On the Risk of Tool Over-tuning in Runtime Verification Competitions

Domenico Bianculli[1] and Srđan Krstić[2]

[1] SnT Centre - University of Luxembourg, Luxembourg
domenico.bianculli@uni.lu
[2] ETHZ, Zürich, Switzerland
srdan.krstic@inf.ethz.ch

Abstract

In this position paper we discuss the risk of tool over-tuning in runtime verification competitions. We believe that the risk is inherently enabled by the format of the competitions and we discuss it in the context of the "First International Competition on Software for Runtime Verification (CSRV 2014)", in which we participated.

1 Context

In 2014 we participated in the "First International Competition on Software for Runtime Verification (CSRV 2014)" [2, 3], held as satellite event of the 14th International Conference on Runtime Verification (RV'2014). The competition was organized in three tracks: offline monitoring, online monitoring of C programs, and online monitoring of Java programs. We participated in the offline track with the tool *ZOT+SOLOIST* [5, 4, 8], a trace checking tool for the SOLOIST [6] specification language.

The competition included three phases (for each track):

1. *Collection of benchmarks*, in which participants submitted their benchmarks, which were further collected in a shared repository. In the case of the offline track, a benchmark consisted of a *trace* and a *specification package*; the latter contained the formal representation of a property, the informal explanation and the expected verdict, instrumentation information, and a brief English description.

2. *Training (and monitor submission)*, in which participants trained their tools with all the benchmarks, before the actual tool submission.

3. *Evaluation*, in which tools were evaluated running the benchmarks.

To the best of our knowledge, this format has been adopted (with slight variations) also in the subsequent editions of the competition, namely CRV2015[1] [7] and CRV2016 [9]. The changes in the format mostly consisted of different times allocated for different phases. Specifically, the training phase, which is the central subject of this paper, was extended compared to the first edition of the competition.

[1] In 2015 the steering committee of the competition decided to change the name of the competition from CSRV (Competition on Software for Runtime Verification) to CRV (Competition on Runtime Verification).

2 The Risk of Tool Over-tuning

We contend that this format for the runtime verification competition (CRV) carries an intrinsic risk. During the training phase participants should "tune" their tools to correctly process the specific benchmarks provided by all the participants. A very näive "tuning" would be to hard-code the input/output pairs of each benchmark in the tool. Although this would be an extreme case of cheating, it could still happen in principle. In a less extreme way, participants might decide to add heuristics specific to the properties included in the benchmarks, instead of addressing the more general problem of supporting one more specification language within their tool.

Furthermore, there is a continuous range of possibilities between complete cheating and having a general tool that always works for *all* the types of properties supported by the other tools participating in the competition. Below we describe an example of what we consider a problematic situation.

Let us assume there are two tools, A and B, which support (before entering the competition), respectively, L_A and L_B as specification languages, with L_B being more expressive than L_A by supporting additional operators (say X, Y, Z). For the competition, the B team provides some benchmarks in which it uses a subset of the operators that are not supported by L_A, e.g., only X. During the tuning phase, team A might decide to add some support specific only to operator X or, even better (or worse), only for the specific property using operator X that is part of the benchmark. Team B has already developed the full "machinery" to support the entire language L_B. This full machinery might add some overhead that could impact the performance, even if the property to be checked does not exercise the full language (e.g., it does not contain operators Y and Z). If tool A gets a better score than tool B when evaluating the property that contains operator X, *does it really mean that it is "better" than tool B?*

3 What Can Be Done

We think that the CRV has a different nature than other similar competitions, like SMT-COMP [1] (Satisfiability Modulo Theories Competitions): SMTs have a well-defined, standardized input language (SMT-LIB), so they are all expected to support the same input. On the other hand, as noted also in [3], clearly CRV involves different specification languages, which can make comparisons meaningless.

We feel that the competition should put emphasis on more general problems (e.g., supporting certain class of properties, possibly expressed in different languages, and checking them efficiently), rather than focusing on checking a specific property on a specific trace. We want to make two suggestions to achieve this vision.

First, the structure of the benchmark should be revised. It should have two parts:

1) A *public* part should consist of the input language specification (syntax and semantics) and the trace format, optionally with examples of properties, traces, and corresponding verdicts.

2) A *private* part, known to the benchmark creator and the CRV organizers, should contain properties, traces and corresponding verdicts. In this part authors should include two groups of ⟨properties, traces, verdicts⟩:

   - A "competition" group containing the properties (and corresponding verdicts and traces) that will be used for assessing the performance and the correctness of the tools in the actual competition.

