Third International Competition on Runtime Verification CRV 2016

Giles Reger, Sylvain Hallé, Yliès Falcone

Third International Competition on Runtime Verification

CRV 2016

Giles Reger[1], Sylvain Hallé[2], and Yliès Falcone[3]

[1] University of Manchester, UK giles.reger@manchester.ac.uk
[2] Université du Québec à Chicoutimi, Canada shalle@acm.org
[3] Univ. Grenoble Alpes, Inria, LIG, F-38000 Grenoble, France
ylies.falcone@imag.fr

Abstract. We report on the Third International Competition on Runtime Verification (CRV-2016). The competition was held as a satellite event of the 16th International Conference on Runtime Verification (RV'16). The competition consisted of two tracks: offline monitoring of traces and online monitoring of Java programs. The intention was to also include a track on online monitoring of C programs but there were too few participants to proceed with this track. This report describes the format of the competition, the participating teams, the submitted benchmarks and the results. We also describe our experiences with transforming trace formats from other tools into the standard format required by the competition and report on feedback gathered from current and past participants and use this to make suggestions for the future of the competition.

1 Introduction

Runtime Verification (RV) [8,13] is a lightweight yet powerful formal specification-based technique for offline analysis (e.g., for testing) as well as runtime monitoring of system. RV is based on extracting information from a running system and checking if the observed behavior satisfies or violates the properties of interest. During the last decade, many important tools and techniques have been developed and successfully employed. However, it has been 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, CASC), the First International Competition on Software for Runtime Verification (CSRV-2014) was established [2]. See [3] for a more in-depth discussion of this first iteration where all submitted properties are presented and the results discussed. Additionally, [11] presents a study discussing how the properties from the competition could be written in two different specification languages. The first iteration of the competition was followed by the second competition the following year which kept the same format but made some minor adjustments based on lessons learnt in the previous year (see [9]).

This is the third edition of the competition and the general aims remain the same:

- To stimulate the development of new efficient and practical runtime verification tools and the maintenance of the already developed ones.
- To produce 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.
- To discuss the metrics employed for comparing the tools.
- To compare different aspects of the tools running with different benchmarks and evaluating them using different criteria.
- To enhance the visibility of presented tools among different communities (verification, software engineering, distributed computing and cyber security) involved in monitoring.

CRV-2016 was held between May and August 2016 with the results presented in September 2016 in Madrid, Spain, as a satellite event of the 16th International Conference on Runtime Verification (RV'16).

Changes. The competition is broadly similar to the previous iteration [9]. The biggest change is that there were not enough participants to run the C track (see Sec. 7). The other changes were designed to make the competition run more smoothly: the number of benchmarks was reduced and an additional stage was introduced to ensure that benchmarks were clarified fully.

Report Structure. We begin by discussing the format of the competition (Sec. 2). We then present and briefly describe the participants to each track (Sec. 3), followed by an overview of the benchmarks submitted in each track (Sec. 4). The results of the competition are then presented (Sec. 5). This is followed by some reflections on the trace format used in the offline track (Sec. 6). Finally, we reflect on the challenges faced and give recommendations to future editions of the competition (Sec. 7) before making some concluding remarks (Sec. 8).

2 Format of the Competition

The format of the competition was broadly similar to that of the previous year (see [9]). The competition website contains a document outlining the full rules of the competition[4], which was distributed to participants before the start of the competition. This section summarises the key points from this document.

2.1 Tracks

As in previous years, the competition was originally meant to consist of three tracks with each track being treated slightly different in each phase. Here we give a brief overview of the general scope of what is covered by the competition and then the idea behind each track.

[4] `http://crv.liflab.ca/CRV2016.pdf`

General Scope. There are many activities that could fall under the umbrella term of runtime verification. Here we describe and defend the *current* scope of the competition. Note that we (the general competition community) are open to suggestions for future iterations.

The general activity we consider is that of taking a trace τ and a specification ψ and answering the question whether τ satisfies/is a model for/is accepted by ψ. In some cases the trace τ is taken as a stand-alone artefact and in other cases it is being generated as a program is running. We restrict our attention to *linear* traces (i.e. we do not consider concurrency) and require programs that generate such traces to be (broadly) deterministic.

