The RERS Grey-Box Challenge 2012: Analysis of Event-Condition-Action Systems

Falk Howar[1], Malte Isberner[2], Maik Merten[2], Bernhard Steffen[2], and Dirk Beyer[3]

[1] Carnegie Mellon University, Mountain View, USA
[2] TU Dortmund, Germany
[3] University of Passau, Germany

Abstract. The goal of the RERS Grey-Box Challenge is to evaluate the effectiveness of various verification and validation approaches on Event-Condition-Action (ECA) systems, which form a specific class of systems that are important for industrial applications. We would like to bring together researchers from all areas of software verification and validation, including theorem proving, model checking, program analysis, symbolic execution, and testing, and discuss the specific strengths and weaknesses of the different technologies.

Keywords: Program Analysis, Model Checking, Verification, Model-Based Testing, Competition, Event-Condition-Action System.

1 Motivation

Event-Condition-Action (ECA) Systems are omnipresent in industrial practice. Notable applications include programmable logic controllers (PLCs) [1], active databases [20], and web-service composition [4]. Moreover, they are the basis of the increasingly popular rule-based systems [16], which can be regarded as de-facto standard for dealing with permissions and access control, and ECAs are promoted as a means for realizing compliant business processes on top of rule engines like Drools [9] or JRules [8].

The popularity of this rule architecture comes from its apparent simplicity: one can add and change the functionality simply by adding and removing rules. However, this simplicity has its price: it is extremely difficult to understand and control the global implications of these apparently simple changes. It is almost impossible to manually find out if there are side-effects, whether the new rule is executed at all, if the whole rule system behaves deterministically, or if the system does actually terminate—to name only a few problems.

Modern verification techniques make it possible to automatically answer many of those questions. However, treating ECA systems is challenging for almost all verification and validation approaches, because there is little control structure to hook on to. The inherent structure of ECA systems (causalities, conflicts, dependencies, etc.) needs therefore to be inferred from their data-flow alone.

To obtain an overview over the various techniques, and to compare the techniques on a common set of problems, we have set up a grey-box challenge.

The RERS[1] Grey-Box Challenge at ISoLA 2012 proceeds in two parts:

– an offline part, where the contestants have two months to analyze all benchmark systems and to carefully prepare their results, and
– an online part during ISoLA, where the contestants have to prepare their results between the opening on Sunday, October 14th, 2012 and the presentation session on Thursday, October 18th, 2012, in the morning.

Everybody who is interested in the verification of ECA systems is invited to apply his/her techniques to the ECA setting. The aim is to reveal, compare, and combine the specific strengths of the various verification techniques, be they manual, tool-supported, or fully automated, for treating this peculiar but nevertheless practically highly important kind of systems.

Springer sponsors a 500 Euro gift certificate for Springer books for the best solutions, and the teams with the best solutions in their categories will be invited for an STTT Special Section summarizing the results of the challenge, and, in particular, presenting the most advanced solutions.

The RERS Grey Box Challenge at ISoLA 2012 is the first of a series of events in which we aim at successively refining the 2012 challenge scenario in order to specifically discuss current strengths and limitations, and to exchange implementations, algorithms, ideas, and visions. In particular, during the challenge meeting at ISoLA 2012 it is planned to discuss the profile of the 2013 challenge held in fall in Mountain View as a satellite of ASE 2013. The third RERS challenge is planned to be part of ISoLA's 10th anniversary in 2014.

2 Characteristics of the Challenge

The challenge is very special. On the one hand, it is fully 'white-box'—the full Java/C code is available. On the other hand, it has a black-box character—ECA code is particularly unstructured, not easy to analyze.

It will therefore be interesting to see how well, e.g., program-analysis techniques and model checking, perform in comparison with black-box techniques like model-based testing, and how these techniques may be profitably combined. We are therefore particularly looking forward to contributions based on tools that comprise one or the combination of many of the following technologies:

– program analysis and verification [22],
– symbolic execution [18],
– software model checking [12, 17],

[1] The name RERS originally was an acronym for *R*egular *E*xtrapolation of *R*eactive *S*ystems. Although the name 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.

- statistical model checking [6],
- model-based testing [10],
- inference of invariants [14],
- automata learning [2, 23],
- run-time verification [19], and
- monitoring [15].

