Int J Softw Tools Technol Transfer (2014) 16:457–464
DOI 10.1007/s10009-014-0337-y

INTRODUCTION

Rigorous examination of reactive systems

The RERS challenges 2012 and 2013

Falk Howar · Malte Isberner · Maik Merten · Bernhard Steffen · Dirk Beyer · Corina S. Păsăreanu

Published online: 3 August 2014

Abstract The goal of the RERS challenge is to evaluate the effectiveness of various verification and validation approaches on reactive systems, a class of systems that is highly relevant for industrial critical applications. The RERS challenge brings together researchers from different areas of software verification and validation, including static analysis, model checking, theorem proving, symbolic execution, and testing. The challenge provides a forum for experimental comparison of different techniques on specifically designed verification tasks. These benchmarks are automatically synthesized to exhibit chosen properties, and then enhanced to include dedicated dimensions of difficulty, such as conceptual complexity of the properties (e.g., reachability, safety, liveness), size of the reactive systems (a few hundred lines to millions of lines), and complexity of language features (arrays and pointer arithmetic). The STTT special section on RERS describes the results of the evaluations and the different analysis techniques that were used in the RERS challenges 2012 and 2013.

Keywords Program analysis · Model checking · Verification · Model-based testing · Competition · Reactive system · Event–condition–action system

An introduction to the RERS challenge 2012 was published in Proc. ISoLA'12 [26].

F. Howar · C. S. Păsăreanu
Carnegie Mellon Silicon Valley/NASA Ames, Mountain View, USA

M. Isberner (✉) · M. Merten · B. Steffen
TU Dortmund, Dortmund, Germany
e-mail: malte.isberner@cs.uni-dortmund.de

D. Beyer
University of Passau, Passau, Germany

1 Introduction

Reactive systems are ubiquitous: for example, reactive systems appear as web services, as decision-support systems, and as logical controllers. Especially, event–condition–action (ECA) systems are omnipresent in industrial practice. Notable applications include programmable logic controllers (PLCs) [1], active databases [31], and web-service composition [4]. Moreover, ECA systems are the basis of the increasingly popular rule-based systems [24], which can be regarded as de-facto standard for dealing with permissions and access control, and ECA systems are promoted as a means for realizing compliant business processes on top of rule engines like Drools [14] or JRules [13].

Validation techniques for reactive systems are as diverse as their appearance and structure. The used techniques comprise various forms of program analysis [34], symbolic execution [28], software model checking [7,16,25], statistical model checking [11], model-based testing [15], inference of invariants [8,18,21,22], automata learning [2,37], run-time verification [29], and monitoring [23], often tailored to rather special environment assumptions. Thus, it is almost impossible to compare these techniques in a common setting, let alone to establish clear application profiles as a means for recommendation.

The RERS [1] challenge aims to overcome this situation by providing a forum for experimental profile evaluation based on specifically designed verification tasks. These benchmarks are automatically synthesized to exhibit chosen prop-

[1] The name RERS originally was an acronym for regular extrapolation of reactive systems. Although the acronym remained the same, the challenge itself has evolved towards a broader focus, addressing a variety of techniques for analyzing and inferring the behavior of reactive systems, leading to a change of the name and scope to rigorous examination of reactive systems.

erties, and then enhanced to include dedicated dimensions of difficulty, ranging from conceptual complexity of the properties (e.g., reachability, safety, liveness), over size of the reactive systems (a few hundred lines to millions of lines), to exploited language features (from mere assignments to pointer arithmetics). So far, the RERS challenges focused on functional properties only, but non-functional properties like time, performance, and stochastic behavior are envisaged for the future.

RERS started with an initial workshop that was held at ISoLA 2010 and continued with two challenges on the analysis of generated verification tasks in 2012 and 2013. A third instantiation is planned for ISoLA 2014. The first RERS challenge in 2012[2] concluded with a workshop that was co-located with ISoLA 2012. The RERS challenge in 2013[3] concluded with a workshop at ASE 2013.

The STTT special section on RERS focuses on the results from organizing the challenges as well as from participating in the challenges. It consists of two papers describing the generation of benchmarks and five contributions covering some approaches to the challenges in 2012 and 2013. The remainder of the introduction will give a brief overview of the challenge setup and results for both past editions, and discusses what sets RERS apart from other program analysis competitions.