Note that our formulation precludes the other, related, activity of finding multiple *matches* between the trace and specification describing failure. In all cases it is sufficient to report failure as soon as it is detected. On a similar note, we do not restrict ourselves to safety properties, but (for obvious reasons) require all specification languages to have an interpretation on finite traces (i.e. one could have bounded liveness).

The Offline Track. This covers the scenario where the trace is collected, stored in a log file, and then processed *offline*. We define three acceptable formats for traces (log files) to be used in benchmarks. In previous years benchmarks in this track have focussed on parametric, or data-carrying, events. Note that this track does not (currently) support notions of time other than as data.

The Online Java Track. This covers a scenario where a Java program is instrumented to produce events that should be handled by a monitor. In the past the majority of instrumentation was carried out via AspectJ. We would like to standardise this where possible. Therefore, benchmark submissions will be required to include AspectJ instrumentation (again, where possible). Entrants may use alternative instrumentation techniques in their submissions but we ask that they justify this.

The Online C Track. This covers a scenario where a C program is run and it is asked whether a specification of that run holds. Starting this year, the C track will consist of two sub-tracks, although this is mainly for organisational reasons and we encourage entrants to participate in both sub-tracks. These are:

1. **Generic Specification.** The C version of the Java track where some instrumentation should abstract the program as a sequence of events to be passed to a monitor. Instrumentation can be automatic or manual.
2. **Implicit Specification.** This covers implicit properties (such as memory-safety and out of bounds array access). Such properties might typically be taken from a standardisation of C rather than formulated in a separate specification language. In this case the trace may also be implicit (although we note that it theoretically exists).

We note that this track did not run due to lack of participants.

2.2 Phases

The competition was divided into five phases as follows:

1. **Registration** collected information about entrants.
2. **Benchmark Phase** In this phase, entrants submitted benchmarks to be considered for inclusion in the competition.
3. **Clarification Phase** The benchmarks resulting from the previous phase were made available to entrants. This phase gave entrants an opportunity to seek clarifications from the authors of each benchmark. Only benchmarks that had all clarifications dealt with by the end of this phase were eligible for the next phase.
4. **Monitor Phase** In this phase entrants were asked to produce monitors for the eligible benchmarks. As described later, these had to be runnable via a script on a Linux system (therefore the tool had to be installable on such a system).
5. **Evaluation Phase** Submissions from the previous phase were collected and executed, with relevant data collected to compute scores as described later. Entrants were given an opportunity to test their submissions on the evaluation system. The output produced during evaluation will be made available after the competition.

Note that it was not necessary to participate in the Benchmark Phase, although not doing so would likely be disadvantageous. However, all entrants were required to take part in the remaining three phases, including the Clarification Phase.

2.3 Timeline

The competition was announced in relevant mailing lists in May 2016. This was much later than in previous years. This could have had an impact on the number of participants. Previous participants and tool developers known to the organisers were contacted directly. Potential participants were requested to declare their intent to participate in the competition using an online form collecting basic information about the participating tools.

The planned timeline was as follows:

Event	Starts	Ends (Deadline)
Registration	May 1st	June 5th
Benchmark Submission	May 1st	May 29th
Clarifications	June 5th	June 12th
Monitor Submission	June 19th	July 10th
Results		August 1st

Extensions were given for each deadline with the final submission deadline being the 22nd July.

2.4 Benchmark Submission Format

Benchmark submissions consisted of three parts:

1. **The Metadata.** Every benchmark requires a name, a description and a domain category.
2. **The Property.** This is a description of the property being monitored and should take the same form for all tracks (with the exception of the **Implicit Specification C subtrack** as described in the full rules document).
3. **The Trace Part.** This describes what the events to be monitored are and is necessarily track-specific. More details are given below.

We now review the last two parts below. The textual information about properties was uploaded to the competition wiki[5] and supporting files were uploaded to the competition server. Each team could submit up to three benchmarks. This is a reduction on previous years to reduce the workload for participants; we discuss the impact of this later.

Describing Properties. The information about a submitted property was formatted as follows:

1. An *informal* description. This should include the context of the property, the relevant events (their names and parameters, if any), and the ordering constraints between events that form the property. Moreover, any assumptions being made should have been reported.
2. Demonstration traces. At least 6 examples traces (3 that should be accepted, 3 rejected) should be provided. Traces can be given in an abstract form e.g. a(1).b(2) and should be explained in terms of the abstract property, not the formal description. The provided traces should ideally highlight edge cases.
3. A *formal* description. This should include resonable detail describing the specification written in a well-defined and documented specification language.

