COST Action IC 1402 ArVI: Runtime Verification Beyond Monitoring

Activity Report

Working Group 2 - Standardization, Benchmarks, and Tool Interoperability
Chair: Giles Reger

Abstract. This report presents the activities of the second working group of the COST Action ArVI, Runtime Verification beyond Monitoring. The report describes the delivered contributions in the areas of running Runtime Verification Competitions, categorising and understanding the set of existing Runtime Verification tools and formats and infrastructure for the production and sharing of Runtime Verification benchmarks.

1 Introduction

Runtime Verification (RV), as a field of research, is referred to by many names such as runtime monitoring, trace analysis, dynamic analysis, passive testing, runtime enforcement etc. (see [4,24,31,37]). We refer the reader to the report of Working Group 1 of this Action for further details on the definition of the field. This report concerns the second Working Group and its focus on Standardization, Benchmarks, and Tool Interoperability.

One of the most important jobs of this working group was to get the different tool developers to talk to each other. We organised two events outside the Action to encourage this. Firstly, the RV-CuBES workshop [49,45] was held alongside the 17th International Conference on Runtime Verification [36]. This contained 11 short tool papers and 5 position papers discussing how RV tools should be evaluated [11,53,55], describing challenges of using RV tools in industry [29], and encouraging the community to use open standards [34]. Secondly, a Dagstuhl seminar [32] considered various issues around behavioural specification languages, inviting researchers from outside the RV community to join the discussion.

Structure of Report. We organise the report into three main sections and highlight the main results from each section here:

- *Competitions (Section 2).* Between 2014 and 2016 three competitions were carried out comparing Runtime Verification tools for monitoring C programs, Java programs, and log files. These competitions compared 14 tools using over 100 different benchmarks.
- *Tools (Section 3).* A survey has identified over 50 runtime verification tools, a taxonomy has been developed and used to classify 20 tools, various concrete steps have been taken in the direction of tool interoperability.

– *Benchmarks (Section 4)*. Standard formats for benchmarks have been agreed and over 100 benchmarks have been collected and shared.

Finally, we describe ongoing activities (Section 5).

2 Runtime Verification Competition and Challenges

The Runtime Verification Competition is an annual event established in 2014 and steered by the COST Action. It is traditionally held as a satellite event of the main Runtime Verification conference. Over the first three years of the competition 14 different runtime verification tools competed on over 100 different benchmarks. In general, the objective of the competitions are to:

– stimulate the development of new efficient and practical runtime verifica- tion 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.

Each competition has consisted of several steps described below. Competitions span over several months before the announcement of results during the conference.

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 were (will be) made available after the competition.

Below we give a little more information about each competition.

CSRV14
The First International Competition on Runtime Verification (CRV) was held in September 2014, in Toronto, Canada, as a satellite event of the 14th international conference on Runtime Verification (RV14). The event was organized in three tracks: (1) offline monitoring, (2) online monitoring of C programs, and (3) online monitoring of Java programs. Complete details about CSRV14 can be found in the dedicated reports [3].

CRV15
The Second International Competition on Runtime Verification (CRV-2015) was held as a satellite event of the 15th International Conference on Runtime Verification (RV15). The competition consisted of three tracks: offline monitoring, online monitoring of C programs, and online monitoring of Java programs. Complete details about CRV15 can be found in the dedicated report [26].

CRV16
The Third International Competition on Runtime Verification (CRV-2016) was held as a satellite event of the 16th International Conference on Runtime Verification (RV16). The competition consisted of two tracks: offline monitoring of traces and online monitoring of Java programs. Complete details about CRV16 can be found in the dedicated report [47].

RVBC18
In 2017 the competition was replaced by a workshop [49] aimed at reflecting on the experiences of the last three years and discussing future directions. A suggestion of the workshop was to hold a benchmark challenge focussing on collecting relevant and impactful benchmarks. Therefore, in 2018 a benchmark challenge was held with a track for MTL properties and an Open track.

The Future
The plan is now to return to a competition as in the first three years of the competition but making use of the wealth of benchmarks generated by the activities of RV-CuBES and RVBC18.

3 Tools

One of the goals of the working group was to better understand and organise the various RV tools being developed. The main deliverables from this section are stored at https://www.rv-competition.org/tools/.