2 Analysis of reactive systems

RERS focuses on the analysis of event–condition–action (ECA) systems, a particular class of reactive systems. A verification task in the challenge consists of a generated ECA program and a reachability or LTL property. In the following, we provide a brief description of the ECA programs and of the properties that we use in the verification tasks. More details and the actual verification tasks can be found on the RERS web page.[4]

2.1 ECA programs

At an abstract level, reactive systems can be seen as open programs that read inputs and produce outputs. In event–condition–action (ECA) systems, transitions for (input) events are guarded by conditions, perform actions on the internal state of the system, and produce outputs. This concept is implemented in many different languages and frameworks. To have systems that verification tools can be applied to, the RERS challenge uses ECA programs from which all

```
int x1 = 1;
int x2 = 2;
...

int calculate_output(int input) {
  if (input == 3 &&               // Event
     (x7 != 0 && x0 == 9)) {  // Condition
    x9 = 4;                           // Action
    return 24;
  }
  if ((x23 == 0 && x3 != 0 && x26 == 3)) {
    error23: assert(0);         // Error
  }
  ...
}

int main() {
  while (1) { // main i/o-loop
    int input, output;
    scanf("%d", &input);
    output = calculate_output(input);
    if (output == -2) {
      fprintf(stderr, "Invalid␣input:␣%d\n");
    } else if (output != -1) {
      printf("%d\n", output);
    }
  }
}
```

Fig. 1 Source code fragment of a challenge program

unnecessary (for the purpose of the challenge) implementation details are omitted: ECA programs consist of a main method with a `while(true)` loop, in which an input is read and passed to a method that calculates the updates on internal states and the outputs. In the most basic form, the programs use only primitive integer variables and no other data types.

Figure 1 shows a snippet of source code of the C version of an ECA program. The ECA logic is contained in a method called `calculate_output`, which is a sequence of (nested) `if-then-else` blocks. The state of the ECA system is represented by a set of variables. At the bottom of this function, a sequence of `if` statements checks whether the system is in an invalid state. If this is the case, an error is raised by a failed assertion. To identify the specific error in the source code, the assertion is labeled with the error ID.

2.2 Properties

The properties to be verified in the RERS challenge fall into two categories.

Reachability properties Some value assignments to internal state variables correspond to error states, which cause the system to fail with a specific error code. Not all of those error states are reachable, and the goal is to check which of these error states can in fact be reached (it is not expected to also provide a sequence of inputs for reaching

[2] http://rers-challenge.org/2012/.

[3] http://rers-challenge.org/2013ase/.

[4] http://www.rers-challenge.org/.

the error state). Those errors come in the form of either an `IllegalStateException` (Java) or a specific error label with a failed assertion (C). Each individual such reachability problem is evaluated and ranked exactly in the same fashion as the LTL properties. The label `error23` in Fig. 1 is an example for this class of properties.

LTL properties An execution trace of an ECA system consists of a sequence of inputs and outputs, each from a finite alphabet. For each of the systems, a file is provided, containing a set of 100 LTL properties for which the contestants have to check whether they are satisfied by all traces, or if there are traces that violate them (it is not expected to also provide these traces as witness). The properties are given both as an LTL formula and as a textual description. To allow an intuitive mapping from LTL expressions to textual descriptions, the properties to be checked are closely adhering to the patterns in property specifications from the literature [20]. The LTL formulas are given in a standard syntax, based on the following temporal operators:

- $\mathbf{X}\phi$ (next): ϕ has to hold after the next step,
- $\mathbf{F}\phi$ (eventually): ϕ has to hold at some point in the future (or now),
- $\mathbf{G}\phi$ (globally): ϕ has to hold always (including now),
- $\phi\mathbf{U}\psi$ (until): ϕ has to hold until ψ holds (which eventually occurs),
- $\phi\mathbf{WU}\psi$ (weak until): ϕ has to hold until ψ holds (which does not necessarily occur), and
- $\phi\mathbf{R}\psi$ (release): ϕ has to hold until ψ held in the previous step.

As usual, the boolean operators & (conjunction), | (disjunction) and ! (negation) are used. The atomic propositions correspond to input and output symbols, where the prefix `i` is used for input and `o` is used for output symbols, to allow a clear distinction.[5]

For example, `G (! oU)` means that output `U` never occurs. In other words, the expression states that it is not possible—by any sequence of input events—to make the system produce an output action `U`.