Optionally we encouraged participants to also describe the property in a standard form of first-order linear temporal logic but few participants did this (see later discussions).

Describing Traces. The trace formats fixed in the last iteration of the competition [9] have been kept. Traces could be in standardised CSV, XML or JSON formats. However, in XML and JSON no nesting of data values is supported. Along with the trace files, a benchmark should also include (i) an explanation of how concrete events in the trace relate to abstract events in the property, and (ii) additional statistics about the number of events in the trace.

[5] `http://crv.liflab.ca/wiki`

Describing Programs. For programs, it was required that a benchmark includes the uninstrumented source files, two scripts `compile.sh` and `run.sh` to compile and run the program, and instrumentation information. For the Java track, we preferred instrumentation in the form of an AspectJ file. If it was not obvious, the relation between instrumentation and property should have been explained. Additionally, participants were encouraged to provide the facility to produce a trace file (in the above formats) from the program.

2.5 Monitor Submission Format

Once teams had written monitors for the benchmarks they wished to participate on they could upload these to the server and test that they worked in the competition environment (after installing their tool and all necessary libraries on the server).

Tools were required to give standardised outputs in the form of a status line. Monitors should output the verdict by printing a status line of the following form:

- **STATUS: Satisfied** if the property is satisfied,
- **STATUS: Violated** if the property is violated,
- **STATUS: TimeOut** if the status is not detected within the time limit,
- **STATUS: GaveUp** if the monitor fails to find the verdict for any reason.

If no status line is printed, it was assumed that the status is TimeOut.

For online tracks participants needed to provide a `setup.sh` script to prepare the benchmark, typically this performs automated instrumentation, and a `run.sh` script to run the benchmark, typically this will be the same as in the original submission (perhaps with additional inclusion of some libraries). For the offline track, a single script was required that took two inputs: (i) the name of the benchmark and (ii) the name of the trace file.

2.6 Scoring

The scoring remains the same as for the previous two iterations of the competition (see [2]). Each submission is awarded three scores for *correctness*, *running time* and *memory utilisation*. The correctness score is negative if there is an error e.g. an incorrect verdict. The scores for running time and memory utilisation are computed by distributing a fixed number of points per benchmark between the competing tools in proportion to their performance. For example, if tool A runs in 10 seconds and tool B runs in 40 seconds and there are 10 points to be awarded team A would get 8 points and team B would get 2 points for that benchmark.

3 Participating Teams

In this section, for each track, we report on the teams and tools that participated in CRV-2016. Tables 1 and 2 give a summary of the teams participating in the

Tool	Ref.	Contact person	Affiliation
LARVA	[5]	Shaun Azzopardi	University of Malta, Malta
MARQ	[16]	Giles Reger	University of Manchester, UK
MUFIN	[6]	Torben Scheffel	University of Lübeck, Germany

Table 1. Tools participating in online monitoring of Java programs track.

Tool	Ref.	Contact person	Affiliation
BEEPBEEP 3	[**?**]	Sylvain Hallé	Université du Québec à Chicoutimi, Canada
CRL	[14]	Ariane Piel	ONERA, France
MARQ	[16]	Giles Reger	University of Manchester, UK

Table 2. Tools participating in the offline monitoring track.

Java and Offline tracks respectively. In the following of this section, we provide a short overview of the tools involved in the competition. We note that the E-ACSL tool [7] from CEA LIST, France entered the C track but was the only tool to do so.

CRL. In the framework of Complex Event Processing, CRL [14] is a C++ library which allows for the analysis of complex event flows to recognise predetermined searched-for behaviours. These behaviours are defined as specific arrangements of events using a behaviour description language called the Chronicle Language. The recognition process has been completely formalised through a set semantics and the algorithms of CRL directly correspond to the mathematical definitions. CRL is available online[6].

BEEPBEEP 3 is a general purpose event stream processor that attempts to reconcile the capabilities of Runtime Verification and Complex Event Processing under a common framework [**?**]. In addition to Boolean properties used in monitoring, BEEPBEEP can compute queries that involve complex manipulations of event data and produce output traces of any type. BEEPBEEP 3 is under active development, and is available online[7].