3.1 Surveys

An important first step is to understand the set of available RV tools. We have conducted three surveys to collect such information.

– *Phase 1* - In 2015 an initial survey was distributed to relevant mailing lists asking for information on historical or actively developed RV tools. This received many responses identifying 35 such tools.

- *Phase 2* - In 2017 the RV-CuBES workshop invited short tool overview papers describing currently developed RV tools. This received 11 submissions. Concurrently, a Dagstuhl workshop was organised and asked participants to submit information about their developed tools, this received information on 12 tools.
- *Phase 3* - In 2018/9. Following the classification attempt described below a more detailed survey has been designed and distributed. This survey is still in-progress and aims to collect information able to classify many more RV tools in further detail.

3.2 Taxonomy and Classification

In conjunction with Working Group 1, a taxonomy was developed to describe Runtime Verification tools [25]. The top-level of this taxonomy is given in Figure 1 and a full explanation is given in the associated paper. The taxonomy was then used to classify 20 significant RV tools. This classification (also reported in [25]) is reproduced in Tables 1-3 at the end of this report and a live version is available at the above linked website.

3.3 Interoperability

We report on the progress of sub-working group actions towards interoperability of tools.

Common Trace Formats. This is described in the Benchmarks section as it is most pertinent to the standardisation of benchmarks.

Common Input Languages. The working group has encouraged a number of activities exploring the links between specification languages for Runtime Verification [50,51,33]. This has been both theoretically (defining translations between languages) and pragmatically (discussing topics such as usability). The 2018 Runtime Verification Benchmark Challenge concluded in proposing two standard specification languages for use in future iterations of the competition: Metric Temporal Logic (MTL) and future-time function-free First-Order Linear Temporal Logic.

Common Interface Definitions. Initial progress in this sub-working group halted when it was established that the community saw little need for such interfaces. It was noted that an ad-hoc defacto interface was already in use in the form of AspectJ aspects for monitoring Java programs.

4 Benchmarks

A major goal of this working group was to collect, categorise, and make available sets of benchmarks for comparing RV tools. The main deliverables from this section are stored at https://www.rv-competition.org/benchmarks/.