Of course, this list is not meant to be exhaustive. Rather we want to encourage everybody to approach this challenge with all the available means and ideas, and people are welcome to join effort and to approach the problem in heterogeneous teams.

3 Challenge Setup and Rules

Contestants are confronted with a number of ECA systems given in both Java and C, ranging from structurally simple and small to structurally complex and large, as well as corresponding collections of properties to be checked against these systems, which fall into two categories:

Reachability Properties: Some assignments to internal state variables correspond to erroneous 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 states can in fact be reached (it is not expected to also provide a sequence of inputs reaching them). Those errors come in the form of either an `IllegalStateException` (Java) or a failed assertion (C), along with a specific error label. Each individual such reachability problem is evaluated and ranked exactly in the same fashion as the behavioral properties.

Behavioral Properties: An execution trace of the ECA system consists of a sequence of inputs and outputs, each from a finite alphabet. For each of the systems, a file `properties.txt` is provided, containing a set of 100 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). The properties are given both as an LTL formula and a textual description. For example, `(G ! oU)` means that output `U` does never occur. 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`.

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 identified by Dwyer et al. [13]

In LTL formulas, 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.[2]

The LTL formulas are given in a standard syntax, making use of the following temporal operators:

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

- $\mathbf{X}\phi$ (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)
- $\phi\mathbf{R}\psi$ (release): ϕ has to hold until ψ held in the previous step.

Additionally, the boolean operators & (conjunction), | (disjunction) and ! (negation) are used.

In order to better reflect the multiple facets of the grey-box challenge, there will be two kinds of rankings:

- A purely numeric ranking, according to the percentage of correctly verified properties, providing a true *competition.* In order to express ones confidence in own verification results, one can assign to each verification result a confidence weight from 0 to 9. In case of a correct answer, the weight value is added to the overall score of the contestant. Otherwise, twice the weight value is subtracted.
- A conceptual ranking, according to the employed (combination of) methods, emphasizing the *challenge* character. In this category, solutions will be reviewed and ranked by the challenge team. Due to the possible variety of methods, there may be several winners in this category.

4 How to Proceed

We have set up the challenge problems for Java and C almost identically. The main difference is that input and output symbols are given as strings in the Java setting, and as plain integers in the C setting (an explicit request from the community). Despite this difference, one can proceed exactly in the same fashion, e.g.:

- For solving the implicit problems, a tool might analyze the code for error/exception/assertion labels. Each such label that occurs in the code defines a reachability problem, which can be solved with the method of the contestant's choice. There are no limitations.
- The explicit problems, even if reminding of typical model-checking problems, may also be dealt with in any fashion, e.g. data-flow analysis, symbolic execution, testing, learning, (statistical) model checking, run-time methods, etc.
- The challenge is free-style. The contestant's are allowed to patch the code in any way, but the validity has, of course, to be stated according to the original problems.
- One should first concentrate on the problems and properties that one can master well. There is no need to give an answer to all problems. Of course, the more problems one can tackle, the more points one may be able to win, but be aware: wrong answers have a large penalty!

– The weighting scheme gives a way to express personal confidence in the obtained results, e.g., if one has found a path to some error and is convinced that this is indeed feasible, then one should weight it with confidence level 9. A liveness property, or stating that certain errors do not occur, is of course more risky.

Concerning the second form of ranking, the team needs to write a short summary of the chosen approach, the encountered hurdles, the solutions, and the obtained results. In these summaries, honesty, e.g., also concerning weaknesses/limitations of the employed methods, is important. Our challenge aims at profiling the various approaches and methods, which in particular means that weaknesses need to be identified. Of course, we are also very interested in new ideas and solutions that were motivated by the challenge.

The challenge starts with nine categories of ECA systems of varying complexity. After an initial phase of four weeks, three further problems of higher complexity will be added, specifically tailored to differentiate the participating competitors.

5 Relation to Other Challenges and Competitions

Competition and challenge events are well-understood in the 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 such as software verification [5], SAT and SMT solving [3, 11], planning [25], quantified boolean formulas [21], hardware model checking [7], or theorem proving [24]. All of those events impact the development pace and the 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] at TACAS is thematically closest to the RERS Grey-Box Challenge, even though it is complementary in the following respects:

– The RERS Challenge focuses on a very specific program pattern, but considers complex properties, and allows competitors to employ arbitrary means, both in terms of hardware and in terms of software.
– In contrast, SV-COMP focuses mainly on reachability problems to be solved on a given platform under clearly defined frame conditions, but with strongly varying program structures.

