_N03.cps01.mrmc.common ..........DONE
    CYCLIC_POLLING_N03.cps01.ymer.common ..........DONE
    CYCLIC_POLLING_N03.cps01.ymer.--pestimate ..........DONE
    CYCLIC_POLLING_N03.cps01.vesta.common ..........DONE
  cps02:
                              ...

------------ Collecting SAMPLE statistics and preparing data ------------
                              ...

------------ Collecting CONFIDENCE statistics and preparing data ------------
                              ...
```

Figure 3.4.: Producing statistical results for the `cps` test.

3.3. Cleaning

Some test runs result in temporary files, such as **.out*, **.diff*, and **.statistics* files, and etc. These files can be automatically erased by executing:

```
MRMC_HOME_DIR/test/test/clean_all.sh
```

When using this script, note that:

- **.eps* and **.dat* files produced by performance tests are not removed, so the resulting data is preserved.
- In order to run performance test without deriving results from the previous runs running `clean_all.sh` is **compulsory**!
- The temporary files are only removed for “enabled” tests, i. e. the test suites and test that are not commented out in the corresponding `test_list` files.

4. Internal and functional tests

In this section we briefly overview the structure of the internal- and functional-test sub suites.

`MRMC_HOME_DIR/test/internal_tests` Stores tests for the MRMC core. These tests are C source files that perform unit testing of some of the MRMC components. The structure of this sub suite is similar to the structure of the functional sub suite.

`MRMC_HOME_DIR/test/functional_tests/` Stores tests for the MRMC interface and the model-checking algorithms. The structure of this sub suite is as follows:

- `./test_list` – the list of tests
- `./test.sh` – runs tests from `test_list`
- `./clean.sh` – removes temporary files
- `./dtmc/` – tests for Discrete Time Markov Chains
- `./ctmc/` – tests for Continuous Time Markov Chains
- `./dtmrm/` – tests for Discrete Time Markov Reward Models
- `./ctmrm/` – tests for Continuous Time Markov Reward Models
- `./ctmdpi/` – tests for Continuous Time Markov Decision Processes

The test suite also contains several supplementary files:

- `./out2golden.sh` – substitutes the **.golden* files with the pre generated **.out* files for the given list of tests. Has to be invoked as: `out2golden.sh test_list`.
- `./sed.rules` – contains `sed` rules for extracting meaningful data from the **.golden* and **.out* files, before applying `diff`.
- `./pf.sh` – performs filtering for **.golden* and **.out* files. Also, invokes `diff` and reports `PASS`/ `FAIL`. This script is called from `test.sh`.

5. Performance tests

At present, the performance test suite of MRMC:

```
MRMC_HOME_DIR/test/performance_tests
```

has the following structure:

- `./test_list` – the list of tests suites
- `./test.sh` – runs test suites from `test_list`
- `./clean.sh` – removes temporary files
- `./scripts/awk/` – scripts (`awk`) for processing statistical data
- `./scripts/shell/` – common scripts used for gathering statistics
- `./scripts/sed/` – scripts (`sed`) required for extracting statistical data
- `./scripts/bin/` – contains the pre-compiled bash shell binary[1]
- `./scripts/gcc/` – supplementary programs needed for test runs
- `./lumping/` – the test suite for the bisimulation (lumping)
- `./simulations/` – the test suite for the simulations-based model checking

Remember that the performance test suite consists of two sub suites, namely: `lumping` – tests for bisimulation minimization [KKZJ07], and `simulations` – tests for the discrete-event simulation engine [Zap08]. Although sharing some common scripts, located in the `./scripts/` directory, these sub suites are quite different. The former one is simpler and therefore we will first discuss its structure, how its performance tests are run, and what statistics is produced. Then, we extend our explanations to the latter sub suite.

5.1. Lumping-performance tests

An approximate structure of the `lumping` sub suite is as follows:

- `./scripts/awk/` – `awk` scripts for computing reduction factors and comparing the probability values with the given error bound

[1] With disabled printing of messages about killed processes.

- `./scripts/sed/` – `sed` scripts which allow to remove unnecessary information from the MRMC output
- `./scripts/shell/` – shell scripts for: running tests, coordinating the statistics generation, and other supplementary scripts
- `./dtmc_lumping/` – the sub suite with tests for DTMCs
- `./ctmc_lumping/` – the sub suite with tests for CTMCs
- `./dtmrm_lumping/` – the sub suite with tests for DMRMs
- `./ctmrm_lumping/` – the sub suite with tests for CMRMs

When the `lumping`-performance tests are run they produce two types of statistics:

- **Model-check time**[2] – based on the "elapsed-time" output of MRMC:
  - `*.performance.statistics` – raw statistical-data files. A name of each file is formed from the input-file name plus the command-line options of MRMC.