LARVA is a Java tool [5] aimed specifically for monitoring Java systems with a specification language targeting business level logic rather than low level properties. The tool takes a specification in the form of a text file, generating the necessary code in Java and AspectJ which verifies that the properties in the script are being adhered to during the execution of the system. Its specification language (DATEs [4]) is a flavour of automata enriched with stopwatches. LARVA is available online[8].

[6] `http://chronicleрecognitionlibrary.github.io/crl/o.html`
[7] `https://liflab.github.io/beepbeep-3`
[8] `http://www.cs.um.edu.mt/svrg/Tools/LARVA/`

MarQ (Monitoring at runtime with QEA) [16] monitors specifications written as Quantified Event Automata [1,15] (QEA). QEA is based on the notion of trace-slicing, extended with existential quantification and free variables. For online monitoring it relies on AspectJ. For offline monitoring of traces it provides a library of *translator* objects that allow the user to define the interface between the alphabets of the specification and trace. MarQ is available online[9].

Mufin (Monitoring with Union-Find) [6] is a framework for monitoring Java programs. (Finite or infinite) monitors are defined using a simple API that allows to manage multiple instances of monitors. Internally Mufin uses hash-tables and union-find-structures as well as additional fields injected into application classes to lookup these monitor instances efficiently. The main aim of Mufin is to monitor properties involving large numbers of objects efficiently. Mufin will hopefully be available online soon[10].

4 Benchmarks

We give a brief overview of the benchmarks submitted to each track.

4.1 Offline Track

There were 6 benchmarks submitted to the Offline track by 2 teams - MarQ and BeepBeep 3. An additional benchmark was submitted by CRL but this team withdrew. The three benchmarks from MarQ (taken from [1,15]) were

1. *AuctionBidding.* Items placed for auction should only be listed for the prescribed period, all bids should be strictly increasing and should be sold for no less than the reserve price.
2. *CandidateSelection.* For every voter there must exist a party that the voter is a member of, and the voter must rank all candidates for that party
3. *SQLInjection.* Every string derived from an input string must be sanitised before being used.

All three benchmarks appeared in last year's competition. The three benchmarks were designed to demonstrate the different ways data can be used within the specification language.

The three benchmarks from BeepBeep 3 where taken from a case study on applying runtime verfication to bug finding in video games [18]. The properties are therefore all about the interaction of *Pingu* characters within the game:

1. *PinguCreation.* From one event to the next, Pingus can only disappear from the game field; no Pingu can be created mid-game.
2. *EndlessBashing.* Every Basher must become a Walker when it stops bashing.
3. *TurnAround.* A Walker encountering a Blocker must turn around and keep on walking.

[9] `https://github.com/selig/qea`
[10] `http://www.isp.uni-luebeck.de/mufin`

Traces for the benchmarks were generated by Pingu Generator[11]. These traces do not immediately conform to the competition format and we describe how they were translated in Section 6. As this was a new tool to the competition all benchmarks were new.

4.2 Java Track

There were 9 benchmarks submitted to the JAVA track by 3 teams - LARVA, MARQ, and MUFIN. All three teams used ASPECTJ as an instrumentation tool, allowing for easy reuse of instrumentation code.

The three benchmarks from LARVA were

1. *GreyListing.* Once greylisted, a user must perform at least three incoming transfers before being whitelisted.
2. *ReconcileAccounts.* The administrator must reconcile accounts after every 1000 attempted external money transfers or after an aggregate total of one million dollars in attempted external transfers.
3. *Logging.* Logging can only be made to an active session (i.e. between a login and a logout).

The first two benchmarks appeared in the first iteration of the competition (although the monitored programs have been extended to present a more challenging workload at the request of the competition organisers).

The three benchmarks from MARQ (taken from [1,15]) were

1. *PublisherSubscriber.* For every publisher, there exists a subscriber that acknowledges every message sent by that publisher.
2. *AnnonyingFriend.* Person A should not contact person B on at least three different social networking sites without any response from person B. There should not be 10 or more such messages across any number of sites.
3. *ResourceLifecycle.* Managed resources must obey their lifecycle e.g. not granted without first being requested nor released without first being granted.