This difference characterizes SV-COMP as a pure competition with a clear ranking, which contrasts the RERS Challenge, whose frame conditions make it difficult to define a global ranking. This is why we have two rankings, one which is purely numerical, simply based on a 'multiple choice' test which may be solved 'free-style', and one where the approach taken, the underlying ideas, and the concrete realization are evaluated by the challenge team.

Acknowledgement. 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.: Event-Condition-Action Systems for Reconfigurable Logic Control. IEEE Transactions on Automation Science and Engineering 4(2), 167–181 (2007)

[2] Angluin, D.: Learning Regular Sets from Queries and Counterexamples. Information and Computation 75(2), 87–106 (1987)

[3] Balint, A., Belov, A., Järvisalo, M., Sinz, C.: SAT Challenge 2012. In: SAT (2012), `http://baldur.iti.kit.edu/SAT-Challenge-2012/index.html`

[4] Benatallah, B., Sheng, Q.Z., Dumas, M.: The Self-Serv Environment for Web Services Composition. IEEE Internet Computing 7(1), 40–48 (2003)

[5] Beyer, D.: Competition on Software Verification (SV-COMP). In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 504–524. Springer, Heidelberg (2012), `http://sv-comp.sosy-lab.org/`

[6] Bianco, A., de Alfaro, L.: Model Checking of Probabilistic and Nondeterministic Systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)

[7] Biere, A., Heljanko, K., Seidl, M., Wieringa, S.: HWMCC 2012. In: FMCAD (2012), `http://fmv.jku.at/hwmcc12/`

[8] Boyer, J., Mili, H.: IBM WebSphere ILOG JRules. In: Agile Business Rule Development, pp. 215–242. Springer (2011)

[9] Browne, P.: JBoss Drools Business Rules. Packt Publishing, Birmingham (2009)

[10] Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.): Model-Based Testing of Reactive Systems. LNCS, vol. 3472. Springer, Heidelberg (2005)

[11] Bruttomesso, R., Cok, D., Griggio, A.: SMT-COMP 2012. In: IJCAR (2012), `http://smtcomp.sourceforge.net/2012/`

[12] Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (2001)

[13] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in Property Specifications for Finite-State Verification. In: Boehm, B.W., Garlan, D., Kramer, J. (eds.) ICSE, pp. 411–420. ACM (1999)

[14] Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D.: Dynamically Discovering Likely Program Invariants to Support Program Evolution. IEEE Transactions on Software Engineering 27(2), 99–123 (2001)

[15] Havelund, K., Roşu, G.: Monitoring Java Programs with Java PathExplorer. Electronic Notes in Theoretical Computer Science 55(2), 200–217 (2001); RV 2001, Runtime Verification (in connection with CAV 2001)

[16] Hayes-Roth, F.: Rule-Based Systems. Commun. ACM 28(9), 921–932 (1985)

[17] Holzmann, G.J., Smith, M.H.: Software Model Checking: Extracting Verification Models from Source Code. Software Testing, Verification and Reliability 11(2), 65–79 (2001)

[18] King, J.C.: Symbolic Execution and Program Testing. Commun. ACM 19(7), 385–394 (1976)

[19] Leucker, M., Schallhart, C.: A Brief Account of Runtime Verification. Journal of Logic and Algebraic Programming 78(5), 293–303 (2009)

[20] McCarthy, D., Dayal, U.: The Architecture of an Active Database Management System. In: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, SIGMOD 1989, pp. 215–224. ACM, New York (1989)

[21] Narizzano, M.: QBFEVAL (2012), http://www.qbflib.org/index_eval.php
[22] Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer-Verlag New York, Inc., Secaucus (1999)
[23] Steffen, B., Howar, F., Merten, M.: Introduction to Active Automata Learning from a Practical Perspective. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 256–296. Springer, Heidelberg (2011)
[24] Sutcliffe, G., Suttner, C.: The State of CASC. AI Communications 19(1), 35–48 (2006), http://www.cs.miami.edu/~tptp/CASC/
[25] Vaquero, T.S., Fratini, S.: ICKEPS – International Competition on Knowledge Engineering for Planning and Scheduling. In: ICAPS (2012), http://icaps12.poli.usp.br/icaps12/ickeps