2.3 Generating ECA programs and properties

All problems are generated in C and Java. Labels in the code are used to encode reachability properties, more complex properties are written as LTL formulas over inputs and outputs. The problem instances are scaled in several dimensions: size of internal state, number of inputs, number of abstract program states, and used data types and language constructs.

[5] The more common prefixes `?` and `!` for inputs and outputs, respectively, cause confusion with the unary negation operator `!`.

The generation of problems and properties was automated and is based on generating automata for properties. These automata are then combined to abstract systems, which in a second step are further complicated by transformations introducing state variables and guards (cf. [38]). How this approach can be conceptually extended to the generation of concurrent benchmarks is discussed in [36].

3 RERS challenge 2012

The RERS grey-box challenge for ISoLA 2012 proceeded in two parts:

- An offline part, where the contestants had two months to analyze all verification tasks and to carefully prepare their results, and
- An online part during ISoLA 2012, where the contestants had to obtain the results between the opening on Sunday, October 14, 2012 and the presentation session on Thursday, October 18, 2012 in the morning.

We used the term “grey-box” in 2012 to refer to the situation that the “usual” control flow of the program is not visible and usable in ECA programs (it is “obfuscated” in the ECA structure).

Springer sponsored a €500 gift certificate for Springer books for the best solutions. The teams with the best solutions in their categories summarize the results of the challenge and present their solution approaches in separate articles [10,32,35,40], respectively.

3.1 Challenge setup and rules

The contestants were confronted with a number of ECA programs given in both C and Java, ranging from structurally simple and small to structurally complex and large, as well as corresponding collections of properties. The contestants had to check, for each ECA program, if each property for the program holds; there were 60 reachability and 100 behavioral properties for each program.

The challenge started with six ECA programs of varying complexity. After an initial phase of four weeks, further programs of higher complexity were added (two of them for self-evaluation purposes only), specifically tailored to differentiate the technologies of the participating contestants. In contrast to the first six programs, the new programs contained arithmetics and linear inequalities.

Finally, during the online phase, six additional programs were added, which were similar to those added in the second round. For these verification tasks, the number of LTL properties to be checked was reduced to 50, while the number of reachability properties remained at 60. In all problems, the

Table 1 Results of the 2012 RERS challenge

Team	Overall				Reachability				LTL			
	Score	Answers Tot. (#)	Err. (#)	Corr. (%)	Score	Answers Tot. (#)	Err. (#)	Corr.	Score	Answers Tot. (#)	Err. (#)	Corr. (%)
Twente	**25,190**	**2,437**	52	97.9	**10,978**	**1,037**	28	97.3	**14,212**	**1,400**	24	98.3
Paris[a]	13,104	1,486	10	99.3	6,039	671	**0**	**100.0**	7,065	815	**10**	**98.8**
Vienna	8,433	938	36	96.2	4,635	416	**0**	**100.0**	3,798	522	36	93.1
Passau[a]	5,382	598	**0**	**100.0**	5,382	598	**0**	**100.0**	–	–	–	–
South.[a]	5,061	705	16	97.7	2,070	305	1	99.7	2,991	400	15	96.3

Best value in respective column are in bold
[a] Offline participation only

number of violated properties varied between about 25 and 75 %.

The challenge was free-style: the contestants were allowed to patch the programs in any way, but the validity had, of course, to be stated according to the original programs. Results were computed by the contestants on their machines and only the final results were submitted for evaluation. Solutions were evaluated in two categories:

1. A purely numeric ranking, according to the amount of correct answers that the contestants provided, as in a true competition. To express confidence in the verification results, the contestants had to assign to each answer a confidence weight from 0 to 9. For each correct answer, the weight value was added to the overall score of the contestant. For each wrong answer, twice the weight value was subtracted (cf. [5]).
2. A conceptual ranking, according to the employed (combination of) methods was used to emphasize the challenge aspect. In this category, solutions were reviewed and ranked by the challenge team. Due to the possible variety of methods, there could have been several winners in this category.

For the second category, the teams were asked to write a short summary of the chosen approach, the encountered hurdles, the solutions, and the obtained results.

3.2 Results

The results of the five teams that participated in the RERS challenge 2012 are shown in Table 1.[6] The table shows both the overall results as well as the results when taking into consideration only reachability or LTL properties, respectively.