The third benchmark appeared in last year's competition; the first two are new. The first two benchmarks were designed to demonstrate complex quantifier usage as they alternate universal and existential quantification.

The three benchmarks from MUFIN (also described in [6]) were

1. *Tree.* There is a tree of communicating nodes. The property is about communication between the nodes. For example, whenever a node receives a `sendCritical` message all descendent nodes must have received a `reset` message since the last `send` message.
2. *Multiplexer.* Clients attached to an inactive channel should not be used.
3. *Toggle.* A work piece may only be processed when it is not the same mode as its creating device.

[11] `https://bitbucket.org/sylvainhalle/pingu-generator`

Table 3. Detailed Results for Offline Track

Benchmark	Tool	Time (seconds)	Memory (MB)	Scores		
				Correctness	Time	Memory
AuctionBidding	BeepBeep 3	36,731.04	1,792	10	0.035	5.66
	MarQ	132.01	2,337	10	9.96	4.34
CandidateSelection	BeepBeep 3	6,362.8	1,320	10	10	10
	MarQ	-	OM	-5	0	0
SQLInjection	BeepBeep 3	87.62	1,991	10	1.70	3.83
	MarQ	18.03	1,235	10	8.29	6.17
PinguCreation	BeepBeep 3	16.94	1,146	10	0.70	0.59
	MarQ	1.29	72	10	9.29	9.41
EndlessBashing	BeepBeep 3	116.95	1,473	10	0.26	1.03
	MarQ	3.08	168	10	9.74	8.97
TurnAround	BeepBeep 3	44.08	1,501	10	1.71	2.40
	MarQ	9.1	475	10	8.29	7.60

All three benchmarks appeared in last year's competition. Each benchmark was designed to stress a certain element of the algorithm. *Tree* presents a scenario where the number of objects is not known in advance with complex relationships between objects. *Multiplexer* presents a scenario with many control states. *Toggle* includes global actions affecting all data values.

5 Results

For the first time the competition has been completed in time for the results to be included in the Runtime Verification conference proceedings rather than being announced for the first time at the conference (or in some cases shortly after). In this section, we report on the results and give some brief analysis.

5.1 Detailed Results

Tables 3 and 4 give the detailed results from the Offline and Java tracks respectively. The tables detail the running times and memory utilisation for each submission. The scores for each submission are then given. Negative correctness scores can be given for an incorrect result or error (which does not happen here) or for failing to give a result within the given resources (here we use TO to indicate time out, in this case 10 hours, and OM to indicate out of memory).

From Table 3 we see that MarQ failed to find a solution for its own *CandidateSelection* benchmark. On inspection it was found that MarQ required more than the 8GB of memory available on the competition machine. MarQ performed better than BeepBeep 3 in terms of running time in all cases. This is not very surprising given the low-level specification language used by MarQ. We note that the trace files being used for the two tools for the last three benchmarks were not the same as MarQ first translated the trace files into the competition-compliant CSV format. This translation time is not included in the results.

Table 4. Detailed Results for Java Track

Benchmark	Tool	Time (seconds)	Memory (MB)	Scores		
				Correctness	Time	Memory
GreyListing	LARVA	562.1	140	10	0.15	1.89
	MARQ	15.43	72	10	5.37	3.67
	MUFIN	18.48	59	10	4.48	4.45
ReoncileAccounts	LARVA	7.06	90	10	2.7	2.45
	MARQ	4.8	73	10	3.97	2.99
	MUFIN	5.73	48	10	3.32	4.56
Logging	LARVA	7691.68	181	10	0.07	2.49
	MARQ	104.62	129	10	5.68	3.49
	MUFIN	140.23	112	10	4.24	4.01
PublisherSubscriber	LARVA	0.44	46	10	6.22	4.62
	MARQ	4.86	335	10	0.56	0.63
	MUFIN	0.85	45	10	3.22	4.73
AnnoyingFriend	LARVA	51.63	836	10	1.73	2.04
	MARQ	26.35	718	10	3.40	2.37
	MUFIN	18.38	304	10	4.87	5.59
ResourceLifecycle	LARVA	TO	-	-5	0	0
	MARQ	282.87	752	10	0.85	2.69
	MUFIN	26.35	276	10	9.15	7.31
Tree	LARVA	TO	-	-5	0	0
	MARQ	-	-	0	0	0
	MUFIN	32.34	775	10	10	10
Multiplexer	LARVA	TO	-	-5	0	0
	MARQ	105.54	1703	10	0.38	1.06
	MUFIN	4.23	201	10	9.61	8.94
Toggle	LARVA	22,393.54	159	10	0.00	1.864
	MARQ	186.12	733	10	0.03	0.40
	MUFIN	0.52	38	10	9.97	7.73

