RV Benchmark Challenge 2018
Rules (Version 1.5)

Giles Reger[1] (Chair), Kristin Yvonne Rozier,[2], and Volker Stolz[3]

[1] University of Manchester, UK. giles.reger@manchester.ac.uk
[2] Western Norway University of Applied Sciences, Norway volker.stolz@hvl.no
[3] Iowa State University, USA kyrozier@iastate.edu

Abstract. This document describes the rules and organisation for the Runtime Verification Benchmark Challenge 2018.
This document is related to work from COST Action ARVI IC1402, supported by COST (European Cooperation in Science and Technology).

1 Introduction

This document describes the rules and organisation for the Runtime Verification Benchmark Challenge 2018. See relevant literature for information about Runtime Verification e.g. [3, 4, 8].

Context. During the last decade, many important tools and techniques for Runtime Verification have been developed and successfully employed. However, it was observed that there is a general lack of standard benchmark suites and evaluation methods for comparing different aspects of existing tools and techniques. For this reason, and inspired by the success of similar events in other areas of computer-aided verification (e.g., SV-COMP, SAT, SMT-COMP,CASC), the First Internal Competition on Software for Runtime Verification (CSRV-2014) was established [1, 2]. This was followed by the second [5] and third [10] competitions. In 2015 a COST Action ARVI IC1402 titled *Runtime Verification beyond Monitoring* began and one its main aims was to support and promote such activities. As such, the action has played a pivotal role in progression this work.

In 2017 the competition took a hiatus to reflect on how best to take the original aims of the competition forward. This took the form RV-CuBES: An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools [9, 11]. Alongside this a Dagstuhl Seminar [6] explored similar themes. This benchmark challenge is a direct consequence of these events.

Motivation. During the RV-CuBES workshop it was highlighted that there is a lack of clarity around what a *good* benchmark for Runtime Verification is and that conflating sourcing benchmarks with evaluating tools has meant that the previous competitions have not always helped in this regard. The proposal was a challenge focussed solely on *benchmarks*, not tools and their performance.

Aims. The main aim for this challenge is:

> To collate benchmark suites for runtime verification tools, by sharing case studies and programs that researchers and developers can use in the future to test and to validate their prototypes.

Key to this aim is that sufficient *meta-data* is collected to allow benchmarks to be reusable.

Terminology. We will generally use the following terminology in this document:

- *Benchmark* - a single runnable scenario that can highlight some property of interest about a runtime verification system e.g. a program+property.
- *Benchmark Generator* - a tool that can generate, on demand, benchmarks (e.g. traces) with different characteristics.
- *Benchmark Suite* - a collection of benchmarks and/or benchmark generators from a common topic or scenario e.g. a number of individual benchmarks that exercise the same set of traces with different properties.

This Document. We aim to fully describe all rules and organisation relevant to this year's edition of the competition. The document is structured as follows:

- Section 2 gives an overview of the structure of the competition
- Sections 3 and 4 give details of what technical submissions need to contain for the two categories
- Section 5 talks about how the challenge is going to be evaluated

2 Overview

This challenge is meant to be a *community-led* activity. We want to engage with participants closely and react to their inputs. This is reflected by the use of participants in the *judging* process.

2.1 Timeline (Revised)

Submission is already open and remains open until the end of October. See below for instructions on how to submit. Please note that ideally initial submissions are made before the final deadline, allowing organisers to check submissions meet the requirements and provide feedback to participants if they do not. Submissions must be made by the initial deadline

The initial deadline for first submissions is 31st October 2018

but submissions can be updated until the final deadline

The hard deadline for final versions 5th November 2018

The hard deadline allows certain organisational activities before judging and presentation at RV 2018 (11 - 13 November).

All submissions will have the opportunity to be presented at a special session at RV 2018

2.2 Categories (Revised)

The challenge is formed of two categories. Here we give a brief overview of each category and its motivation. The following sections outline further details of what is expected in each category.

MTL category. In this category a benchmark has a prescribed format of (i) a trace in a given format, (ii) a specification in metric temporal logic (MTL), and (iii) an oracle giving the expected result. The motivation is to fix general and flexible formats to allow benchmarks to be directly comparable e.g. remove any manual translation effort. We have chosen a logic that is relatively simple (compared to some of the very expressive logics used in RV) to enable objective comparison at a level accessible to the majority of tools in the field.