In the score-based ranking, the solution by the team from Twente University, based on explicit-state, multi-core model checking using LTSmin [12,40], emerged as the clear winner in the competition. However, two of the teams did not participate in the online part of the challenge, and thus had a principal disadvantage. If counting only the offline phase, the winning margin shrinks from almost 100 % to a mere 17 %. According to the obtained score, the second-best solution was obtained with the Frama-C program analysis framework [19]. It is followed by the solution by the team from Vienna, using the *CodeThorn* program analysis and verification tool [35] based on the ROSE [30] compiler. The team from Passau used a BDD-based approach to symbolic model checking [9,10], and the team from Southampton used ESBMC [32,33], an SMT-based bounded model checker.

The variations in the score-based ranking are partially caused by the different numbers of questions answered, and by the different precisions that the verifiers claimed. While all teams gave a relatively low percentage of incorrect answers, only the team from Passau provided 100 % correct solutions, i.e., if an answer was provided, then the answer was correct. The second-best solution according to the precision of the results (correctness) was the Frama-C-based solution.

The conceptual award ("Method combination prize") was given to the team from Vienna. Instead of relying on static analysis techniques only, as was the case for all other participants, they additionally employed dynamic, black-box testing to validate their answers and to complement shortcomings of their approach.

4 RERS challenge 2013

A goal for the 2013 challenge was to investigate potential for synergy between source code-based (white-box) approaches and purely testing-based (black-box) approaches: Black-box-based testing approaches are independent of language features, but they can only prove the presence of errors, in general not their absence. Formal source code analysis has the power to prove the absence of errors, but adding support for a new language feature may require an enormous effort.

[6] An overview of the team members, as well as more detailed statistics can be found at http://rers-challenge.org/2012/index.php?page=results.

Realistic problems often require both approaches, which are so far mostly applied in isolation.

In this spirit, the 2013 instantiation of RERS offered challenge problems that also scaled in the dimension of new language features such as arrays: the programs also contained guards and assignments with complex arithmetics as well as array operations. The RERS 2013 challenge was held offline completely, to have more time for discussions during the workshop. As in 2012, Springer sponsored gift certificates (€ 1,250 in total) for the winners.

4.1 Challenge setup and rules

The RERS challenge 2013 offered considerably more verification tasks, compared to 2012. Alone in the white-box category with reachability properties, there were 46 programs and 60 reachability properties, composing a total of 2,760 verification tasks. The set of all verification tasks was partitioned into three categories:

White-box As in 2012, some problems were offered as source code, targeting static analyses. A detailed characterization of all programs in category white-box can be found in a separate article (cf. Table 1 in [10]).

Black-box. A set of verification tasks was offered as executable binaries, targeting dynamic approaches. These binaries were compiled from C source code, with heavy optimizations enabled. The task for the contestants was to observe these programs' behavior by interaction via standard I/O, even though in principle static binary-analysis techniques could have been applied as well.

Grey-box The grey-box category presents itself as a mixture of the two above: the main part of the system is distributed as source code. However, the code contains calls to functions that are not provided as source code. The source code itself is not compilable, due to these missing function references. To complement the missing information, the linked executable binary of the whole program was, therefore, provided.

To not impose a preference towards a particular approach or technology, the results are reported as boolean answers to more than 12,000 queries on input/output reachability and LTL properties. The contestants were ranked in three evaluations.

1. For the numeric ranking, there were four rankings: white-box, black-box, grey-box, and overall. The scores were evaluated according to the number of correctly answered queries. Each correct answer yields one point, and each wrong answer costs two points (correct = +1, wrong = −2, and unknown = 0). The "confidence" weight from 2012 was dropped.
2. Achievements were assigned to emphasize a verifiers perspective: all given answers had to be correct, but it is not required to give an answer for every verification task. Moreover, "achievements" did only concern the easiest verification tasks of each category.
3. The employed (combination of) methods was evaluated by the organizing team. Due to the potential variety of methods, several winners were allowed in this evaluation (method combination award).

We restricted the achievement-specific evaluation as follows, to take the specific character of grey-box and black-box problems into account.

White-box The evaluation of the reachability and LTL problems is independent from the categories grey-box and black-box, and follows the same scheme (incorrect answers forbidden): Bronze requires a 70 % score on one problem in sub-category "small/easy". Silver requires a 75 % score on one problem in sub-category "medium/easy". Gold requires an 80 % score on one problem in sub-category "large/easy". Platinum requires that all problems are solved correctly.

