International Competition on Runtime Verification (CRV)

Ezio Bartocci[1(✉)], Yliès Falcone[2], and Giles Reger[3]

[1] TU Wien, Vienna, Austria
ezio.bartocci@tuwien.ac.at
[2] Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France
[3] University of Manchester, Manchester, UK

Abstract. We review the first five years of the international Competition on Runtime Verification (CRV), which began in 2014. Runtime verification focuses on verifying system executions directly and is a useful lightweight technique to complement static verification techniques. The competition has gone through a number of changes since its introduction, which we highlight in this paper.

1 Introduction

Runtime verification (RV) is a class of lightweight scalable techniques for the analysis of system executions [5,7,17,18]. The field of RV is broad and encompasses many techniques. The competition has considered a significant subset of techniques concerned with the analysis of user-provided specifications, where executions are checked against a property expressed in a formal specification language. The core idea of RV is to instrument a software/hardware system so that it can emit events during its execution. The sequence of such events (the so-called trace) is then processed by a monitor that is automatically generated from the specification. One usually distinguishes online from offline monitoring, depending on whether the monitor runs with the system or post-mortem (and thus collects events from a trace).

In 2014, we observed that, in spite of the growing number of RV tools developed over the previous decade, there was a lack of standard benchmark suites as well as scientific evaluation methods to validate and test new techniques. This observation motivated the promotion of a venue[1] dedicated to comparing and evaluating RV tools in the form of a competition. The Competition on Runtime Verification (CRV) was established as a yearly event in 2014 and has been organized as a satellite event of the RV conference since then [4,6,19,32,33].

[1] https://www.rv-competition.org/.

This work was partially supported by the Austrian FWF-funded National Research Network RiSE/SHiNE S11405-N23 and ADynNet project (P28182).

D. Beyer et al. (Eds.): TACAS 2019, Part III, LNCS 11429, pp. 41–49, 2019.
https://doi.org/10.1007/978-3-030-17502-3_3

Over the last five years, the competition has helped to shape the development of new tools and evaluation methods but the broad objectives of the competitions remain the same. CRV aims to:

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

Related Work. Over the last two decades, we have witnessed the establishment of several software tool competitions [1,3,9,22–24,34] with the goal of advancing the state-of-the-art in the computer-aided verification technology.

In particular, in the area of software verification, there are three related competitions: SV-COMP [9], VerifyThis [23] and the RERS Challenge [22].

SV-COMP targets tools for software model checking, while CRV is dedicated to monitoring tools analyzing only a single program's execution using runtime and offline verification techniques. While in software model checking the verification process is separated from the program execution, runtime verification tools introduce instead an overhead for the monitored program and they consume memory resources affecting the execution of the program itself. As a consequence CRV assigns a score to both the overhead and the memory utilization. Another related series of competitions are VerifyThis [23] and the *Rigorous Examination of Reactive Systems (RERS)* challenge [22] that provide to the participants verification problems to be solved. On the contrary of CRV format, these competitions are problem centred and focus on the problem solving skills of the participants rather than on the tool characteristics and performance.

In the remainder of this paper, we discuss the early years of the competition during 2014–2016 (Sect. 2), the activities held in 2017 and 2018 that have shifted the focus of the competition (Sect. 3), and what the future holds for the competition in 2019 and beyond (Sect. 4).

2 The Early Years: 2014–2016

The early competition was organized into three different tracks: (1) offline monitoring, (2) online monitoring of C programs, and (3) online monitoring of Java programs. The competition spanned over several months before the

announcement of results during the conference. The competition consisted of the following steps:

1. **Registration** collected information about participants.
2. **Benchmark Phase.** In this phase, participants submitted benchmarks to be considered for inclusion in the competition.

Table 1. Participants in CRV between 2014 and 2016.

Offline Track	Online C Track	Online Java Track
AgMon [26]	E-ACSL [14]	JavaMOP [25]
BeepBeep3 [20]	MarQ [2]	JUnitRV [13]
Breach	RiTHM-1 [29]	Larva [10]
CRL [31]	RTC [28]	MarQ [2]
LogFire [21]	RV-Monitor [27]	Mufin [12]
MonPoly [8]	TimeSquare [11]	RV-Monitor [27]
MarQ [2]		
OCLR-Check [16]		
OptySim [15]		
RiTHM-2 [29]		
RV-Monitor [27]		

3. **Clarification Phase.** The benchmarks resulting from the previous phase were made available to participants. This phase gave participants 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, participants were asked to produce monitors for the eligible benchmarks. Monitors had to be runnable via a script on a Linux system. Monitor code should be generated from the participant's tool (therefore the tool had to be installable on a Linux system).
5. **Evaluation Phase.** Submissions from the previous phase were collected and executed, with relevant data collected to compute scores as described later. Participants were given an opportunity to test their submissions on the evaluation system. The outputs produced during the evaluation phase were made available after the competition.