The results of the Java track given in table 4 are less obvious. We have four cases where LARVA failed to complete monitoring within the time limit. There was also one case where MARQ chose not to compete on a benchmark. According to the tool developer this was due to the complexity of the benchmark making it time-consuming to translate and debug. In general, MUFIN had significantly lower running times. Both LARVA and MARQ struggled due to garbage collection. LARVA is not optimised for memory leaks of this kind and MARQ switched off one of its optimisations prior to the competition due to a bug.

5.2 Scores and Winners

Table 5 gives the total scores for each tool in each track. This gives MARQ as the winner of the Offline track and MUFIN as the winner of the Java track. In previous years it has been the case that the ranking of average scores has not agreed with the ranking of total scores as some tools decided to only compete on

Table 5. Total Scores

Team	Submissions	Correctness	Time	Memory	Total	Average
Offline Track						
BeepBeep 3	6	60	14.42	25.51	97.93	16.32
MarQ	6	45	45.58	36.49	127.07	21.18
Java Track						
Larva	9	45	10.88	15.36	71.24	7.92
MarQ	8	80	20.25	17.30	117.65	14.71
Mufin	9	90	58.87	57.34	206.21	22.91

a subset of the benchmarks they were suited to. This was not the case this year, with the average score and total score rankings being the same.

6 Discussion of Trace Formats

In this section we will briefly discuss some observations about the trace formats introduced for the Offline track. Throughout different iterations of the competition, tools have either been developed around the advertised competition trace formats or chosen to translate their existing format into one of the competition ones. There is a growing interest in the best way to capture traces [12] and we briefly discuss three cases where other traces have needed to be translated.

MonPoly. In the first iteration of the competition the MonPoly tool already had a native trace format that they translated into the CSV format of the competition. The main issue that needed to be overcome was that MonPoly supports multiple events per time step i.e. an event is a set of labelled observations. The translation necessarily introduced an additional time step field and arbitrarily ordered events coming from the same time step.

BeepBeep 3. The trace files submitted by BeepBeep 3 this year did not conform to the XML requirements of the competition as they included nested data structures. A single event consisted of a variable number of character objects, each describing a different Pingu character. To translate this into the CSV format, the organisers introduced an event per character object, with the other metadata being copied between these new events. This led to additional orderings that did not occur in the original trace as a timestamp parameter was required to differentiate between characters occurring in different original events.

CRL. The benchmark submitted by CRL this year did not follow the required format. It consisted of separate files giving different parts of the overall behaviour. As the traces were related by timestamps it was relatively straightforward to merge the traces into a single trace file. However, the idea that different behaviour is recorded separately and then merged is reasonable. In this case, there was one trace file for inter-aircraft communication and one trace file per aircraft giving position information.

Discussion. These observations suggest that the trace format should be extended to allow either more complex structures as data values in events or the notion of multiple events occurring unordered at a single point in time. In both of the affected cases above, flattening the events led to more complex specifications that needed to deal with the arbitrary ordering of events that should be observed at the same point. Additionally, the last example suggests supporting traces in multiple files may be useful.

7 Feedback and Reflection

As part of preparing this report we contacted all participants in this and the two previous competitions and asked a number of questions about their experiences. More broadly we asked for general thoughts on the design and future of the competition. Here we summarise the result of this feedback, along with some thoughts of our own, organised around challenge areas.

7.1 Engagement and The Missing C Track

In the first year of the competition, 17 teams registered their interest and 10 teams submitted something. In the second year, 14 teams registered their interest and 7 teams submitted something. This year, 8 teams registered their interest and 5 submitted something.

Last year we identified the fact that entering the competition was a lot of work so this year we reduced the number of benchmarks. However, as one participant pointed out, this has drawbacks as there is more scope for over-fitting. The notion of a benchmark repository (discussed below) could sidestep this issue.