Grey-box and black-box In the categories grey-box and black-box, we only consider the sub-set of verification tasks for which the program violates the property, i.e., the result can be demonstrated explicitly via an appropriate error path as witness (i.e., reachable error labels and violated LTL properties). These violations can be observed via testing. Besides this restriction, the assignment of Bronze, Silver, Gold, and Platinum follows the same schema as for the category white-box.

While the achievements evaluation was new for RERS 2013, the other two evaluations essentially worked as for RERS 2012, except that no confidence levels apply and there are now a total of three categories: white-box, black-box, and grey-box.

4.2 Results

In 2013, two teams competed in the final challenge: Markus Schordan and Adrian Prantl from Lawrence Livermore National Laboratory (LNLL), USA (the Vienna team of 2012), and Jaco van de Pol (also participated 2012) and Theo Ruys from Twente University. The tools that the two teams used are the same as in 2012, namely, CodeThorn and LTSmin. Their approaches are described in separate articles [35,40], respectively.

Table 2 Results of the 2013 RERS challenge

Team	Overall				Reachability				LTL			
	Score	Answers Tot. (#)	*Err*.(#)	Corr. (%)	Score	Answers Tot. (#)	Err. (#)	Corr. (%)	Score	Answers Tot. (#)	Err. (#)	Corr. (%)
Twente-Rank	**7,317**	**7,365**	16	99.78	**3,144**	**3,180**	12	99.62	4,173	4,185	4	**99.90**
Dortmund[a]	6,518	7,040	174	97.53	2,184	2,640	152	94.42	**4,334**	**4,400**	22	99.50
Twente-Achiev	5,801	5,813	4	**99.93**	1,628	1,628	**0**	**100.0**	4,173	4,185	4	**99.90**
LNLL-Achiev	341	344	**1**	99.71	341	344	1	99.71	–	–	–	–
LNLL-Rank	279	282	**1**	99.65	277	277	**0**	**100.0**	2	5	**1**	80.00

Best value in respective column are in bold
[a] No official participation

Due to the small number of participants, they were allowed to submit two solutions: one aimed at achieving a high overall score (Rank), which was considered for the main challenge ranking, and one aimed at obtaining a high number of achievements (Achiev), i.e., cautiously avoiding incorrect answers. This can be seen as a ”qualitative” equivalent to the confidence weights from 2012.

In addition, a third team from TU Dortmund contributed a solution based on active automata learning to the challenge. Due to the black-box nature of active automata learning, this team was the only team that approached the black-box problems. However, because there was a partial overlap with the challenge organizers, this team did not participate officially and was not eligible for winning one of the prizes. That said, their solution—described separately [3]—exploited no ”secret” knowledge about the verification tasks.

The overall (i.e., combining all three categories) results of the 2013 challenge are shown in Table 2. Once again, the team from Twente University came first. The automata learning-based solution scored well, but produced by far the most incorrect answers. It should be noted that the Achiev submissions were due one month after the deadline for the Rank submissions. This explains the higher score of the former for the LNLL team. Results differentiated by categories can be found on the RERS 2013 website.[7] Both (external) teams submitted detailed and thorough descriptions of their approaches, and the Method Combination Award was given to both teams.

5 Relation to other challenges

Competition and challenge events are well-understood in the verification community as an effective means for technology evaluation and exchange, for revealing the state of the art in a tangible fashion, and to stimulate robust tool implementations. Notable examples range over various fields, for example, automatic software verification [7],[8] interactive software verification [27],[9] termination checking,[10] hardware model checking,[11] SAT solving,[12] SMT solving [17],[13] QBF solving,[14] theorem proving [39],[15] and planning.[16] All of those events impact the development pace and quality of the competing software tools; results from theory are almost instantly transferred to practical tool implementations.

Of the mentioned events, the competition on software verification (SV-COMP) [5–7] at TACAS is thematically closest to the RERS grey-box challenge [26], even though it is complementary in the following aspects.

The RERS challenge is characterized by

(a) A strong focus on tailored program patterns (e.g., generated ECA programs with restricted data structures),
(b) Complex properties (reachability and LTL),
(c) Unrestricted computational means, in terms of both hardware and software, and
(d) Possible interaction of the participants with their verification tools.