Input Formats. The competition organizers fixed input formats for traces in the offline track. These were based on XML, JSON, and CSV and evolved between the first and second years of the competition based on feedback from participants. The CSV format proved the most popular for its simplicity and is now used by many RV tools. See the competition report from 2015 [19] for details.

Participants. Over the first three years of the competition 14 different RV tools competed in the competition in the different tracks. These are summarized in Table 1. One of these tools, Mufin, was written specifically in response to the competition and all tools were extended or modified to handle the challenges introduced by the competition.

Benchmarks. Benchmarks, as submitted by the participants, should adhere to requirements that ensured compliance with the later phases of the competition. This also ensured uniformity between benchmarks and was also the first step in building a benchmark repository dedicated to Runtime Verification. A benchmark contains two packages: a program/source package and a specification package. The program/source package includes the traces or the source of the program as well as scripts to compile and run it. In these early years of the competition, we chose to focus on closed, terminating and deterministic programs. The specification package includes an informal and a formal description (in some logical formalism), the instrumentation information (i.e., what in the program influences the truth-value of the specification), and the verdict (i.e., how the specification evaluates w.r.t. the program or trace).

In these three competitions, over 100 benchmarks were submitted and evaluated. All benchmarks are available from the competition website[2] organized in a repository for each year.

Evaluation Criteria/Scores. Submissions from the participants were evaluated on correctness and performance. For this purpose, we designed an algorithm that uses as inputs (i) the verdicts produced by each tool over each benchmark (ii) the execution time and memory consumption in doing so, and produces as output a score reflecting the evaluation of the tool regarding correctness and performance (the higher, the better). Correctness criteria included (i) finding the expected verdict, absence of crash, and the possibility of expressing the benchmark specification in the tool formalism. Performance criteria were based on the classical time and memory overheads (lower is better) with the addition that the score of a participant accounts for the performance of the other participants (e.g., given the execution time of a participant, more points would be given if the other participants performed poorly) using the harmonic mean. Tools were evaluated against performance, only when they produced a correct result (negative points were given to incorrect results). A benchmark score was assigned for each tool against each submitted benchmark, and the tool final score was the sum of all its benchmark scores. A participant could decide not to compete on a benchmark and would get a zero score for this benchmark.

Experimental Environment, Availability, Reproducibility, Quality. Git-based repositories and wiki pages were provided to the participants to share their benchmarks and submissions. This facilitated the communication and ensured transparency. To run the experiments, we used DataMill [30], to ensure robust

[2] https://www.rv-competition.org/benchmarks/.

and reproducible experiments. We selected the most powerful and general-purpose machine and evaluated all submissions on this machine. DataMill ensured flexibility and fairness in the experiments. Benchmarks could be setup and submitted via a Web interface and then be scheduled for execution. DataMill ensured that only one monitor was running on the machine at a time, in addition to a minimalist operating system, cleaned between each experiments. Execution times and memory consumption measures were obtained by averaging 10 executions. Results were available through the Web interface.

Table 2. Winners of CRV between 2014 and 2016.

Year	Offline Track	Online C Track	Online Java Track
2014	MarQ	RiTHM-1	MarQ & JavaMOP (joint)
2015	LogFire	E-ACSL	Mufin
2016	MarQ	-	Mufin

Winners. Table 2 indicates the winners in each track in each year. The detailed results are available from the competition website and associated reports [4,6,19]. In 2014, the scores in the Online Java track were so close that a joint winner was announced. In 2016, only one participant entered the C track and the track was not run. (We note that, more tools have been developed for monitoring Java programs thanks to the AspectJ support for instrumentation.)

Issues. The early years of the competition were successful in encouraging RV tool developers to agree on common formats but the number of participants dropped in each year with two main issues identified:

1. The amount of work required to enter was high. This was mainly due to the need to translate each benchmark into the specification language of the entered tool. Common specification languages would address this problem but there was no agreement on such languages at the time.
2. It was not clear how good the benchmarks were at differentiating tools. More work was required to understand which benchmarks were useful for evaluating RV tools.

The next two years of activities addressed these issues as described below.

