First International Competition on Software for Runtime Verification

Ezio Bartocci[1], Borzoo Bonakdarpour[2], and Yliès Falcone[3]

[1] Vienna University of Technology, Austria
ezio.bartocci@tuwien.ac.at
[2] McMaster University, Canada
borzoo@mcmaster.ca
[3] Université Grenoble-Alpes, Laboratoire d'Informatique de Grenoble, France
ylies.falcone@ujf-grenoble.fr

Abstract. We report on the process of organizing the First International Competition on Software for Runtime Verification (CSRV). The report describes the format, participating teams and evaluation process. The competition was held as a satellite event of the 14th International Conference on Runtime Verification (RV'14). The Competition was organized in three tracks: offline monitoring, online monitoring of C programs, and online monitoring of Java programs.

1 Introduction

Runtime Verification (RV) is a lightweight yet powerful formal specification-based technique for offline analysis (e.g., for testing) as well as runtime monitoring of software. 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, due to lack of standard benchmark suites as well as scientific evaluation methods to validate and test new techniques, we believe our community is in pressing need to have an organized venue whose goal is to provide mechanisms for comparing different aspects of existing tools and techniques.

For these reasons, inspired by the success of similar events in other areas of computer-aided verification (e.g., SV-COMP, SAT, SMT), we organized the First International Competition on Software for Runtime Verification (CSRV 2014) with the aim to foster the process of comparison and evaluation of software runtime verification tools. The aim of CSRV'14 was the following:

- 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.

B. Bonakdarpour and S.A. Smolka (Eds.): RV 2014, LNCS 8734, pp. 1–9, 2014.

- 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 software monitoring.

CSRV'14 was held in September 2014, in Toronto, Canada, as a satellite event of the 14th International conference on Runtime Verification (RV'14). The event was organized in three tracks: (1) offline monitoring, (2) online monitoring of C programs, and (3) online monitoring of Java programs. The competition included three phases for each track:

1. collection of benchmarks,
2. training and monitor submissions,
3. evaluation.

This report presents the procedures, rules, and participating teams of CSRV'14. The final results of the competition are planned to be announced during the RV'14 conference.

2 Format of the Competition

In this section we describe in detail the phases of the competition.

2.1 Declaration of Intent and Submission of Benchmarks and Specifications

The competition was announced in relevant mailing lists starting from October 2013. Potential participants were requested to declare their intent for participating in CSRV by December 15, 2013. For each of the three main tracks (offline, C and Java), the tools participating in the competition listed in alphabetical order in Tables 1, 2, and 3, respectively.

Subsequently, participants were asked to prepare benchmark/specification sets. These were collected in a shared repository[1]. The deadline was June 1st, 2014. The benchmarks were collected and classified into a hierarchy of folders representing the competition tracks and participating teams.

Online monitoring of Java and C programs tracks. In the case of Java and C tracks, each benchmark contribution was required to be structured as follows:

- *Program package* containing the program source code, a script to compile it, a script to run the executable, and an English description of the functionality of the program.
- *Specification package* is a collection of files, each containing a property that contains a formal representation of it, informal explanation and the expected verdict (the evaluation of the property on the program), instrumentation information, and an English description.

[1] `https://bitbucket.org/borzoob/csrv14`

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

Tool	Ref.	Contact person	Affiliation
RiTHM	[11]	B. Bonakdarpour	McMaster Univ. and U. Waterloo, Canada
E-ACSL	[7]	J. Signoles	CEA LIST, France
RTC		P. Pirkelbauer	University of Alabama at Birmingham, USA

Table 2. Tools participating in online monitoring of Java programs track

Tool	Ref.	Contact person	Affiliation
Larva	[4]	C. Colombo	University of Malta, Malta
jUnitRV	[5]	N. Decker	ISP, University of Lübeck, Germany
jUnitRV (MMT)	[6]	N. Decker	ISP, University of Lübeck, Germany
JavaMop	[10]	G. Rosu	U. of Illinois at Urbana Champaign, USA
prmj4	[12]	E. Bodden	TU Darmstadt, Germany
QEA	[1]	G. Reger	University of Manchester, UK

Table 3. Tools participating in the offline monitoring track

Tool	Ref.	Contact person	Affiliation
ZOT+SOLOIS	[3]	D. Bianculli	Politecnico di Milano, Italy
		S. Krstic	University of Luxembourg, Luxembourg
LogFire	[9]	K. Havelund	NASA JPL, USA
RiTHM2	[11]	B. Bonakdarpour	McMaster Univ. and U. Waterloo, Canada
MonPoly	[2]	E. Zalinescu	ETH Zurich, Switzerland
STePr		N. Decker	ISP, University of Lübeck, Germany
Breach	[8]	A. Donzé	University of California, Berkeley, USA
QEA	[1]	G. Reger	University of Manchester, England

The instrumentation information maps the events referred in the properties to concrete program events. A property consists of a formally defined object (e.g., an automaton, logical formula, etc), an informal description, and whether the program satisfies the property (i.e., the expected verdict). Instrumentation is a mapping from concrete events (in the program) to abstract events (in the specification). For instance, if one considers the HasNext property on iterators, the mapping should indicate that the `hasNext` event in the property refers to a call to the hasNext() method on an Iterator object. We allow for several concrete events to be associated to one abstract event.