  - `*.performance.dat` – post-processed statistical data files which are used with gnuplot scripts to generate performance plots.
  - `*.performance.eps` – the performance plots. These files are generated from the corresponding `*.performance.dat` files.
- **Memory Consumption** – based on the results provided by the `ps` utility:
  - `*.TYPE.memory.statistics` – raw statistical-data files. A name of each file is formed from the input-file name plus the command-line options of MRMC. Here `TYPE` $\in \{mvsz,\ mrss,\ avsz,\ arss\}$.
  - `*.memory.dat` files – post-processed statistical data files which are used with gnuplot scripts to generate memory-consumption plots.
  - `*.TYPE.memory.eps` files – the memory-consumption plots. These files are generated from the corresponding `*.memory.dat` files.

The memory-consumption statistics is based on the output of the standard `ps` utility (Linux) which samples the memory usage of MRMC process approximately every 0.1 second. This sampling is done only during the functional-test part of each performance test.

Note that, the `./lumping/scripts/shell/test_suite.sh` script, used in performance testing, employs a pre-compiled bash interpreter, located in the

`MRMC_HOME_DIR/test/performance_tests/scripts/bin/bash`

directory. The reason for using this binary is that, in case of a test-case timeout, MRMC execution is terminated by invoking the `kill` command. If using a standard shell binary, this procedure results in printing an unwanted text to the console. Since such bash output breaks the structure of the test-script output, we use the modified version of bash.

In cases when it is undesirable or impossible to use the modified shell binary, one has to substitute the first line of `test_suite.sh` in the following manner:

Change "`#!../../../scripts/bin/bash -u`" into "`#!bash -u`".

[2] Only when the value of `NUMBER_OF_PERFORMANCE_REPETITIONS` is > 0, see Section 2.3

5.1.1. Test structure

Let us consider the `lumping`-test structure, using the Workstation Cluster test (`wscl`) as an example. The `wscl` test is located in the `./lumping/ctmrm_lumping/wscl` directory. To prevent this test from being executed one can modify the `test_list` file located in the `ctmrm_lumping` folder. The `wscl` test's directory has the following structure:

- `./mrmc_WORKSTATION_CLUSTER_NXX/` – the test case directory. It contains a test-invocation script and golden files. The test-case (MRMC) model is generated from `wscl.sm` and `wscl.csl`, with the model parameter $N = XX$.
- `./wscl.sm` – the PRISM model of Workstation Cluster.
- `./wscl.csl` – the PRISM property file containing the model labeling.
- `./input_list` – the list of available `*.input` files. Here, we have only one input: `wscl01`. In principle, it is possible to define several input files for the given `wscl` model and to use them for evaluating performance of MRMC on several different model-checking properties.
- `./wscl01.input` – the MRMC input file. Each tests can have several inputs, each of which is a set of MRMC command-prompt commands, that include a model-checking property. The `wscl01.input` file contains the time- and reward-bounded until property and also the `quit` command which is *obligatory* for any `*.input` file.
- `./option_list` – the list of command-line options MRMC should be invoked with: For the same `wscl01.input` file, each of the `option_list` file lines is used to form the MRMC command-line parameters. In case of `wscl`, the file's content indicates that MRMC should be run three times: *first* in the CSRL mode without lumping; *second* with the formula-independent lumping; *third* with the formula-dependent lumping. Note that, any changes done to the `option_list` file must be consistent with the `multiple_list` file.
- `./input.data.files` – the list of MRMC input files. Necessary, because different models (e. g. CTMC vs. CMRM) require different number of MRMC input files.
- `./multiple_list` – the number of "`Total Elapsed * Time * :`" lines in the MRMC output. These numbers are related (line wise) to the options from the `option_list` file. Here, 1 means that if MRMC is run, e. g. with the `csrl` or `csrl -flump` option, there is just one "elapsed-time" output, whereas for 2, e. g. for `csrl -ilump` option, indicates that there are two. In case of the `-ilump` option, the first output corresponds to the lumping time and the second to the model-checking time.
- `./wscl.param` – the values of N with which MRMC models are generated from the PRISM model. This file determines the X-axes values on the generated statistics plots (produced using `gnuplot`). After a test execution the `wscl.param` file values are copied into the first column of the (generated) **.dat* file.
- `./wscl01.performance.gnuplot` – the `gnuplot` template for the "model-check time" statistics. This file contains several "dummy" names, as:

INPUT, TIME_UNIT, MIN, MAX.

These are automatically substituted with the actual values by the statistics script. If any changes are to be done to this file, they must be consistent with the changes in the `option_list` file.