3 Shifting Focus: 2017–2018

In 2017, the competition was replaced by a workshop (called RV-CuBES) [33] aimed at reflecting on the experiences of the last three years and discussing future directions. A workshop was chosen over a competition as there was strong feedback from participants in 2016 that the format of the competition should be revised (mainly to reduce the amount of work required by participants). It was

decided that this was a good opportunity to reassess the format of the competition in an open setting. The workshop attracted 12 tool description papers and 5 position papers and led to useful discussion at the 2017 RV conference. A full account can be found in the associated report.

A suggestion of the workshop was to hold a benchmark challenge focusing on collecting relevant new benchmarks. Therefore, in 2018 a benchmark challenge was held with a track for Metric Temporal Logic (MTL) properties and an Open track. The purpose of the MTL track was to see what happened when participants were restricted to a single input language whilst the Open track gave full freedom on the choice of the specification language.

There were two submissions in the MTL track and seven in the Open track. The submissions in the Open track were generally in much more expressive languages than MTL and no two submissions used the same specification language. All submissions were evaluated by a panel of experts and awarded on qualities in three categories: (1) correctness and reliability (2) realism and challenge and (3) utility in evaluation. As a result of the evaluation two benchmark sets were identified for use in future competitions (see below).

4 Back to the Future

The 2019 competition is now in its initial stages and will return to a competition comparing tools, using the benchmarks from the 2018 challenge. The competition will use two specification languages: MTL and a future-time first-order temporal logic. We have chosen to fix two specification languages (with differing levels of expressiveness) to reduce the overall work for participants. Standardising the specification language of the competition has been a goal of the competition from the start and the benchmark challenge has allowed us to pick two good candidates. MTL was chosen as it can be considered a 'smallest shared' specification language in terms of expressiveness and usage. Similarly, the future-time first-order temporal logic was chosen as it can be considered a 'largest shared' specification language in terms of expressiveness and usage.

Beyond 2019, there are many opportunities to take the competition in different directions. For example, a key issue in RV is that of specifications. Thus, when organizing a competition, one may wonder whether a competition could also focus on evaluating aspects related to specifications (e.g., expressiveness, succinctness and elegance of specifications). Moreover, in so far, the competition has neglected the area of hardware monitoring, and the comparison of tools in such domains remains an open question. We note that there have been less research efforts on monitoring hardware where instrumentation aspects are more challenging. The main reasons for common specification languages not being used in the early years stemmed from two facts: (i) a main research activity in RV consists in developing new languages to have alternative representation of problems (ii) the monitoring algorithm of an RV tool is often closely coupled to the input language. Hence, a challenge is to rely on a shared specification language whilst encouraging research that explores the relationship between input language and performance or usability.

Acknowledgements. The authors would like to thank the anonymous reviewers for their helpful comments and all the colleagues involved in CRV over the last years (B. Bonakdarpour, D. Thoma, D. Nickovic, S. Hallé, K. Y. Rozier, and V. Stolz).

References

1. Barrett, C., Deters, M., de Moura, L.M., Oliveras, A., Stump, A.: 6 Years of SMT-COMP. J. Autom. Reasoning **50**(3), 243–277 (2013)
2. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.: Quantified event automata: towards expressive and efficient runtime monitors. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 68–84. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32759-9_9
3. Bartocci, E., et al.: TOOLympics 2019: an overview of competitions in formal methods. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019, Part III. LNCS, vol. 11429, pp. 3–24. Springer, Cham (2019)
4. Bartocci, E., Bonakdarpour, B., Falcone, Y.: First international competition on software for runtime verification. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 1–9. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_1
5. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification - Introductory and Advanced Topics. LNCS, vol. 10457. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5
6. Bartocci, E., et al.: First international competition on runtime verification: rules, benchmarks, tools, and final results of CRV 2014. Int. J. Softw. Tools Technol. Transfer **21**, 31–70 (2019)
7. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 1–33. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_1
8. Basin, D., Harvan, M., Klaedtke, F., Zălinescu, E.: MONPOLY: monitoring usage-control policies. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 360–364. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_27
9. Beyer, D.: Software verification and verifiable witnesses - (report on SV-COMP 2015). In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 401–416. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_31
10. Colombo, C., Pace, G.J., Schneider, G.: Larva — safer monitoring of real-time java programs (tool paper). In: Van Hung, D., Krishnan, P. (eds.) Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM 2009, Hanoi, Vietnam, 23–27 November 2009, pp. 33–37. IEEE Computer Society (2009). https://doi.org/10.1109/SEFM.2009.13. ISBN 978-0-7695-3870-9
11. DeAntoni, J., Mallet, F.: TimeSquare: treat your models with logical time. In: Furia, C.A., Nanz, S. (eds.) TOOLS 2012. LNCS, vol. 7304, pp. 34–41. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30561-0_4
12. Decker, N., Harder, J., Scheffel, T., Schmitz, M., Thoma, D.: Runtime monitoring with union-find structures. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 868–884. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_54