- An optional "coverage" group containing a set of properties (and corresponding verdicts and traces) that will be used for measuring the language coverage (and hence the expressiveness) of each tool participating in the CRV. Ideally there should be one property exercising each main construct/operator provided by the specification language. The "coverage" group is optional since the language coverage can be also measured using the "competition" group.

In the example presented above, the adoption of this format would mean that team B can provide a benchmark with the "competition" properties containing only expressions with operator X and the "coverage" properties exercising all the constructs of language L_A including the X, Y, Z operators.

The rationale for this change is that during the training phase potential participants would work on extending their tools to support the *class* of properties defined by the input language specification, instead of over-tuning their tool for the specific properties contained in the benchmark. We acknowledge that the proposed benchmark structure increases the participation overhead; however, we believe that more people will be motivated to participate if over-tuning is prevented by design.

The second suggestion is about the calculation of the final score for each tool. The current practice for computing the score in CRV [3] considers the overhead, the memory consumption, and the correctness. In particular, the latter takes into account whether a tool can express a property associated with a benchmark (i.e., a property from the "competition" group in our proposal). We advocate that the correctness score should only take into account the correctness of the verdicts for properties that are expressed. The number of language constructs that are supported by each tool (as determined by the "coverage" and the "competition" properties) can constitute a new *expressiveness* score. The more properties are supported and correctly verified, the higher the expressiveness score should be. A lower score would then reflect the participants' decision not to cover some features of a certain input language, which might translate into a simpler tool with better overhead and memory consumption.

Acknowledgment. This work has been partially supported by the National Research Fund, Luxembourg (FNR/P10/03) and by the Swiss National Science Foundation grant Big Data Monitoring (167162). The authors are listed alphabetically. The authors wish to thank Carlo Ghezzi, for his comments on an early draft of this paper, the chairs of CSRV 2014 (Ezio Bartocci, Borzoo Bonakdarpour, and Ylies Falcone) for the initial discussions on the topic of this paper, and the anonymous reviewers who provided helpful feedback.

References

[1] Clark Barrett, Morgan Deters, Leonardo Moura, Albert Oliveras, and Aaron Stump. 6 years of smt-comp. *J. Autom. Reason.*, 50(3):243–277, March 2013.

[2] Ezio Bartocci, Borzoo Bonakdarpour, and Yliès Falcone. First international competition on software for runtime verification. In *Proceeedings of the 2014 Runtime Verification Conference (RV 2014)*, volume 8734 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2014.

[3] Ezio Bartocci, Yliès Falcone, Borzoo Bonakdarpour, Christian Colombo, Normann Decker, Klaus Havelund, Yogi Joshi, Felix Klaedtke, Reed Milewicz, Giles Reger, Grigore Rosu, Julien Signoles, Daniel Thoma, Eugen Zalinescu, and Yi Zhang. First international competition on runtime verification: rules, benchmarks, tools, and final results of crv 2014. *International Journal on Software Tools for Technology Transfer*, Apr 2017.

[4] Marcello Maria Bersani, Domenico Bianculli, Carlo Ghezzi, Srđan Krstić, and Pierluigi San Pietro. SMT-based checking of SOLOIST over sparse traces. In *Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE 2014)*, volume 8411 of *Lecture Notes in Computer Science*, pages 276–290. Springer, April 2014.

[5] Domenico Bianculli, Carlo Ghezzi, Srđan Krstić, and Pierluigi San Pietro. Offline trace checking of quantitative properties of service-based applications. In *Proceedings of the 7h International Conference on Service Oriented Computing and Application (SOCA 2014)*, pages 9–16. IEEE, November 2014.

[6] Domenico Bianculli, Carlo Ghezzi, and Pierluigi San Pietro. The tale of SOLOIST: a specification language for service compositions interactions. In *Proceedings of the 9th International Symposium on Formal Aspects of Component Software (FACS'12)*, volume 7684 of *Lecture Notes in Computer Science*, pages 55–72. Springer, September 2012.

[7] Yliès Falcone, Dejan Ničković, Giles Reger, and Daniel Thoma. Second international competition on runtime verification. In *Proceedings of the 2015 International Conference on Runtime Verification (RV 2015)*, volume 9333 of *Lecture Notes in Computer Science*, pages 405–422. Springer, 2015.

[8] Matteo Pradella Marcello Maria Bersani, Srđan Krstić. Zot: a Bounded Satisfiability Checker. https://github.com/fm-polimi/zot, 2013.

[9] Giles Reger, Sylvain Hallé, and Yliès Falcone. Third international competition on runtime verification. In *Proceedings of the 2016 International Conference on Runtime Verification (RV 2016)*, volume 10012 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2016.