The main reason past participants gave for not re-entering was that they did not foresee any new insights coming from entering. One participant said *"We did not expect any new insights about the performance of our tool, since no major changes to our tool were made"*. Another made a suggestion *"I suggest to have such a competition every second year. I am not sure if there are many changes and improvements to too many tools within a year."*. This seems like a reasonable suggestion and we discuss this idea further below.

Whilst most participants were positive about the relevance of the competition the same participants expressed disappointment in the impact of the competition so far. One reason for this is the lack of engagement: *"even for the first competition, I was disappointed that only a few teams participated"*. Another pointed out that we have not taken full advantage of the process: *"I was hoping for a more sustainable report of the competition and its results."*. Lastly, due to logistic issues, the results from last years competition were only announced on a website some time after the announcement at the conference leading one participant to comment *"If winners are not announced, why participate?"*, a reasonable point.

Finally, it is disappointing that the C track is missing this year due to lack of participants. As mentioned earlier, we aimed to appeal to a wider range of tools by introducing the notion of implicit specifications and we received positive

feedback on this from the one participant. However, it seems that the competition still lacks appeal to such tools.

7.2 A Benchmark Repository

The intention of the first competition organisers was for benchmarks to be reused from year to year. However, this has proved difficult for two main reasons. Firstly, a lack of common specification language means much of the effort in dealing with benchmarks involves translating properties from one specification language to another, we discuss this more later. Secondly, without a common format for capturing benchmarks it is not clear that we have captured enough information to fully describe the benchmark. This is an issue we have attempted to address by the addition of demonstration traces and clarification requests. But benchmarks still contain ambiguities and unwritten assumptions.

If these issues can be overcome then the development of an independent benchmark repository has clear benefits as resource for the community beyond the competition. Indeed, this is a continued aim of the COST Action associated with the competition.

This idea is supported by our feedback with one participant suggesting this approach, adding that benchmarks could be slightly mutated for use in the competition to avoid over-fitting. Another participant stated that *"creating benchmarks is the costly part of the competition"* suggesting that the perceived need to submit benchmarks is a barrier to entry. It was also pointed out that re-using benchmarks can be used to analyse a tools evolution. Finally, one participant expressed a wish for benchmarks to be released at the point the competition is announced to make the amount of work required clear from the beginning. This would require an independent benchmark repository.

7.3 A Common Specification Language

It is clear that without a common specification language the competition will continue to involve a lot of hard work. In the feedback, participants spoke of days spent translating specifications by hand and one spoke of this as a reason not to enter the competition again.

The main suggestion for a common specification language is first-order LTL. We encouraged benchmarks in such a language this year but this was seen as too much additional effort by participants. One issue is that there exist a number of variants of first-order LTL in the community and it is not clear if one of these should be used or a new language developed. Once a language has been selected then each tool developer needs to consider how the selected language relates to their specification language. Whilst there has been some work on relating different specification languages for runtime verification [17] we see this as a different hurdle for participants and it is not clear which is more significant.

7.4 Achieving Better Coverage

One criticism of the competition from two participants was the lack of coverage. Currently a single trace is used for evaluation. There is therefore no guarantee that the submitted monitor implements the property correctly beyond the single known trace. The suggestion here is to have multiple traces or workloads per benchmark with some being seen and others unseen. This allows the competition to check the completeness of the submitted monitor as well as the efficiency of the monitoring tool.

7.5 Beyond (or Ignoring) Performance

It has been suggested that holding the current version of the competition every year is not useful. The suggestion is to hold different styles of competition in years where the current style is not run. The question is then what such a competition should look like. One comment that came from the feedback is that a concentration on performance leads to a style of research that does not necessarily lead to usable tools. Below we list some suggestions for alternative focuses.

Different monitoring scenarios. One participant suggested a scenario where several properties are checked for a single trace. Another suggestion would be to detect multiple violations of a single safety property or explain violations.

Hardware. The previous point was about keeping the setting but changing the problem. Another approach would be to consider a different setting. Whilst most research on hardware monitoring is difficult to compare, setting a challenging problem to be solved in plenty of time may lead to new research on solving an interesting problem.

Concurrency. Currently the issue of monitoring distributed or concurrent systems has not played a large part in the competition. An iteration of the competition focussing on this issue could encourage more focussed research.