13. Decker, N., Leucker, M., Thoma, D.: jUnit^{RV}–adding runtime verification to jUnit. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 459–464. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_34
14. Delahaye, M., Kosmatov, N., Signoles, J.: Common specification language for static and dynamic analysis of C programs. In: Proceedings of SAC 2013: The 28th Annual ACM Symposium on Applied Computing, pp. 1230–1235. ACM, March 2013
15. Díaz, A., Merino, P., Salmeron, A.: Obtaining models for realistic mobile network simulations using real traces. IEEE Commun. Lett. **15**(7), 782–784 (2011)
16. Dou, W., Bianculli, D., Briand, L.: A model-driven approach to offline trace checking of temporal properties with OCL. Technical report SnT-TR-2014-5, Interdisciplinary Centre for Security, Reliability and Trust (2014)
17. Falcone, Y.: You should better enforce than verify. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 89–105. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16612-9_9
18. Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. In: Broy, M., Peled, D.A., Kalus, G. (eds.) Engineering Dependable Software Systems. NATO Science for Peace and Security Series, D: Information and Communication Security, vol. 34, pp. 141–175. IOS Press (2013)
19. Falcone, Y., Ničković, D., Reger, G., Thoma, D.: Second international competition on runtime verification. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 405–422. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23820-3_27
20. Hallé, S.: When RV Meets CEP. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 68–91. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_6
21. Havelund, K.: Rule-based runtime verification revisited. STTT **17**(2), 143–170 (2015). https://doi.org/10.1007/s10009-014-0309-2
22. Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D., Pasareanu, C.S.: Rigorous examination of reactive systems - the RERS challenges 2012 and 2013. STTT **16**(5), 457–464 (2014)
23. Huisman, M., Klebanov, V., Monahan, R.: Verifythis 2012 - a program verification competition. STTT **17**(6), 647–657 (2015)
24. Järvisalo, M., Berre, D.L., Roussel, O., Simon, L.: The international SAT solver competitions. AI Mag. **33**(1), 89–92 (2012)
25. Jin, D., Meredith, P.O., Lee, C., Roşu, G.: JavaMOP: efficient parametric runtime monitoring framework. In: Proceedings of ICSE 2012: The 34th International Conference on Software Engineering, Zurich, Switzerland, 2–9 June, pp. 1427–1430. IEEE Press (2012)
26. Kane, A., Fuhrman, T.E., Koopman, P.: Monitor based oracles for cyber-physical system testing: practical experience report. In: 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, 23–26 June 2014, pp. 148–155. IEEE (2014)
27. Luo, Q., et al.: RV-monitor: efficient parametric runtime verification with simultaneous properties. In: Proceedings of Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, 22–25 September 2014, pp. 285–300 (2014)
28. Milewicz, R., Vanka, R., Tuck, J., Quinlan, D., Pirkelbauer, P.: Runtime checking C programs. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 2107–2114. ACM (2015)

29. Navabpour, S., et al.: RiTHM: a tool for enabling time-triggered runtime verification for C programs. In: ACM Symposium on the Foundations of Software Engineering (FSE), pp. 603–606 (2013)
30. Petkovich, J., de Oliveira, A.B., Zhang, Y., Reidemeister, T., Fischmeister, S.: Datamill: a distributed heterogeneous infrastructure for robust experimentation. Softw. Pract. Exper. **46**(10), 1411–1440 (2016)
31. Piel, A.: Reconnaissance de comportements complexes par traitement en ligne de flux d'événements. (Online event flow processing for complex behaviour recognition). Ph.D. thesis, Paris 13 University, Villetaneuse, Saint-Denis, Bobigny, France (2014)
32. Reger, G., Hallé, S., Falcone, Y.: Third international competition on runtime verification. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_3
33. Reger, G., Havelund, K. (eds.): RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools. Kalpa Publications in Computing, vol. 3. EasyChair (2017)
34. Sutcliffe, G.: The 5th IJCAR automated theorem proving system competition - CASC-J5. AI Commun. **24**(1), 75–89 (2011)