In contrast, the SV-COMP competition focuses on

(a) A wide range of program structures (including industrial code, like Linux and Windows device drivers),
(b) Specific properties from reachability, memory safety, and termination,
(c) Fully controlled evaluation, which is done on a dedicated experimental environment (specific computer platform

[7] http://rers-challenge.org/2013ase/index.php?page=results.

[8] http://sv-comp.sosy-lab.org.
[9] http://www.verifythis.org.
[10] http://termination-portal.org.
[11] http://fmv.jku.at/hwmcc.
[12] http://www.satcompetition.org.
[13] http://www.smtcomp.org.
[14] http://www.qbflib.org/competition.html.
[15] http://www.cs.miami.edu/~tptp/CASC.
[16] http://ipc.icaps-conference.org/.

under defined environmental conditions and limited time and memory resources), and

(d) Fully automatic verification (no interaction of participants with their verifiers, which run in isolation on a platform to which the participants have no access).

This difference characterizes SV-COMP as a competition with a clear ranking, and contrasts RERS as a challenge, whose setup makes it difficult to define a global ranking. This is why we have several rankings for different purposes, for example, several are purely numerical, simply based on a 'multiple choice' test which may be solved 'free-style', and one considers the approach taken, the underlying ideas, and the concrete realization are evaluated by the challenge team (with similarities to the competition on interactive software verification [27]).

6 Conclusion and future plans

The RERS grey-box challenge for ISoLA 2012 [26] was the first of a series of events in which we aim at successively refining the challenge scenario to specifically discuss current strengths and limitations, and to exchange implementations, algorithms, ideas, and visions. This was continued in 2013 with RERS being a satellite event of ASE 2013, this time focusing on the distinction between white-box, grey-box and black-box problems.

The generated problems turned out to be very hard and complex, in particular, also concerning the mere size of the benchmark programs, which scared a number of potential competitors away.

The third RERS challenge, which is part of ISoLA's 10th anniversary edition in 2014, specifically addresses the concerns raised in 2013 to attract more competitors. In addition, it comprises a new trail category with concurrent problems. This category has an enormous potential and many facets. Thus, one of the main discussion topics during RERS 2014 will be how to leverage this potential in the future.

Acknowledgments We would like to thank Rustan Leino and Jaco van de Pol for their helpful comments, and Maren Geske for her assistance in implementing the challenge infrastructure.