- `./wscl01.memory.gnuplot` – the template file for the memory-consumption statistics. This file is similar to `./wscl01.performance.gnuplot`.

It is important to note that:

- If a test case fails the **.out* and **.diff* files are placed in the corresponding directory.
- The test-case golden files (**.golden*) are stored in the **.zip* archive located in the test-case directory. These files are automatically extracted during the functional part of testing.
- The PRISM output, produced while generating MRMC models, can be found in the **.prism.log.out* file of the corresponding test-case directory.

Further, we discuss a `lumping`-performance test's structure and its statistical outputs in mode detail.

5.1.2. Test statistics

The `lumping`-performance tests generate two types of statistics by means of the

`./lumping/scripts/shell/statistics.sh`

script. Below, we discuss the resulting-statistics files in details. Note that, these files are placed in the root of each test's directory.

Model-Check Time Statistics:

- **.performance.statistics* – Contain average model-check times for the test test-cases. The file name is formed by the **.input* file name plus the command-line options of MRMC from the `option_list` file. For example, in this particular case one may expect the following statistic files:
  - `wscl01.csrl.performance.statistics`
  - `wscl01.csrl.-ilump.performance.statistics`
  - `wscl01.csrl.-flump.performance.statistics`

  Each of these files contains one column of values. Let us discuss how these files are produced. For the "wscl" test we have 7 test cases:

  `mrmc_WORKSTATION_CLUSTER_N01`, ..., `mrmc_WORKSTATION_CLUSTER_N07`

  In each directory `TEST_CASE_NAME` (after the test-case is finished) we have the following files:

  - `TEST_CASE_NAME.csrl.wscl01.results`
  - `TEST_CASE_NAME.csrl.-ilump.wscl01.results`
  - `TEST_CASE_NAME.csrl.-flump.wscl01.results`

  which contain "elapsed-time" information produced by MRMC for the predefined number of test-case repetitions: `NUMBER_OF_PERFORMANCE_REPETITIONS` (see Chapter 2.3). Then, for the test case `mrmc_WORKSTATION_CLUSTER_N01` the average values are computed for each file:

  - `mrmc_WORKSTATION_CLUSTER_N01.csrl.wscl01.results`
  - `mrmc_WORKSTATION_CLUSTER_N01.csrl.-ilump.wscl01.results`
  - `mrmc_WORKSTATION_CLUSTER_N01.csrl.-flump.wscl01.results`

  and are (respectively) placed to be the first row elements of the

  - `wscl01.csrl.performance.statistics`
  - `wscl01.csrl.-ilump.performance.statistics`
  - `wscl01.csrl.-flump.performance.statistics`

  files. Further, the average values for `mrmc_WORKSTATION_CLUSTER_N02` are computed and placed into the second rows, and etc. For more details see:

  `./performance_tests/scripts/awk/average.awk`.

- **.performance.dat* – This is the input file for the **.performance.gnuplot* script. The file is formed by placing the columns from the **.param* and **.performance.statistics* files parallel to each other. Every **.input* file results in its own **.performance.dat* file. For `wscl`, we only have: `wscl01.performance.data`. For more details see:

  `./performance_tests/scripts/awk/arrange_table.awk`.

- **.performance.eps* – Contains the plot for the data from the the corresponding **.data* file. In case of `wscl` test we obtain:

  `wscl01.performance.eps`

The time units, used when generating performance statistics, are defined by the value of the `MILLISECONDS` variable (see Chapter 2.3).

Memory-Consumption Statistics: Before going further, let us note that for the memory statistics we collect the following data, based on the output of the standard `ps` utility [DC03]:

- `VSIZE` (Virtual memory size) – The amount of memory the process is using. This includes the amount in RAM and the amount in swap.

- RSS (Resident Set Size) – The portion of a process that exists in physical memory (RAM). The rest of the program exists in swap. If a computer has not used swap, this number will be equal to VSIZE.

Further we assume that TYPE is one of:

- mvsz – The results for the Maximum measured VSIZE
- mrss – The results for the Maximum measured RSS
- avsz – The results for the Average measured VSIZE
- arss – The results for the Average measured RSS

Below, we describe data files produced during gathering of the memory statistics:

- **.TYPE.memory.statistics* – These files are constructed out of **.memstat* test-case files. In each TEST_CASE_NAME directory (after the functional part of testing is finished) we have the following files:
  - TEST_CASE_NAME.csrl.wscl01.memstat
  - TEST_CASE_NAME.csrl.-ilump.wscl01.memstat
  - TEST_CASE_NAME.csrl.-flump.wscl01.memstat

  Each of these files has three rows of two elements:

  1. MVSZ RSS – the pair of ps results with the max VSZ
  2. VSZ MRSS – the pair of ps results with the max RSS.
  3. AVSZ ARSS – the average over all VSZ RSS pairs.

  For more details on how AVSZ and ARSS are computed, see:

  ```
  ./performance_tests/scripts/awk/on_the_fly_average.awk
  ```

  As a result, for every test case we have the following files:

  - *.csrl.TYPE.memory.statistics
  - *.csrl.-ilump.TYPE.memory.statistics
  - *.csrl.-flump.TYPE.memory.statistics

  They are generated in such a way that each of them has one column of TYPE values. For example, if TYPE = avsz then the AVSZ value from:

  ```
  mrmc_WORKSTATION_CLUSTER_N01.csrl.wscl01.memstat
  ```

  is placed into the first row of wscl01.avsz.memory.statistics.
  The AVSZ value from:

  ```
  mrmc_WORKSTATION_CLUSTER_N02.csrl.wscl01.memstat
  ```

goes into the second row, and etc. For more details see:

```
./performance_tests/scripts/awk/split_memory_statistics.awk
```

- **.memory.dat* file – The input file for the corresponding **.memory.gnuplot* script. This file is formed from the **.param* plus **.TYPE.memory.statistics* files data. For each **.input* its own **.${TYPE}.memory.dat* files are generated. In this example case it is just: `wscl01.memory.data`. For more details see:

```
./performance_tests/scripts/awk/arrange_table.awk
```

- **.TYPE.memory.eps* – Contains the plot for the values from the corresponding **.data* file. In case of `wscl` test we obtain:
  - `wscl01.mvsz.memory.eps`
  - `wscl01.mrss.memory.eps`
  - `wscl01.avsz.memory.eps`
  - `wscl01.arss.memory.eps`

The time units, used when generating memory statistics, are defined by the value of the `KILOBYTES` variable (see Chapter 2.3).

Additional information about the `lumping` sub suite can be found in the comments of the test scripts and other files.

5.2. Simulations-performance tests

An approximate structure of the `simulations` sub suite is as follows:

- `./scripts/sed/` – `sed` scripts that allow to filter tool outputs, for `TOOL` $\in$ { MRMC, Ymer, VESTA }:
  - `TOOL.main.rules` – removes unnecessary data from the output of `TOOL`.
  - `TOOL.result.rules` – removes all (remaining) data except for the model-checking result.
  - `TOOL.sample.rules` – removes all (remaining) data except for the number of used observations.
  - `TOOL.time.rules` – removes all (remaining) data except for the model-check time.
- `./scripts/shell/` – shell scripts for: running tests, checking correctness of the model-checking result, and coordinating the statistics generation.
- `./scripts/shell/invoke_tools` – shell scripts for tool invocations.
- `./scripts/shell/extract_data` – shell scripts for extracting number of samples and model-checking time from the tool outputs.

- `./scripts/shell/generate_statistics` – scripts used for generating all supported types of statistics.
- `./ctmc/` – the sub suite with tests for CTMCs.

When the `simulations`-performance tests are run, for every test and each of its inputs, test-scripts produce four types of statistics:

- **Model-Check Time** – similar to the files produced by the `lumping` sub suite:
  - `*.performance.dat` – post-processed statistical data files which are used with gnuplot scripts to generate "model-check time" plots.
  - `*.performance.eps` – the "model-check time" plots, generated from the corresponding `*.performance.dat` files.
- **Memory Consumption** – similar to the files produced by the `lumping` sub suite:
  - `*.memory.dat` files – post-processed statistical data files which are used with gnuplot scripts to generate memory-consumption plots.
  - `*.memory.mvsz.eps` files – the memory-consumption plots, only for the `MVSZ` statistics (Generated from the corresponding `*.memory.dat` files.).
- **Actual Confidence Levels** – the % of correct answers to the model-checking problem, per tool and per test case. The % value is computed relative to the number of the test-case repetitions.
  - `*.confidence.dat` – post-processed statistical data files which are used with gnuplot scripts to generate actual confidence-level plots.
  - `*.confidence.eps` – the actual confidence-level plots, generated from the corresponding `*.confidence.dat` files.
- **Number of Used Observations** – the average number of observations needed for verifying a given formula on a given test-case model (per tool and its command-line options). The average value is computed relative to the number of the test-case repetitions.
  - `*.sample.dat` – post-processed statistical data files which are used with gnuplot scripts to generate number-of-used-observations plots.
  - `*.sample.eps` – the number-of-used-observations plots, generated from the corresponding `*.sample.dat` files.

Note that, the resulting statistical data is stored in the `statistics` sub folder of each test.

Further, we discuss a `simulations`-performance test's structure and its statistical outputs in mode detail.