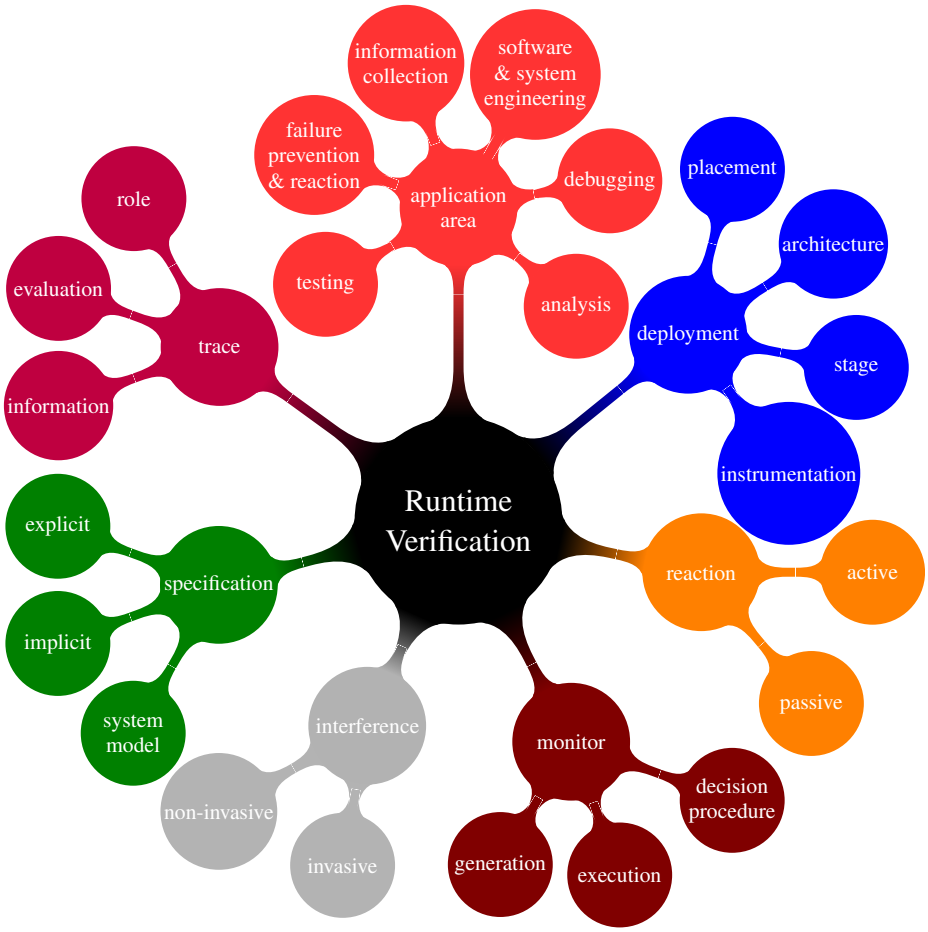

Fig. 1. Top-level representation of RV tool taxonomy [25].

Competition Trace Formats

The 2014 competition introduced trace formats in XML, CSV, and JSON and the 2015 competition refined these definitions. Below we give the final agreed format definitions.

At an abstract level, we define traces as finite sequences of events. An event is an entity that has a name and arguments each of which has a name and a value i.e. it is a named record of the form:

```
NAME{
  field1 : value1,
  ...
  fieldn : valuen
}
```

The accepted trace formats are XML, CSV, or JSON format (all following the official standards for these formats). Below, we illustrate the three formats accepted for traces, using `an_event_name` to range over the set of possible event names, `a_field_name` to range over the set of possible field names, and `a_value` to range over the set of possible runtime values.

- In XML format:

```
<log>
  <event>
   <name>an_event_name</name>
   <field>
     <name>a_field_name</name>
     <value>a_value</value>
   </field>
   <field>
     <name>a_field_name</name>
     <value>a_value</value>
   </field>
 </event>
</log>
```

- In CVS format[1], where the spaces are intended and required.

```
event, a_field_name, a_field_name, a_field_name
an_event_name, a_field_value, a_field_value, a_field_value
an_event_name, a_field_value, a_field_value, a_field_value
```

  Note that we require a single header giving the field name for different events. This can either name fields arbitrarily (i.e. arg1,arg2 etc) or have as many columns as field names, leaving some columns blank for some records.

- In JSON format[2]:

```
an_event_name: {
  a_field_name: a_value,
  a_field_name: a_value
}
```

  Whilst JSON supports arbitrarily structured values we do not currently support such events.

Other Work on Trace Formats. These have been the subject of further exploration and discussion [48,34].

[1] Following the standard http://www.ietf.org/rfc/rfc4180.txt
[2] Following the standard https://tools.ietf.org/html/rfc7159

Meta-Data

From 2015 onwards we have collected the following meta-data for benchmarks:

- Description of setting/context
- Specification
  - English Language description
  - Formalisation in at least one specification language
  - Instrumentation information describing the connection between specification and artefact
- Data
  - A collection of labelled traces demonstrating good/bad behaviour
  - A collection of labelled artefacts (trace/program) for monitoring

Overview of Benchmark Collections

These are referenced directly from the above website. The following collections have been created and curated:

- **CRV14.** This contains benchmarks from the 2014 competition. Benchmarks are in three categories:
  - C - 4 benchmark sets (20 benchmarks in total)
  - Java - 5 benchmark sets (25 benchmarks in total)
  - Offline - 7 benchmark sets (35 benchmarks in total)

  Each benchmark set is accompanied by instructions on how to use the benchmark set and the submitted solutions for each benchmark set from all participants. This provides sufficient information to replicate the results.
- **CRV15.** This contains benchmarks from the 2015 competition. Benchmarks are in three categories:
  - C - 4 benchmark sets (20 benchmarks in total)
  - Java - 3 benchmark sets (15 benchmarks in total)
  - Offline - 6 benchmark sets (30 benchmarks in total)

  Each benchmark set is accompanied by the above detailed meta-data.
- **CRV16.** This contains benchmarks from the 2015 competition. Benchmarks are in two categories:
  - Java - 3 benchmark sets (9 benchmarks in total)
  - Offline - 3 benchmark sets (9 benchmarks in total)

  Each benchmark set is accompanied by the above detailed meta-data.
- **RVBC18.** This contains benchmarks from the 2018 benchmark challenge. Benchmarks are in two categories:
  - MTL - 2 submissions of benchmark generators
  - Open - 7 submissions of benchmark sets of varying size and complexity

  Each benchmark set is accompanied by a short paper describing the benchmark set.

