What does AI have to do with RV? Extended Abstract

Klaus Havelund*

Jet Propulsion Laboratory
California Institute of Technology
California, USA

Runtime Verification (RV) consists of monitoring the behavior of a system, either on-the-fly as it executes, or post-mortem after its execution for example by analyzing log files. Within the last decade several systems have been developed to address this issue. These systems usually implement specification languages which are based on formalisms such as state machines [11, 14, 8, 12, 5], regular expressions [1, 8], temporal logic [16, 10, 15, 3, 19, 9, 18, 8, 5], or grammars [8].

Some systems are based on some form of rewriting. In the Java PathExplorer (JPAX) system [15] a property is represented as an LTL [17] term. The semantics of LTL is in turn specified by a set of rewrite rules of the form $lhs \Rightarrow rhs$, for example, it contains a rewrite rule reflecting the semantics of the until operator (p until q): $p \mathsf{U} q = q \vee (p \wedge \bigcirc(p \mathsf{U} q))$. Each new event causes the current LTL term to be rewritten into a new term representing the formula that holds in the next step. For example, a formula on the form "always, an a implies eventually b": $\Box(a \rightarrow \Diamond b)$, on an event a is rewritten into $\Diamond b \ \wedge \ \Box(a \rightarrow \Diamond b)$. Properties in JPAX are propositional in the sense that formulas cannot refer to events that carry data. The TRACECONTRACT SCALA API lifts this principle to the SCALA programming language, while also handling data parameterization as well as state machines. Other systems based on this form of LTL-rewriting include [10, 3, 19, 18].

The RULER system [6, 7], and its state machine oriented derivative LOGSCOPE [4], implement rule-based systems. The state of such a system at any point in time is a set of facts, for example $\{open(file_1), closed(file_2)\}$. An incoming event is a new fact that is added to this set. A RULER specification is in principle a set of rules of the form: $lhs \Rightarrow rhs$, where the left hand side (lhs) is a set of conditions on the current monitoring state (set of facts), and the right hand side rhs is a set of actions to be taken in case the conditions are satisfied, for example adding or deleting facts. RULER's inspiration comes from imperative (executable) temporal logics, as for example found in METATEM [2]. The key problem in evaluating a set of rules given a set of facts is to perform *efficient* matching of facts against conditions in rules. It is not difficult to imagine, that in the case where many facts are stored this matching process can be costly if not performed in a smart manner.

* Part of the work described in this publication was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The field of Artificial Intelligence (AI) has itself studied a problem very similar to the runtime verification problem, namely *rule-based production systems*, used for example to represent knowledge systems. In such systems a specification is likewise a set of rules $lhs \Rightarrow rhs$, with a similar interpretation as in RULER. The classic AI approach to efficient matching is the RETE algorithm [13]. This algorithm maintains a network of facts, avoiding to re-evaluate all conditions in each rule's left hand side each time the fact database changes. We have implemented the rather sophisticated RETE algorithm in the SCALA programming language and are exploring its utility for the RV problem. We address its functionality (is it a solution for implementing runtime monitors) and its efficiency (how does it compare with state-of-the-art RV systems). Dynamic program visualization is used to demonstrate the algorithm and the modifications needed for it to apply to the RV problem.

References

1. C. Allan, P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, O. Lhoták, O. de Moor, D. Sereni, G. Sittamplan, and J. Tibble. Adding trace matching with free variables to AspectJ. In *OOPSLA'05*. ACM Press, 2005.
2. H. Barringer, M. Fisher, D. M. Gabbay, G. Gough, and R. Owens. MetateM: An introduction. *Formal Asp. Comput.*, 7(5):533–549, 1995.
3. H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *VMCAI*, volume 2937 of *LNCS*, pages 44–57. Springer, 2004.
4. H. Barringer, A. Groce, K. Havelund, and M. Smith. Formal analysis of log files. *Journal of Aerospace Computing, Information, and Communication*, 7(11):365–390, 2010.
5. H. Barringer and K. Havelund. TraceContract: A Scala DSL for trace analysis. In *17th International Symposium on Formal Methods (FM'11), Limerick, Ireland, June 20-24, 2011. Proceedings*, volume 6664 of *LNCS*, pages 57–72. Springer, 2011.
6. H. Barringer, D. Rydeheard, and K. Havelund. Rule systems for run-time monitoring: from Eagle to RuleR. In *Proc. of the 7th Int. Workshop on Runtime Verification (RV'07)*, volume 4839 of *LNCS*, pages 111–125, Vancouver, Canada, 2007. Springer.
7. H. Barringer, D. E. Rydeheard, and K. Havelund. Rule systems for run-time monitoring: from Eagle to RuleR. *J. Log. Comput.*, 20(3):675–706, 2010.
8. F. Chen and G. Roşu. Parametric trace slicing and monitoring. In *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09)*, volume 5505 of *LNCS*, pages 246–261, 2009.
9. M. D'Amorim and K. Havelund. Event-based runtime verification of Java programs. In *Workshop on Dynamic Program Analysis (WODA'05)*, volume 30(4) of *ACM Sigsoft Software Engineering Notes*, pages 1–7, 2005.
10. D. Drusinsky. The temporal rover and the ATG rover. In *SPIN Model Checking and Software Verification*, volume 1885 of *LNCS*, pages 323–330. Springer, 2000.
11. D. Drusinsky. *Modeling and Verification using UML Statecharts*. Elsevier, 2006. ISBN-13: 978-0-7506-7949-7, 400 pages.
12. Y. Falcone, J.-C. Fernandez, and L. Mounier. Runtime verification of safety-progress properties. In *Proc. of the 9th Int. Workshop on Runtime Verification (RV'09)*, volume 5779 of *LNCS*, pages 40–59. Springer, 2009.

13. C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.
14. K. Havelund. Runtime verification of C programs. In *Proc. of the 1st TestCom/-FATES conference*, volume 5047 of *LNCS*, Tokyo, Japan, 2008. Springer.
15. K. Havelund and G. Rosu. Monitoring programs using rewriting. In *16th ASE conference, San Diego, CA, USA*, pages 135–143, 2001.
16. I. Lee, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan. Runtime assurance based on formal specifications. In *PDPTA*, pages 279–287. CSREA Press, 1999.
17. A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977.
18. V. Stolz and E. Bodden. Temporal assertions using AspectJ. In *Proc. of the 5th Int. Workshop on Runtime Verification (RV'05)*, volume 144(4) of *ENTCS*, pages 109–124. Elsevier, 2006.
19. V. Stolz and F. Huch. Runtime verification of concurrent Haskell programs. In *Proc. of the 4th Int. Workshop on Runtime Verification (RV'04)*, volume 113 of *ENTCS*, pages 201–216. Elsevier, 2005.