Offline monitoring track. In the case of offline track, each benchmark contribution should consist of:

- a *trace* in either XML, CSV, or JSON format
- a *specification package*, which consists of a collection of files, each containing the formal representation of a property, informal explanation and the expected verdict (the evaluation of the property on the program), instrumentation information, and a brief English description.

Below we present some examples, where `an_event_name` ranges over the set of possible event names, `a_field_name` ranges over the set of possible field names, `a_value` ranges over the set of possible runtime values.

```
JSON format:
an_event_name
a_field_name = a_value
a_field_name = a_value

an_event_name
a_field_name = a_value
a_field_name = a_value

CSV format:
an_event_name, a_field_name = a_value, a_field_name = a_value
an_event_name, a_field_name = a_value, a_field_name = a_value

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>
 <event>
   <name>EVR</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>
```

2.2 Training Phase and Monitor Collection phase

After a sanity check of the benchmarks performed by the organisers, the training phase started on June 18, 2014. During this phase, all participants are supposed to train their tools with all the available benchmarks in the repository. This phase was scheduled to be completed by July 20, 2014, when the participants will submit the monitored versions of benchmarks. In this phase, a contribution consists of a the source of a program and a list of pairs of program and property identifier. That is, a contribution is related to a program and contains monitors for the properties of this program. Each monitor is related to one property. A monitor consists of two scripts, one for building the (monitored version of) program, one for running the monitored version of the program.

2.3 Benchmark Evaluation Phase

The competition experiments for evaluation will be performed on DataMill (`http://datamill.uwaterloo.ca`), a distributed infrastructure for computer performance experimentation targeted at scientists that are interested in performance evaluation. DataMill aims to allow the user to easily produce robust and reproducible results at low cost. DataMill executes experiments on real hardware and incorporates results from existing research on how to setup experiments and hidden factors.

Each participant will have the possibility to setup and try directly their tool using DataMill. The final evaluation will be performed by the competition organizers. In the next section, we present in detail the algorithm to calculate the final score for each tool.

3 Evaluation - Calculating Scores

Let us consider one of the three competition tracks (Java, C, and offline). Let N be the number of tools participating in the considered track and L be the total number of benchmarks provided by all teams. The total number of experiments for the track will be $N \times L$. Then, for each tool T_i $(1 \leq i \leq N)$ w.r.t. each benchmark B_j $(1 \leq j \leq L)$, we assign three different scores: the correctness score $C_{i,j}$, the overhead score $O_{i,j}$, and the memory utilization score $M_{i,j}$. In case of online monitoring, let E_j be the execution time of benchmark B_j (without monitor). Note, in the following, for simplicity of notation, we assume that all participants of a track want to compete on benchmark B_j. Participants can of course decide not to qualify on a benchmark of their track. In this case, the following score definitions can be adapted easily.

3.1 Correctness Score

The correctness score $C_{i,j}$ for a tool T_i running a benchmark B_j is calculated as follows:

- $C_{i,j} = 0$, if the property associated with benchmark B_j cannot be expressed in the specification language of T_i.
- $C_{i,j} = -10$, if the property can be expressed, but the monitored program crashes.
- $C_{i,j} = -5$, if, in case of online monitoring, the property can be expressed and no verdict is reported after $10 \times E_j$.
- $C_{i,j} = -5$, if, in case of offline monitoring, the property can be expressed, but the monitor crashes.
- $C_{i,j} = -5$, if the property can be expressed, the tool does not crash, and the verification verdict is incorrect.
- $C_{i,j} = 10$, if the tool does not crash, it allows to express the property of interest, and it provides the correct verification verdict.

Note that in case of a negative correctness score there is no evaluation w.r.t the overhead and memory utilization scores for the pair (T_i, B_j).

3.2 Overhead Score

The overhead score $O_{i,j}$ for a tool T_i running a benchmark B_j is related to the timing performance of the tool for detecting the (unique) verdict. For all benchmarks, a fixed total number of points O is allocated when evaluating the tools on a benchmark. Thus, the scoring method for overhead ensures that

$$\sum_{i=1}^{N} \sum_{j=1}^{L} O_{i,j} = O.$$

The overhead score is calculated as follows. First, we compute the *overhead index* $o_{i,j}$, for tool T_i running a benchmark B_j, where the larger overhead index, the better.