5 Summary

The activities of WG2 led to 5 years of successful activities surrounding competitions, challenges, and workshops relating to the development of tools, benchmarks, and ideas for how to evaluate and classify tools for Runtime Verification. This happened over numerous COST meetings and third-party meetings, including three meetings exclusively dedicated to the activities of the Working Group. We would like to thank all those who contributed.

Looking to the future, the competition will continue, the benchmark repositories will continue to be maintained and extended, and the large tool classification effort is under-way. We expect further positive outcomes from this Working Group over the coming few years.

References

1. Azzopardi, S., Colombo, C., Ebejer, J.P., Mallia, E., Pace, G.: Runtime verification using VALOUR. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 10–18. EasyChair (2017)
2. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.E.: Quantified event automata: Towards expressive and efficient runtime monitors. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 68–84. Springer (2012)
3. Bartocci, E., Bonakdarpour, B., Falcone, Y.: First international competition on software for runtime verification. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8734, pp. 1–9. Springer (2014)
4. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457, pp. 1–33. Springer (2018)
5. Basin, D.A., Bhatt, B.N., Traytel, D.: Almost event-rate independent monitoring of metric temporal logic. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 94–112. Springer (2017)
6. Basin, D.A., Harvan, M., Klaedtke, F., Zalinescu, E.: MONPOLY: monitoring usage-control policies. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 360–364. Springer (2011)
7. Basin, D.A., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. J. ACM 62(2), 15:1–15:45 (2015)
8. Basin, D.A., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 19–28. EasyChair (2017)
9. Basin, D.A., Krstic, S., Traytel, D.: Almost event-rate independent monitoring of metric dynamic logic. In: Lahiri, S.K., Reger, G. (eds.) RV 2017. LNCS, vol. 10548, pp. 85–102. Springer (2017)
10. Basin, D.A., Krstic, S., Traytel, D.: AERIAL: almost event-rate independent algorithms for monitoring metric regular properties. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 29–36. EasyChair (2017)
11. Bianculli, D., Krstic, S.: On the risk of tool over-tuning in run-time verification competitions (position paper). In: Reger and Havelund [49], pp. 37–40, https://easychair.org/publications/paper/N6cC

12. Bonakdarpour, B., Navabpour, S., Fischmeister, S.: Time-triggered runtime verification. Formal Methods in System Design 43(1), 29–60 (Aug 2013)
13. Cassar, I., Francalanza, A., Attard, D.P., Aceto, L., Ingólfsdóttir, A.: A suite of monitoring tools for Erlang. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 41–47. EasyChair (2017)
14. Colombo, C., Pace, G.J.: Runtime verification using LARVA. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 55–63. EasyChair (2017)
15. Colombo, C., Pace, G.J., Schneider, G.: Dynamic event-based runtime monitoring of real-time and contextual properties. In: Cofer, D.D., Fantechi, A. (eds.) FMICS 2008. LNCS, vol. 5596, pp. 135–149. Springer (2008)
16. Colombo, C., Pace, G.J., Schneider, G.: LARVA — safer monitoring of real-time java programs (tool paper). In: Hung, D.V., Krishnan, P. (eds.) SEFM 2009. pp. 33–37. IEEE Computer Society (2009)
17. Decker, N., Harder, J., Scheffel, T., Schmitz, M., Thoma, D.: Runtime monitoring with union-find structures. In: Chechik, M., Raskin, J. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 868–884. Springer (2016)
18. Decker, N., Leucker, M., Thoma, D.: jUnit$^{\mathrm{rv}}$-adding runtime verification to jUnit. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 459–464. Springer (2013)
19. Decker, N., Leucker, M., Thoma, D.: Monitoring modulo theories. STTT 18(2), 205–225 (2016)
20. Delahaye, M., Kosmatov, N., Signoles, J.: Common specification language for static and dynamic analysis of C programs. In: Shin, S.Y., Maldonado, J.C. (eds.) SAC 2013. pp. 1230–1235. ACM (2013)
21. Dou, W., Bianculli, D., Briand, L.: A model-driven approach to offline trace checking of temporal properties with OCL. Tech. Rep. SnT-TR-2014-5, Interdisciplinary Centre for Security, Reliability and Trust (2014), http://hdl.handle.net/10993/16112
22. Dou, W., Bianculli, D., Briand, L.: TemPsy-Check: a tool for model-driven trace checking of pattern-based temporal properties. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 64–70. EasyChair (2017)
23. Drabek, C., Weiss, G.: DANA – description and analysis of networked applications. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 71–80. EasyChair (2017)
24. 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)
25. Falcone, Y., Kristc, S., Reger, G., Traytel, D.: A taxonomy for classifying runtime verification tools. In: Colombo, C., Leucker, M. (eds.) Proceedings of the 18th International Conference on Runtime Verification, Lecture Notes in Computer Science, vol. in this volume
26. Falcone, Y., Nickovic, D., Reger, G., Thoma, D.: Second international competition on runtime verification CRV 2015. In: Bartocci, E., Majumdar, R. (eds.) Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9333, pp. 405–422. Springer (2015)
27. Hallé, S.: When RV meets CEP. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 68–91. Springer (2016)
28. Hallé, S., Khoury, R.: Event stream processing with BeepBeep 3. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 81–88. EasyChair (2017)