Open category (New). In this category there is no prescription of formats but the general structure follows the above e.g. there should be some data (trace/program) and some property. Ideally there should also be an expected result but benchmarks with unknown results, e.g. potentially containing undetected violations, are accepted. The motivation is to collect together benchmarks that have been produced in a particular context and make them more widely available. The category acknowledges that (i) not all benchmarks will fit into the MTL category, and (ii) not all benchmark producers have the resources required to map their benchmark to the MTL category. However, we do encourage dual submissions i.e. the same benchmark(s) in the open category and mapped to the MTL category.

Generators. In each category there is the option to either provide traces directly or (more commonly in the open category) something that *generates* traces. A generator may produce a fixed set of traces (in which case it is often helpful to provide these as log files as well) or a set of traces with characteristics controllable by some parameters.

2.3 Submission (Revised)

Each submission needs two parts:

- A *benchmark suite* consisting of one or more category-specific benchmark packages (see Section 3 or 4)
- A 2-4 page overview document using the EasyChair style[4]

Details on how to make each submission are below. Both parts are required.

[4] `https://easychair.org/publications/for_authors`

Benchmark suite submission. Submissions of benchmark suites should be made to the following GitHub repository

`https://github.com/runtime-verification/benchmark-challenge-2018`

as a pull-request. If you are unsure how to do this then please contact Giles. **Importantly**, all uploaded data will be released under an *open data* agreement such that all data may be freely available for use by others.

Overview document. Submissions of overview papers should be made to EasyChair

`https://easychair.org/conferences/?conf=rvbc18`

This document should explain (i) the general context, and (ii) each part of the benchmark package e.g. what their components are, how they should be run (in the case of generators), and what examples of good/bad behaviours would be for the given properties.

2.4 Guidelines on what a submission can be

Please note that:

1. The unit of submission is a benchmark suite e.g. this is the unit at which submissions will be evaluated
2. Suites should be based around a distinct domain or scenario *not* a tool
3. Teams can submit multiple suites but the organisers will inspect whether these should be merged
4. Submissions *should not* contain any runtime verification tools; this is a *benchmark challenge*. Although reference to such tools is okay.

3 MTL Category

Entrants to this category are invited to submit benchmarks (individually or as part of a suite) consisting of the following three components.

3.1 Component 1: Trace(s)

Behaviour(s) produced by systems (e.g. traces). These may be provided directly as files or as generators producing such files (in which case some examples of generated files must be included). Traces should be recorded as a set of finite sequences of pairs <time, value> with each sequence representing the behaviour from one source/variable and the pairs consisting of the value of that source and a timestamp. The timestamp should be an integer and timestamps *must be increasing within a sequence*. The value is some alphanumeric sequence that either does not contain whitespace, or is quoted.

Traces can be presented as a single CSV file with the timestamp as the first column and each variable stored in its own column (indicated by a header line) or as a set of CSV files, one for each variable. If all variables are captured in a single file each timestamp should be unique and strictly increasing. We will provide a tool for moving between these two separate file formats.

Example 1. Consider recording the behaviour of an alarm system with two detectors and an alarm. A motion detector variable records the beginning and end of periods of detected motion. A sound alarm variable records decibel levels every 5 seconds. An alarm variable records when the alarm sounds. Below we give a sample set of data as three separate CSV files:

```
time, motion        time, sound         time, alarm
3, start            0, 12               21, true
5, stop             5, 11
12, start           10, 13
21, stop            15, 25
                    20, 36
                    25, 11
```

and as a single CSV file

```
time, motion, sound, alarm
0, , 12,
3, start,
5, stop, 11,
10, , 13,
12, start,
15, , 25,
20, , 36, on
21, stop,
25, , 11,
```

3.2 Component 2: Specification

A description of desired behaviour (specification) in Metric Temporal Logic [7].

Mathematical Syntax. We make use of the standard MTL syntax

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathcal{U}_I\, \varphi$$

where I is an interval of reals with endpoints in $\mathbb{N} \cup \{\infty\}$. The standard definitions exist for $\top$, $\bot$, $\vee$ and $\rightarrow$. We also have $\Diamond\varphi \equiv \top \,\mathcal{U}\, \varphi$ and $\Box \equiv \varphi \,\mathcal{U}_I\, \bot$. If no interval is given then we assume the interval $[0, \infty]$ which gives the same behaviour as their LTL counterparts.

ASCII Syntax. To allow for plaintext definitions we define the following corresponding ASCII representation for mathematical symbols.