- In the case of offline monitoring, for the overhead, we consider the elapsed time till the property under scrutiny is either found to be satisfied or violated. If monitoring (with tool T_i) of the trace of benchmark B_j executes in time V_i, then we define the overhead as

$$o_{i,j} = \begin{cases} \frac{1}{V_i} & \text{if } C_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- In the case of online monitoring (C or Java), the overhead associated with monitoring is a measure of how much longer a program takes to execute due to runtime monitoring. If the monitored program (with monitor from tool T_i) executes in $V_{i,j}$ time units, we define the overhead index as

$$o_{i,j} = \begin{cases} \frac{\sqrt[N]{\prod_{l=1}^{N} V_{l,j}}}{V_{i,j}} & \text{if } C_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases}$$

In other words, the overhead index for tool T_i evaluated on benchmark B_j is the geometric mean of the overheads of the monitored programs with all tools over the overhead of the monitored program with tool T_i.

Then, the overhead score $O_{i,j}$ for a tool T_i w.r.t benchmark B_j is defined as follows:

$$O_{i,j} = O \times \frac{o_{i,j}}{\sum_{l=1}^{N} o_{l,j}}.$$

For each tool, the overhead score is a harmonization of the overhead index so that the sum of overhead scores is equal to O.

3.3 Memory Utilization Score

The memory utilization score $M_{i,j}$ is calculated similarly to the overhead score. For all benchmarks, a fixed total number of points O is allocated when evaluating the tools on a benchmark. Thus the scoring method for memory utilization ensures that

$$\sum_{i=1}^{N} \sum_{j=1}^{L} M_{i,j} = M.$$

First, we measure the memory utilization index $m_{i,j}$ for tool T_i running a benchmark B_j, where the larger memory utilization index, the better.

- In the case of offline monitoring, we consider the maximum memory allocated during the tool execution. If monitoring (with tool T_i) of the trace of benchmark B_j uses a quantity of memory D_i, then we define the overhead as

$$m_{i,j} = \begin{cases} \frac{1}{D_i} & \text{if } C_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases}$$

  That is, the memory utilization index for tool T_i evaluated on benchmark B_j is the geometric mean of the memory utilizations of the monitored programs with all tools over the memory utilization of the monitored program with tool T_i.
- In the case of online monitoring (C or Java tracks), memory utilization associated with monitoring is a measure of the extra memory the monitored program needs (due to runtime monitoring). If the monitored program uses D_i, we define the memory utilization as

$$m_{i,j} = \begin{cases} \frac{\sqrt[N]{\prod_{l=1}^{N} D_{l,j}}}{D_{i,j}} & \text{if } C_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Then, the memory utilization score $M_{i,j}$ for a tool T_i w.r.t. a benchmark B_j is defined as follows:

$$M_{i,j} = M \times \frac{m_{i,j}}{\sum_{l=1}^{N} m_{l,j}}.$$

3.4 Final Score

The final score F_i for tool T_i is then computed as follows:

$$F_i = \sum_{j=1}^{L} S_{i,j}$$

where:

$$S_{i,j} = \begin{cases} C_{i,j} & \text{if } C_{i,j} \leq 0, \\ C_{i,j} + O_{i,j} + M_{i,j} & \text{otherwise.} \end{cases}$$

4 Concluding Remarks

This report was written during the training phase. Once this phase is complete, the organizers will evaluate all the submitted monitors using the formula proposed in Section 3. The results of the competition is expected to be announced during the RV 2014 conference in Toronto, Canada. This report is published to assist future organizers of CSRV to build on the efforts made to organize CSRV 2014.

References

1. 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)
2. 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)
3. Bianculli, D., Ghezzi, C., San Pietro, P.: The Tale of SOLOIST: A Specification Language for Service Compositions Interactions. In: Păsăreanu, C.S., Salaün, G. (eds.) FACS 2012. LNCS, vol. 7684, pp. 55–72. Springer, Heidelberg (2013)
4. Colombo, C., Pace, G.J., Schneider, G.: 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 2009, pp. 33–37. IEEE Computer Society, Washington, DC (2009), http://dx.doi.org/10.1109/SEFM.2009.13
5. Decker, N., Leucker, M., Thoma, D.: jUnit$^{\text{RV}}$-adding runtime verification to junit. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 459–464. Springer, Heidelberg (2013)
6. Decker, N., Leucker, M., Thoma, D.: Monitoring Modulo Theories. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 341–356. Springer, Heidelberg (2014)
7. 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 (2013)

8. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-14295-6_17
9. Havelund, K.: Rule-based Runtime Verification Revisited. International Journal on Software Tools for Technology Transfer (STTT) (to appear, 2014)
10. 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, June 2-9, pp. 1427–1430. IEEE Press (2012)
11. 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: ACM Symposium on the Foundations of Software Engineering (FSE), pp. 603–606 (2013)
12. Parzonska, M.: A Library-Based Approach to Efficient Parametric Runtime Monitoring of Java Programs. Master's thesis, TU Darmstadt, Germany (2013)