29. Hallé, S., Khoury, R., Gaboury, S.: A few things we heard about RV tools (position paper). In: Reger and Havelund [49], pp. 89–95, https://easychair.org/publications/paper/q246
30. Havelund, K.: Rule-based runtime verification revisited. STTT 17(2), 143–170 (2015)
31. Havelund, K., Goldberg, A.: Verify your runs. In: Meyer, B., Woodcock, J. (eds.) Verified Software: Theories, Tools, Experiments, First IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 10-13, 2005, Revised Selected Papers and Discussions. Lecture Notes in Computer Science, vol. 4171, pp. 374–383. Springer (2005)
32. Havelund, K., Leucker, M., Reger, G., Stolz, V.: A shared challenge in behavioural specification (Dagstuhl seminar 17462). Dagstuhl Reports 7(11), 59–85 (2017), https://doi.org/10.4230/DagRep.7.11.59
33. Havelund, K., Reger, G.: Runtime verification logics A language design perspective. In: Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday. pp. 310–338 (2017), https://doi.org/10.1007/978-3-319-63121-9_16
34. Jakšić, S., Leucker, M., Li, D., Stolz, V.: COEMS – open traces from the industry. In: Reger and Havelund [49], pp. 96–105, https://easychair.org/publications/paper/QljX
35. Jin, D., Meredith, P.O., Lee, C., Rosu, G.: JavaMOP: Efficient parametric runtime monitoring framework. In: Glinz, M., Murphy, G.C., Pezzè, M. (eds.) ICSE 2012. pp. 1427–1430. IEEE Computer Society (2012)
36. Lahiri, S.K., Reger, G. (eds.): Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings, Lecture Notes in Computer Science, vol. 10548. Springer (2017), https://doi.org/10.1007/978-3-319-67531-2
37. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. 78(5), 293–303 (2009)
38. Luo, Q., Zhang, Y., Lee, C., Jin, D., Meredith, P.O., Serbanuta, T., Rosu, G.: RV-Monitor: efficient parametric runtime verification with simultaneous properties. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 285–300. Springer (2014)
39. Meredith, P.O., Jin, D., Griffith, D., Chen, F., Rosu, G.: An overview of the MOP runtime verification framework. STTT 14(3), 249–289 (2012)
40. Milewicz, R., Vanka, R., Tuck, J., Quinlan, D., Pirkelbauer, P.: Lightweight runtime checking of C programs with RTC. Comput. Lang. Syst. Str. 45, 191–203 (2016)
41. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. Form. Methods Sys. Des. 51(1), 31–61 (2017)
42. Navabpour, S., Joshi, Y., Wu, C.W.W., Berkovich, S., Medhat, R., Bonakdarpour, B., Fischmeister, S.: RiTHM: a tool for enabling time-triggered runtime verification for C programs. In: Meyer, B., Baresi, L., Mezini, M. (eds.) ESEC/FSE 2013. pp. 603–606. ACM (2013)
43. Rapin, N.: ARTiMon monitoring tool, the time domains. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 106–122. EasyChair (2017)
44. Reger, G.: An overview of MarQ. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 498–503. Springer (2016)
45. Reger, G.: A report of RV-CuBES 2017. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 1–9. EasyChair (2017)
46. Reger, G., Cruz, H.C., Rydeheard, D.E.: MarQ: monitoring at runtime with QEA. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 596–610. Springer (2015)
47. Reger, G., Hallé, S., Falcone, Y.: Third international competition on runtime verification - CRV 2016. In: Falcone, Y., Sánchez, C. (eds.) Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10012, pp. 21–37. Springer (2016)