$\neg$	`not`
$\wedge$	`and`
$\vee$	`or`
$\rightarrow$	`implies`
$\mathcal{U}_{[i,j]}$	`until`[i,j]
$\Diamond_{[i,j]}$	`eventually`[i,j]
$\Box_{[i,j]}$	`always`[i,j]
∞	`inf`

Propositions. It may be necessary to explicitly relate the propositions from a MTL formula to the signature of the traces. If the values for a variable are `true`/`false` then the variable name can be used directly as a proposition. Otherwise, for each proposition a pair of a *variable* and *predicate* must be given indicating the values for a variable that make the proposition true.

Example 2. Returning to our alarm system let us consider the property that whenever both motion and a sound level above 20 dB is detected the alarm should sound within 2 seconds. This could be captured as

$$\Box(\mathit{motion\_start} \rightarrow ((\mathit{sound} \rightarrow \Diamond_{[0,2]}\mathit{alarm})\ \mathcal{U}\ \mathit{motion\_stop}))$$

or in plaintext

```
always ( motion_start implies
       (( sound implies eventually[0,2] alarm) until motion_stop))
```

where the *alarm* proposition is implicit and other propositions are mapped as

```
motion_start is <motion,=start>
motion_stop is <motion,=stop>
sound is <sound,>20>
```

3.3 Component 3: Oracle

An oracle i.e. the correct answer of whether the system/traces at a given time satisfy the specification. An oracle takes the same form as a behavior: it consists of a finite sequence of pairs <time, verdict> where verdict is a Boolean value stating whether the behavior(s) satisfy the specification at the given time. This allows single verdicts e.g.

```
0,true
```

if the full trace(s) satisfy the property as well as the more general verdict-per-timepoint presentation.

3.4 Summary

A single benchmark package should contain:

1. Either a single CSV trace file for all variables, a set of CSV trace files for each variable, or a program that generates one of those
2. A file named `specification` giving the MTL property in plaintext and any proposition mappings necessary
3. A file named `oracle` giving the verdicts at one or more times

4 Open Category

Whilst we want to be as flexible as possible in this category, it is necessary to balance this with a structure that allows the benchmarks to be easily evaluated and used by others. If submissions in the MLT category are based on a more general setting then we `strongly` encourage the submission of this more general benchmark suite in the open category.

Again we describe the three main components of a benchmark package.

4.1 Component 1: Data

There is a choice to provide the data as trace files, a program that generates trace files directly, or a program that can be instrumented to produce trace files. We describe these choices below.

Option 1: Trace Data In this option one or more trace files should be provided along with a description of their format. If possible the format should conform to an accepted standard. If tools exist to manipulate traces in the given format then it is helpful to provide (links to) these. If the size of files means that they cannot be uploaded to a GitHub repository then they may be published elsewhere and linked to here. Contact the organisers if you need help with hosting such data.

Option 2: Trace Generator In this option a program will be provided which produces trace files in a given format. Instructions should be given on (i) how to install/compile the given program[5], (ii) how to run the program and what parameters it expects to produce trace files, and (iii) the format of the produced trace files (ideally conforming to an accepted standard).

In addition to this program we ask that the submission also include a small number of example trace files that may be produced by the trace generator.

[5] This should be relatively straightforward and not require many complex steps or be too heavily reliant on external packages or platforms.

Option 3: Instrumented Program In this option a program will be provided along with instructions on how to instrument the program to observe events of interest. Ideally an instrumentation solution is also provided e.g. as an AspectJ file or manual instruments. In addition to detailed information about instrumentation, instructions should be given on how to install/compile the given program[6] and (ii) how to run the program.

4.2 Component 2: Specification

A specified behaviour must be given in both plain English and one or more well-defined specification language. Links should be provided to papers/technical reports explaining the specification language(s). This information should be given in plaintext wherever possible. If there is a particular file format for specifications then this may be used but should be explained. An explanation should also be given as to how the propositions/events of the specification relate to the data (even if this might seem obvious).

4.3 Component 3: Oracle

A description of the expected result of monitoring either for the whole trace or at different time points. If this is a more refined-result than a simple true/false then this should be explained. It is okay to have have completely or partially unknown results but this should be clearly stated and the reason for this stated.

4.4 Summary

A single benchmark package should contain:

1. a README file explaining which of the submitted files correspond to the relevant parts
2. Some files representing *data*
3. A plain text file named `specification` providing the specification information described above
4. A plain text file named `oracle` explaining to what extent the data satisfies the specification as described above

5 Challenge Evaluation

In this section we discuss how the challenge will be evaluated.