5.2.1. Test structure

Let us consider the `simulations`-test structure, using the Cyclic Server Polling System test (`cps`) as an example. The `cps` test is located in the `./simulations/ctmc/cps` directory. To prevent this test from being executed one can modify the `test_list` file located in the `ctmc` folder. The `cps` test's directory has the following structure:

- `./CYCLIC_POLLING_NXX/` – the test case directory. It contains golden files, the PRISM model: **.sm*, and an equivalent VESTA model: **.ctmc*. The test-case model for MRMC is generated from the PRISM model using the **.sh* script. Note that, the value of the model parameter $N = XX$ is hard coded into the PRISM and VESTA models of each test case.
- `./cps.csl` – the PRISM property file containing the model labeling (the same for all test cases).
- `test_list` – the list of enabled test cases, this list is managed the same way as any other `test_list` file. Note that, if a test case is disabled then one has to do corresponding changes in the `cps.param` file.
- `cps.param` – the values of N for each test case. This file determines the X-axes values on the generated statistics plots (produced using `gnuplot`). After a test execution the `cps.param` file values are copied into the first column of the (generated) **.dat* files.
- `./input_list` – the list of available inputs. The inputs here have a much more complex structure. Each input is represented by a folder in the `./inputs` directory.
- `./inputs/cps01/` – contains data related to the `cps01` input:
  - `./tools_list` – the list of tools that are going to be tested with this input,
  - `./*.gnuplot` – the `gnuplot` template scripts for generating plots for the corresponding statistic,
  - `./mrmc/` – the MRMC parameters for the `cps01` input:
    - `./common.options` – the sequence of common command-line options used in every invocation of MRMC with this input.
    - `./files` – the script for providing MRMC with the right input files.
    - `./input` – the sequence of MRMC command-prompt commands and verification properties. This file is similar to the **.input* files of the `lumping`-performance sub suite,
    - `./options` – the list of additional MRMC command-line options. In this file each (not commented and possibly empty) line corresponds to a different set of extra tool options. Remember that, common options have to be placed in the `./common.options` file. If `./options` contains more than one uncommented line, even if it is empty, the tool will be run several times, each time taking a different set of options. This way one can, e. g., run Ymer with and without `--estimate-probabilities` option, and treat these

two invocations as if they are for two different tools. This file is similar to the `option_list` files of the `lumping`-performance sub suite.

- `./vesta/` – the VESTA parameters for the input: `cps01`. Has the same structure as `./mrmc/`.
- `./ymer/` – the Ymer parameters for the input: `cps01`. Has the same structure as `./mrmc/`.

5.2.2. Test statistics

The `simulations`-performance tests generate statistics by means of the following script:

`./simulations/scripts/shell/statistics.sh`

Note that, the initially-gathered statistical data is placed in the `statistics` sub folder of each test case. Such a directory always contains sub folders corresponding to the enabled inputs. In other words, for each test case `CYCLIC_POLLING_NXX` and the input `cpsYY`, where `XX` $\in \{03, 06, 09, 12, 15, 16, 17, 18\}$ and `YY` $\in \{01, 02, 03, 04\}$, the initial statistics is located inside the following folder:

`./CYCLIC_POLLING_NXX/statistics/cpsYY`

Below, we discuss the statistics-generation process and the produced files in details.

Model-Check Time Statistics: The time statistics is collected in a way similar to how it is done for the `lumping`-performance tests. The model-check times are first stored in the **.timestat* files. For example, the file:

`CYCLIC_POLLING_N03.cps01.mrmc.common.timestat`

contains model-check times reported by MRMC for each of

`NUMBER_OF_PERFORMANCE_REPETITIONS`

repetitions, when run on the `cps01` input. Note that, the sub string "`common`" in the file's name indicates that MRMC is run with the common options, given in the file:

`./cps/inputs/cps01/mrmc/common.options`.

Note that, `./cps/inputs/cps01/mrmc/options` only contain one empty line (i. e. no extra options). In contrast, the files:

`CYCLIC_POLLING_N03.cps01.ymer.common.timestat`, and
`CYCLIC_POLLING_N03.cps01.ymer.--estimate-probabilities.timestat`

contain model-check times reported by Ymer, when run with the options defined by the content of `./cps/inputs/cps01/ymer/common.options` and, respectively, the first (empty) and second (non-empty) line of `./cps/inputs/cps01/ymer/options`.