48. Reger, G., Havelund, K.: What is a trace? A runtime verification perspective. pp. 339–355. https://doi.org/10.1007/978-3-319-47169-3_25
49. 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)
50. Reger, G., Rydeheard, D.: From parametric trace slicing to rule systems. In: Colombo, C., Leucker, M. (eds.) Proceedings of the 18th International Conference on Runtime Verification, Lecture Notes in Computer Science, vol. in this volume
51. Reger, G., Rydeheard, D.E.: From first-order temporal logic to parametric trace slicing. In: Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings. pp. 216–232 (2015), https://doi.org/10.1007/978-3-319-23820-3_14
52. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 357–372. Springer (2014)
53. Rozier, K.Y.: On the evaluation and comparison of runtime verification tools for hardware and cyber-physical systems. In: Reger and Havelund [49], pp. 123–137, https://easychair.org/publications/paper/877G
54. Schumann, J., Moosbrugger, P., Rozier, K.Y.: Runtime analysis with R2U2: A tool exhibition report. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 504–509. Springer (2016)
55. Signoles, J.: Online runtime verification competitions: How to possibly deal with their issues (position paper). In: Reger and Havelund [49], pp. 157–163, https://easychair.org/publications/paper/m1vV
56. Signoles, J., Kosmatov, N., Vorobyov, K.: E-ACSL, a runtime verification tool for safety and security of C programs (tool paper). In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 164–173. EasyChair (2017)

Table 1. Details of the participating tools.

Tool	References	Specification formalism name + some remarks
Aerial	[10,9,5]	MDL
ARTiMon	[43]	ARTiMon (no quantification; only aggregation)
BeepBeep	[28,27]	stateful stream function API + DSLs for LTL-FO+, FSM
DANA	[23]	UML state machines
detectEr	[13]	μHML (only universal quantification)
E-ACSL	[56,20]	ACSL/implicit
JavaMOP	[39,35]	MOP with plugins (LTL,FMS,ERE,CFG,SRS,CaReT)
jUnitRV	[18,19]	Temporal Data Logic
Larva	[14,16,15]	DATEs
LogFire	[30]	LogFire DSL
MarQ/QEA	[46,2,44]	QEA
MonPoly	[8,6,7]	MFOTL
Mufin	[17]	Projection Automata
R2U2	[41,54,52]	MTL + mission time LTL
RiTHM	[42,12]	LTL_3
RTC	[40]	- (implicit)
RV-Monitor	[38]	MOP (see above)
STePr	-	Scala-internal DSL
TemPsy/OCLR-Check	[22,21]	TemPsy
VALOUR	[1]	Valour Script / Rules

Table 2. Key for Table 3.