References

1. Almeida, E.E., Luntz, J.E., Tilbury, D.M.: Event-condition–action systems for reconfigurable logic control. IEEE Trans. Autom. Sci. Eng. **4**(2), 167–181 (2007)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
3. Bauer, O., Geske, M., Isberner, M.: Analyzing program behavior through active automata learning. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-014-0333-2 (2014)
4. Benatallah, B., Sheng, Q.Z., Dumas, M.: The Self–Serv environment for web-services composition. Internet Comput. IEEE **7**(1), 40–48 (2003)
5. Beyer, D.: Competition on software verification (SV-COMP). In: Proceedings of TACAS, LNCS 7214, pp. 504–524. Springer (2012)
6. Beyer, D.: Second competition on software verification. In: Proceedings od TACAS, LNCS 7795, pp. 594–609. Springer (2013)
7. Beyer, D.: Status report on software verification. In: Proceedings of TACAS, LNCS 8413, pp. 373–388. Springer (2014)
8. Beyer, D., Henzinger, T. A., Majumdar, R., Rybalchenko, A.: Path invariants. In: Proceedings of PLDI, pp. 300–309. ACM (2007)
9. Beyer, D., Stahlbauer, A.: BDD-based software model checking with CPAchecker. In: Proceedings of MEMICS, LNCS 7721, pp. 1–11. Springer (2013)
10. Beyer, D., Stahlbauer, A.: BDD-based software verification: applications to event-condition–action systems. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-014-0334-1 (2014)
11. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Proceedings of FSTTCS, LNCS 1026, pp. 499–513. Springer (1995)
12. Blom, S.C.C., van de Pol, J.C., Weber, L.T., Smin, M.: Distributed and symbolic reachability. In: Proceedings of CAV, LNCS 6174, pp. 354–359. Springer (2010)
13. Boyer, J., Mili, H.: IBM WebSphere ILOG JRules. In: Agile Business Rule Development, pp. 215–242. Springer (2011)
14. Browne, P.: JBoss Drools Business Rules: Capture, Automate, and Reuse Your Business Processes in a Clear English Language that Your Computer Can Understand. Packt Publishing (2009)
15. Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (editors): Model-based testing of reactive systems. In: LNCS 3472. Springer (2005)
16. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge, USA (2001)
17. Cok, D. R., Griggio, A., Bruttomesso, R., Deters, M.: The 2012 SMT competition. In: Proceedings of SMT, pp. 131–142 (2012)
18. Colón, M., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using non-linear constraint solving. In: Proceedings of CAV, LNCS 2725, pp. 420–432. Springer (2003)
19. Cuoq, P., Signoles, J., Baudin, P., Bonichon, R., Canet, G., Correnson, L., Monate, B., Prevosto, V., Puccetti, A.: Experience report: OCaml for an industrial-strength static analysis framework. In: Proceedings of ICFP, pp. 281–286. ACM (2009)
20. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of ICSE, pp. 411–420. ACM (1999)
21. Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D.: Dynamically discovering likely program invariants to support program evolution. IEEE Trans. Softw. Eng. **27**(2), 99–123 (2001)
22. Gulwani, S., Srivastava, S., Venkatesan, R.: Constraint-based invariant inference over predicate abstraction. In: Proceedings of VMCAI, pp. 120–135 (2009)
23. Havelund, K., Roşu, G.: Monitoring Java programs with Java PathExplorer. ENTCS **55**(2), 200–217 (2001)
24. Hayes-Roth, F.: Rule-based systems. Commun. ACM **28**(9), 921–932 (1985)
25. Holzmann, G.J., Smith, M.H.: Software model checking: extracting verification models from source code. Softw. Test. Verif. Reliab. **11**(2), 65–79 (2001)
26. Howar, F., Isberner, M., Merten, M., Steffen, B., and Beyer, D.: The RERS grey-box challenge 2012: analysis of event-condition-action systems. In: Proceedings of ISoLA, LNCS 7609, pp. 608–614. Springer (2012)
27. Huisman, M., Klebanov, V., Monahan, R.: On the organisation of program-verification competitions. In: Proceedings of COMPARE, CEUR Workshop Proceedings 873, pp. 50–59. CEUR-WS.org (2012)

28. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976)
29. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Logic Alg. Progr. **78**(5), 293–303 (2009)
30. Lidman, J., Quinlan, D.J., Liao, C., McKee, S.A.: ROSE:FTTransform—a source-to-source translation framework for exascale fault-tolerance research. In: Proceedings of FTXS. IEEE (2012)
31. McCarthy, D., Dayal, U.: The architecture of an active database management system. In: Proceedings of ICMD, pp. 215–224. ACM (1989)
32. Morse, J., Cordeiro, L., Nicole, D., Fischer, B.: Context-bounded model checking of LTL properties for ANSI-C software. In: Proceedings of SEFM, LNCS 7041, pp. 302–317. Springer (2011)
33. Morse, J., Cordeiro, L., Nicole, D., Fischer, B.: Applying symbolic bounded model checking to the: RERS greybox challenge, p. 2014. J. Softw. Tools Technol. Transf. Int. doi:10.1007/s10009-014-0335-0 (2014)
34. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, New York, USA (1999)
35. Schordan, M., Prantl, A.: Combining static analysis and state transition graphs for verification of event-condition-action systems in the RERS 2012 and 2013 challenges. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-014-0338-x (2014)
36. Steffen, B., Howar, F., Isberner, M., Naujokat, S., Margaria, T.: Tailored generation of concurrent benchmarks. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-014-0339-9 (2014)
37. Steffen, B., Howar, F., Merten, M.: Introduction to active automata learning from a practical perspective. In: Proceedings of SFM, LNCS 6659, pp. 256–296. Springer (2011)
38. Steffen, B., Isberner, M., Naujokat, S., Margaria, T., Geske, M.: Property-driven benchmark generation: synthesizing programs of realistic structure. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-014-0336-z (2014)
39. Sutcliffe, G., Suttner, C.: The state of CASC. AI Commun. **19**(1), 35–48 (2006)
40. van de Pol, J., Ruys, T. C., te Brinke, S.: Thoughtful brute force attack of the RERS 2012 and 2013 challenges. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-014-0324-3 (2014)