When the test runs for `cps` are finished the time statistics is produced by the next steps:

1. The **.performance.statistics* files are produced and stored in the directory:

`./cps/statistics/cpsYY`

For example, `cps01.mrmc.common.sample.statistics` contains average, for each test case, model-check times reported by MRMC on the `cps01` input. The corresponding files for Ymer are

`cps01.ymer.common.performance.statistics`, and
`cps01.ymer.--estimate-probabilities.performance.statistics`

The first file corresponds to running Ymer with the common options and the second one for running Ymer with the common options with an extra option:

`--estimate-probabilities`,

as defined by the files: `./cps/inputs/cps01/ymer/common.options`, and `./cps/inputs/cps01/ymer/options`.

2. The `./cps/statistics/cpsYY.performance.dat` files are produced. For each input `cpsYY` the file is generated by putting the `cps.param` file date as the first column and then the data from the relevant

   `cpsYY.*.performance.statistics`

   files as the subsequent columns.

3. The `./cps/statistics/cpsYY.performance.eps` files are produced using the corresponding `*.dat` file and the `gnuplot` template script:

   `./cps/inputs/cpsYY/performance.gnuplot`.

Memory-Consumption Statistics: The "memory-consumption" statistics is produced similar to how it is done for the `lumping`-performance tests. One should only take into account that for one input we can have several different tools and each of these tools can be run with a set of different command-line options. In essence, the main steps for generating "memory-consumption" statistics of each kind (i. e. `MVSZ`, `MRSS`, `AVSZ`, `ARSS`) are the same as for the "model-check time" statistics. Note that, as a result we only produce plots for the `MVSZ` statistics, i. e. `./cps/statistics/cpsYY.memory.mvsz.eps`. The latter is generated using the `gnuplot` template script:

`./cps/inputs/cpsYY/memory.gnuplot`.

and the `./cps/statistics/cpsYY.memory.dat` data file.

Actual Confidence-Levels Statistics: The main steps for generating the "actual confidence-levels" statistics are the same as for the "model-check time" statistics.
Note that:

- The reported data can be in *probability* or in %, as defined by the `CONFUNIT` parameter, see Section 2.3.
- The **.confstat* files, stored in the

  `./CYCLIC_POLLING_NXX/statistics/cpsYY`

  folders, contain a column of ones and zeroes. One corresponds to a correct, and zero to a wrong model-checking result.
- The files of the intermediate statistics, located in the `./cps/statistics/cpsYY` folders, have an extension: **.confidence.statistics*.
- As a result we produce `./cps/statistics/cpsYY.confidence.dat` and `./cps/statistics/cpsYY.confidence.eps`. The latter is generated using the `gnuplot` script: `./cps/inputs/cpsYY/confidence.gnuplot`.

Number of Used Observations Statistics: The main steps for generating the "number of used observations" statistics are the same as for the "model-check time" statistics.
Note that:

- The **.samplestat* files, stored in the

  `./CYCLIC_POLLING_NXX/statistics/cpsYY`

  folders, contain the number of used observations as reported by the corresponding tools. An exception is Ymer, which does not report this number. Therefore, the tool was extended in order to provide us with the desired output. For more details on using Ymer with the test suite see Appendix A.
- The files of the intermediate statistics, located in the `./cps/statistics/cpsYY` folders, have an extension: **.sample.statistics*.
- As a result we produce `./cps/statistics/cpsYY.sample.dat` and `./cps/statistics/cpsYY.sample.eps`. The latter is generated using the `gnuplot` script: `./cps/inputs/cpsYY/sample.gnuplot`.

6. Contact

The development of MRMC began in 2004 in the Formal Methods and Tools group (FMT) at the University of Twente (The Netherlands) under the supervision of Prof. Dr. Ir. Joost-Pieter Katoen. Later, the main development of the tool was moved to the Software Modeling and Verification group at the RWTH Aachen (Germany). At present there are several other groups involved into the tool development, namely the Informatics for Technical Applications group at the Radboud University Nijmegen (The Netherlands), the Dependable Systems and Software group at the University of Saarland (Germany), and the Scientific Computing and Control Theory group at the Centrum voor Wiskunde en Informatica (The Netherlands).

If you have any questions, comments or ideas, or if you want to participate in MRMC development, please consider the following contact information:

Name: Prof. Dr. Ir. Joost-Pieter Katoen
Relation: The MRMC team leader, 2004 – present
Affiliation: Software Modeling and Verification, RWTH Aachen, Germany

Name: Dr. Ivan S. Zapreev
Relation: MRMC development, 2004 – present
Affiliation: Scientific Computing and Control Theory, Centrum voor Wiskunde en Informatica, The Netherlands

Name: Dr. Ir. David N. Jansen
Relation: MRMC extension and optimization, 2007 – present
Affiliation: Informatics for Technical Applications, Radboud University Nijmegen, The Netherlands

Name: Prof. Dr. Ing. Holger Hermanns
Relation: CTMDPI model checking, 2007 – present
Affiliation: Dependable Systems and Software, University of Saarland, Germany

More contact information can be found on the MRMC web-page [ZJN+08].