Column	Values
	Specification
implicit	ms = *memory safety*
data	p = *propositional*, s = *simple parametric*, c = *complex parametric*
output	s = *stream*, v = *verdict*, w = *witness*, r = *robustness*
logical time	tot = *total order*, par = *partial order*
physical time	$\mathbb{N}$ = *discrete*, $\mathbb{R}$ = *dense*, none = *no time*
modality	f = *future*, p = *past*, c = *current*
	Monitor
generation	e = *explicit*, i = *implicit*
execution	i = *interpreted*, d = *direct*
	Deployment
stage	on = *online*, off = *offline*
synchronisation	sync = *synchronous*, async = *asynchronous*
architecture	c = *centralised*, d = *decentralised*
placement	out = *outline*, in = *inline*,
instrumentation	sw=*software*, swAJ = *software with AspectJ*, swR = *software with reflection*
	Reaction
active	e = *exception*, r = *recovery*
passive	so = *specification output*, e = *explanations*
	Trace
information	e = *events*, s = *states*
sampling	et = *event-triggered*, tt = *time-triggered*
evaluation	p = *points*, i = *intervals*
precision	p = *precise*, i = *imprecise*
model	f = *finite trace model*, i = *infinite trace model*
	General
	all = *all features supported*, none = *no features supported*
	na = *not applicable*, ? = *insufficient information*

Table 3. Classification of participating tools.

Tool	Specification							Monitor			Deployment					Reaction		Trace				
	implicit	explicit				modality	paradigm	decision procedure	generation	execution	stage	synchronisation	architecture	placement	instrumentation	active	passive	information	sampling	evaluation	precision	model
		data	output	time: logical	time: physical																	
Aerial	none	p	s	tot	$\mathbb{N}$	all	d	dynamic programming	i	i	on	none	c	out	none	none	so	e	et	p	p	i
ARTiMon	none	s	s	tot	$\mathbb{NR}$	all	d	?	i	i	on	none	c	out	none	none	so	e	et	i	p	i
BeepBeep	none	c	s	tot	none	f	all	stream-processing	i	d	on	all	c	out	sw	none	so	e	et	p	p	i
DANA	none	p	s	tot	$\mathbb{R}$	all	o	?	i	d	on	sync	c	?	?	none	so	e	et	p	p	f
detectEr	none	s	v	par	none	f	d	dynamic programming	e	i	on	all	c	in	sw	none	so	e	et	p	p	i
E-ACSL	ms	na	r	?	na	na	o	code rewriting with assertions	e	d	on	sync	c	in	sw	e	e	s	na	na	na	na
JavaMOP	none	s	w	tot	none	all	all	trace slicing plugin-based	e	d	on	sync	c	in	swAJ	r	e	e	et	p	p	f
jUnitRV	none	s	v	tot	none	f	d	automata-based (modulo theories eg. SMT solver)	e	d	on	sync	c	in	swR	?	?	e	et	p	p	f
Larva	none	s	v	tot	$\mathbb{N}$	f	o	automata-based	e	d	on	all	c	all	sw	r	so	e	et	p	p	f
LogFire	none	s	w	tot	none	all	o	rewriting-based (RETE)	i	d	all	sync	c	out	none	none	so	e	et	p	p	f
MarQ/ QEA	none	s	v	tot	$\mathbb{N}$	f	o	automata-based	i	d	all	sync	c	all	sw	none	so	e	et	p	p	f
MonPoly	none	s	s	tot	$\mathbb{N}$	all	d	first-order queries	i	i	on	none	c	out	none	none	so	e	et	p	p	all
Mufin	none	s	v	tot	none	f	o	automata-based (union-find)	i	d	on	sync	c	out	none	none	so	e	et	p	p	f
R2U2	none	p	s	tot	$\mathbb{N}$	all	d	automata-based	e	i	on	async	c	out	none	none	so	e	et	p	p	i
RiTHM	none	p	s	tot	none	f	o	time-triggered runtime verification	e	d	on	async	c	in	sw	none	so	s	all	p	p	i
RTC	ms	na	w	?	na	na	o	?	i	d	on	sync	c	in	sw	r	?	?	et	p	p	na
RV-Monitor	none	s	w	tot	$\mathbb{N}$	all	all	(see JavaMOP)	i	d	all	sync	c	all	sw	r	e	e	et	p	p	f
STePr	none	s	s	tot	$\mathbb{N}$	all	o	?	i	d	on	?	c	out	none	none	so	e	et	p	p	?
TemPsy/ OCLR-Check	none	p	v	tot	$\mathbb{N}$	all	d	OCL constraint	i	i	off	na	c	out	none	none	so	e	et	p	p	f
VALOUR	none	s	v	tot	$\mathbb{N}$	all	o	automata-based	i	d	on	all	c	in	swAJ	none	all	e	all	p	p	f