Usability. It is often mentioned that usability of tools and specification languages is a large barrier for uptake of formal methods tools. It is not immediately clear how usability could be measured objectively. One suggestion would be to have a showcase rather than a competition. One participant suggested the use of the summer school to carry out such a study. This is an interesting idea although complex logistically.

8 Concluding Remarks

This report described the Third Competition on Runtime Verification. The organisation of the competition was reviewed along with the competing teams. The results have been announced and some reflections on the structure and organisation of the competition have been given.

Acknowledgment. Thanks to Klaus Havelund, Julien Signoles, Torben Scheffel, Domenico Bianculli, Daniel Thoma and Felix Klaedtke for providing the feedback discussed in Section 7. The *Laboratoire d'informatique formelle* from Université du Québec à Chicoutimi lent the server for hosting the wiki and running the benchmarks. This article is based upon work from COST Action ARVI IC1402, supported by COST (European Cooperation in Science and Technology).

References

1. H. Barringer, Y. Falcone, K. Havelund, G. Reger, and D. Rydeheard. Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors. In Dimitra Giannakopoulou and Dominique M., editors, *FM 2012: Formal Methods*, volume 7436 of *Lecture Notes in Computer Science*, pages 68–84. Springer Berlin Heidelberg, 2012.
2. 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.
3. 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)*, submitted.
4. Christian Colombo, Gordon J. Pace, and Gerardo Schneider. Dynamic event-based runtime monitoring of real-time and contextual properties. In *Formal Methods for Industrial Critical Systems (FMICS)*, volume 5596 of *Lecture Notes in Computer Science*, pages 135–149. Springer, 2008.
5. Christian Colombo, Gordon J. Pace, and Gerardo Schneider. Larva — safer monitoring of real-time java programs (tool paper). In *Proceedings of the 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods*, SEFM '09, pages 33–37, Washington, DC, USA, 2009. IEEE Computer Society.
6. Normann Decker, Jannis Harder, Torben Scheffel, Malte Schmitz, and Daniel Thoma. Runtime monitoring with union-find structures. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016*, volume LNCS,. Springer, Springer, 2016.
7. M. Delahaye, N. Kosmatov, and J. Signoles. Common specification language for static and dynamic analysis of c programs. In *Proceedings of SAC '13: the 28th Annual ACM Symposium on Applied Computing*, pages 1230–1235. ACM, 2013.
8. 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.
9. Yliès Falcone, Dejan Ničković, Giles Reger, and Daniel Thoma. Second international competition on runtime verification. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*, pages 405–422, Cham, 2015. Springer International Publishing.
10. Sylvain Hallé and Roger Villemaire. Runtime enforcement of web service message contracts with data. *IEEE Trans. Services Computing*, 5(2):192–206, 2012.

11. Klaus Havelund and Giles Reger. Specification of parametric monitors. In Rolf Drechsler and Ulrich Kühne, editors, *Formal Modeling and Verification of Cyber-Physical Systems: 1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015*, pages 151–189. Springer Fachmedien Wiesbaden, 2015.
12. Klaus Havelund and Giles Reger. What is a trace? a runtime verification perspective. In *7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2016)*, page (accepted), 2016.
13. Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, may/june 2008.
14. Ariane Piel. *Reconnaissance de comportements complexes par traitement en ligne de flux d'événements. (Online event flow processing for complex behaviour recognition)*. PhD thesis, Paris 13 University, Villetaneuse, Saint-Denis, Bobigny, France, 2014.
15. Giles Reger. *Automata Based Monitoring and Mining of Execution Traces*. PhD thesis, University of Manchester, 2014.
16. Giles Reger, Helena Cuenca Cruz, and David E. Rydeheard. Marq: Monitoring at runtime with QEA. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of ETAPS 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 596–610. Springer, 2015.
17. Giles Reger and David Rydeheard. From first-order temporal logic to parametric trace slicing. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*, pages 216–232. Springer International Publishing, 2015.
18. Simon Varvaressos, Kim Lavoie, Alexandre Blondin Massé, Sébastien Gaboury, and Sylvain Hallé. Automated bug finding in video games: A case study for runtime monitoring. In *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*, ICST '14, pages 143–152, Washington, DC, USA, 2014. IEEE Computer Society.