Bibliography

[ABFH+08] Cerion Armour-Brown, Jeremy Fitzhardinge, Tom Hughes, Nicholas Nethercote, Paul Mackerras, Dirk Mueller, Julian Seward, Robert Walsh, and Josef Weidendorfer, *Valgrind*, http://www.valgrind.org/, 2008.

[BCG02] A. Bondavalli, A. Coccoli, and F. Di Giandomenico, *QoS Analysis of Group Communication Protocols in Wireless Environment*, Kluwer Academic Publishers Concurrency in Dependable Computing, 2002.

[BKKT03] P. Buchholz, J.-P. Katoen, P. Kemper, and C. Tepper, *Model-checking large structured Markov chains*, Journal of Logic and Algebraic Programming **56** (2003), 69–96.

[DC03] Jake Dawley-Carr, *HowTo: Profile Memory in a Linux System*, http://mail.nl.linux.org/linux-mm/2003-03/msg00077.html, 2003.

[FP04] W. Fokkink and J. Pang, *Simplifying Itai-Rodeh leader election for anonymous rings*, Electronic Notes in Theoretical Computer Science **128** (2004), no. 6, 53–68.

[GSB94] Rajiv Gupta, Scott A. Smolka, and Shaji Bhaskar, *On randomization in sequential and distributed algorithms*, ACM Computing Surveys **26** (1994), no. 1, 7–86.

[HHK00] B. Haverkort, H. Hermanns, and J.-P. Katoen, *On the Use of Model Checking Techniques for Dependability Evaluation*, Symposium on Reliable Distributed Systems (SRDS), IEEE Computer Society, 2000, pp. 228–237.

[HKMKS00] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle, *A Markov Chain Model Checker*, Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (Susanne Graf and Michael Schwartzbach, eds.), LNCS, vol. 1785, Springer, 2000, pp. 347–362.

[IR90] Alon Itai and Michael Rodeh, *Symmetry breaking in distributed networks*, Information and Computation **88** (1990), no. 1, 60–87.

[IT90] Oliver C. Ibe and Kishor S. Trivedi, *Stochastic Petri Net Models of Polling Systems*, Selected Areas in Communications **8** (1990), no. 9, 1649–1657.

[JKO+07] David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Mariëlle Stoelinga, and Ivan S. Zapreev, *How Fast and Fat Is Your Probabilistic Model Checker?*, Haifa Verification Conference (HVC), LNCS, vol. 4899, Springer, 2007, pp. 65 – 79.

[KKZ05] Joost-Pieter Katoen, Maneesh Khattri, and Ivan S. Zapreev, *A Markov Reward Model Checker*, Quantitative Evaluation of Systems (QEST), IEEE Computer Society, 2005, pp. 243–244.

[KKZJ07] Joost-Pieter Katoen, Tim Kemna, Ivan S. Zapreev, and David N. Jansen, *Bisimulation Minimisation Mostly Speeds Up Probabilistic Model Checking*, Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (Orna Grumberg and Michael Huth, eds.), LNCS, vol. 4424, Springer, 2007, pp. 87–101.

[KNP02] M. Kwiatkowska, G. Norman, and D. Parker, *PRISM: Probabilistic Symbolic Model Checker*, Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS) (T. Field, P. Harrison, J. Bradley, and U. Harder, eds.), LNCS, vol. 2324, Springer, 2002, pp. 200–204.

[KNP06] ———, *Symmetry Reduction for Probabilistic Model Checking*, Computer Aided Verification (CAV) (T. Ball and R. Jones, eds.), LNCS, vol. 4114, Springer, 2006, pp. 234–248.

[KNP08a] ———, *Prism case studies*, http://www.prismmodelchecker.org/casestudies/, 2008.

[KNP08b] ———, *Prism web-page, Workstation Cluster Example*, http://www.prismmodelchecker.org/casestudies/cluster.php, 2008.

[LP02] Richard Lassaigne and Sylvain Peyronnet, *Approximate verification of probabilistic systems*, Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV) (Holger Hermanns and Roberto Segala, eds.), Springer, 2002, pp. 213–214.

[MKL04] Mieke Massink, Joost-Pieter Katoen, and Diego Latella, *Model Checking Dependability Attributes of Wireless Group Communication*, Dependable Systems and Networks (DSN), IEEE Computer Society, 2004, pp. 711–720.

[MNS99] Michael Mock, Edgar Nett, and Stefan Schemmer, *Efficient Reliable Real-Time Group Communication for Wireless Local Area Networks*, European Dependable Computing Conference (Jan Hlavicka, Erik Maehle, and Andrs Pataricza, eds.), LNCS, vol. 1667, Springer, 1999, pp. 380–400.