[6] This should be relatively straightforward and not require many complex steps or be too heavily reliant on external packages or platforms.

5.1 Evaluation Process

A starting point of the evaluation process is that some aspects of what makes a *good* benchmark for runtime verification will be potentially subjective and necessarily qualitative. Therefore, evaluation will be performed by an **evaluation panel** whose selection will be discussed below. We outline the main phases of the evaluation process (these happen in quick succession after initial submission).

Criteria-setting phase. Once the evaluation panel has been formed it will be asked to review the proposed evaluation criteria and discuss possible modifications/additions such that they are happy with the evaluation criteria they will be using.

Bidding phase. Once initial submissions have been received, the evaluation panel will be informed of all submissions. The panel will be asked to bid for submissions that they wish to review, although it should be noted that all members of the panel will be asked to consider each submission in the final phase.

Review phase. The panel will write a brief (a few sentences per criteria) review of how well their allocated submissions meet the agreed evaluation criteria.

Voting phase. The panel will be asked to read the reviews for each submission and score submissions with respect to the agreed evaluation criteria. During this phase the panel may wish to refer back to the original submission.

Awarding phase. Awards will be made in each evaluation criteria and overall in each category. The evaluation panel may also make special mentions where they feel it is appropriate.

5.2 Evaluation Panel

The evaluation panel will consist of a combination of invited independent members of the Runtime Verification community and one nominated person from each of the teams participating in the competition. The size of the evaluation panel will be such that independent persons will be in the majority.

5.3 Evaluation Criteria

A number of evaluation criteria are proposed but these may be revised by the evaluation panel to ensure that they reflect the main criteria of interest:

1. *Challenging Data.* Providing traces that will be challenging to process e.g. are particularly large or complex
2. *Diverse Data.* Providing a diverse set of traces for the same scenario
3. *Complexity of Specification.* A complex and interesting specification

4. *Coverage of Specification.* Providing sufficient data to cover a complex specification (e.g. both positive and negative cases, other measures with good evidence)
5. *Flexibility.* Providing infrastructure to allow many different scenarios to be considered using the same data
6. *Realism.* The benchmark reflects a very real scenario that could be used to convince those in industry of the relevance of Runtime Verification

6 Summary

We have outlined the main motivations, timings, structure, and processes of the challenge and welcome suggestions to improve the challenge. We hope that people enjoy taking part and that the result is of general benefit to the community.

Bibliography

[1] Ezio Bartocci, Borzoo Bonakdarpour, and Yliès Falcone. First international competition on software for runtime verification. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, volume 8734 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2014.

[2] Ezio Bartocci, Borzoo Bonakdarpour, Yliès Falcone, Christian Colombo, Normann Decker, Felix Klaedtke, Klaus Havelund, Yogi Joshi, Reed Milewicz, Giles Reger, Grigore Rosu, Julien Signoles, Daniel Thoma, Eugen Zalinescu, and Yi Zhang. First international competition on runtime verification. *International Journal on Software Tools for Technology Transfer (STTT)*, 2015 (To appear).

[3] Ezio Bartocci, Ylies Falcone, Adrian Francalanza, Martin Leucker, and Giles Reger. An introduction to runtime verification. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *LNCS*, pages 1–23. 2018.

[4] Y. Falcone, K. Havelund, and G. Reger. A tutorial on runtime verification. In Manfred Broy and Doron Peled, editors, *Summer School Marktoberdorf 2012 - Engineering Dependable Software Systems, to appear*. IOS Press, 2013.

[5] Yliès Falcone, Dejan Nickovic, Giles Reger, and Daniel Thoma. Second international competition on runtime verification CRV 2015. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, pages 405–422, 2015.

[6] Klaus Havelund, Martin Leucker, Giles Reger, and Volker Stolz. A Shared Challenge in Behavioural Specification (Dagstuhl Seminar 17462). *Dagstuhl Reports*, 7(11):59–85, 2018.

[7] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, Nov 1990.

[8] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, may/june 2008.

[9] Giles Reger. A report of rv-cubes 2017. In *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA*, pages 1–9, 2017.

[10] Giles Reger, Sylvain Hallé, and Yliès Falcone. Third international competition on runtime verification CRV 2016. In *Runtime Verification - 16th International Conference, RV 2016. Proceedings*, 2016.

[11] Giles Reger and Klaus Havelund, editors. *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and*

Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA, volume 3 of *Kalpa Publications in Computing*. EasyChair, 2017.