[PZ86] A. Pnueli and L. Zuck, *Verification of Multiprocess Probabilistic Protocols*, Distributed Computing **1** (1986), no. 1, 53–72.

[RR98] M. K. Reiter and A. D. Rubin, *Crowds: Anonymity for Web Transactions*, ACM Transactions on Information and System Security, vol. 1, ACM Press, 1998, pp. 66–92.

[Som97] Fabio Somenzi, *CUDD: CU decision diagram package*, http://vlsi.colorado.edu/~fabio/CUDD/, 1997, Public software.

[SVA04] Koushik Sen, Mahesh Viswanathan, and Gul Agha, *Statistical Model Checking of Black-Box Probabilistic Systems*, Computer Aided Verification (CAV) (Rajeev Alur and Doron A. Peled, eds.), LNCS, vol. 3114, Springer, 2004, pp. 202–215.

[YKNP04] H. Younes, M. Kwiatkowska, G. Norman, and D. Parker, *Numerical vs. Statistical Probabilistic Model Checking: An Empirical Study*, Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (K. Jensen and A. Podelski, eds.), LNCS, vol. 2988, Springer, 2004, pp. 46–60.

[YKNP06] Håkan Younes, Marta Kwiatkowska, Gethin Norman, and David Parker, *Numerical vs. Statistical Probabilistic Model Checking*, Software Tools for Technology Transfer (STTT) **8** (2006), no. 3, 216–228.

[You05a] H. Younes, *Verification and Planning for Stochastic Processes with Asynchronous Events*, Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[You05b] _____, *Ymer: A Statistical Model Checker*, Computer Aided Verification (CAV) (Kousha Etessami and Sriram K. Rajamani, eds.), LNCS, vol. 3576, Springer, 2005, pp. 429–433.

[YS06] H. Younes and R. Simmons, *Statistical Probabilistic Model Checking with a Focus on Time-Bounded Properties*, Information and Computation **204** (2006), no. 9, 1368–1409.

[Zap08] I. S. Zapreev, *Model Checking Markov Chains: Techniques and Tools*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, 2008.

[ZJN+08] Ivan S. Zapreev, Christina Jansen, Viet Yen Nguyen, David N. Jansen, et al., *MRMC homepage*, http://www.mrmc-tool.org/, 2008.

A. Using Ymer

In order to use Ymer with the performance test suite one has to consider the following steps:

1. According to the Ymer installation instructions, the CUDD [Som97] package has to be installed. Typically, installation of this package is a simple task, but there are two things, one might need to take into account:

   a) At least up until CUDD v2.4.1, the package does not have support for the 64-bit architecture. If one is to compile it on a 64-bit machine then it can be done by appending the "`-m32`" flag to the assignments of `CPPFLAGS`, `ICFLAGS`, and `LDFLAGS` variables in the CUDD makefile (`CUDD_HOME_DIR/Makefile`).

   b) The `configure` script of Ymer requires the `CUDDDIR` parameter. Its value should be the name of the folder containing required library and header files of CUDD. To our knowledge, if CUDD is compiled but is not installed, the required files are located in several different folders. Thus, one has to add the following soft links to the `CUDD_HOME_DIR/include` folder:

      - `libcudd.a -> ../cudd/libcudd.a`
      - `libdddmp.a -> ../dddmp/libdddmp.a`
      - `libepd.a -> ../epd/libepd.a`
      - `libmtr.a -> ../mtr/libmtr.a`
      - `libst.a -> ../st/libst.a`
      - `libutil.a -> ../util/libutil.a`

2. Because Ymer uses CUDD, it does not support 64-bit architecture as well. In order to overcome this problem, simply add the "`-m32`" flag to the assignments of `AM_CPPFLAGS`, `AM_CFLAGS`, and `AM_LDFLAGS` variables in the template makefile (`YMER_HOME_DIR/Makefile.am`) of Ymer.

3. At this point, the Ymer sources can be configured by running:

   ```
   ./configure CUDDDIR=CUDD_HOME_DIR/include
   ```

4. In order to fix some minor source-code problems and to add output, required by the MRMC test suite, `YMER_HOME_DIR/ymer.cc` has to be modified . The modifications that have to be done are indicated by the following listing in Figure A.1:

5. Now, Ymer is ready for the test suite and can be compiled by running its `make` file.

```
>>diff ymer.cc ymer.old.cc
25d24
< #include <math.h>
411c410
< /* if (optopt == '?') {*/
---
> if (optopt == '?') {
414c413
< /* } */
---
> }
716d714
< std::cout << "Sampled states: " << total_path_lengths << std::endl;
```

Figure A.1.: Modifying Ymer