Rule systems for run-time monitoring: from EAGLE to RULER

Howard Barringer, David Rydeheard*
EMAIL: {Howard.Barringer, David.Rydeheard}@manchester.ac.uk

Klaus Havelund†
EMAIL: Klaus.Havelund@jpl.nasa.gov

January 26, 2007

Summary

EAGLE was introduced in [2] as a general purpose rule-based temporal logic for specifying run-time monitors. A novel and relatively efficient interpretative trace-checking scheme via stepwise transformation of an EAGLE monitoring formula was defined and implemented. However, application in real-world examples has shown efficiency weaknesses, especially those associated with large-scale symbolic formula manipulation. For this presentation, first we reflect briefly on the strengths and weaknesses of EAGLE and then we introduce RULER, a primitive conditional rule-based system, which can be more efficiently implemented for run-time checking, and into which one can compile various temporal logics used for run-time verification.

Background and motivation

A plethora of logics have been used for the specification of behavioural system properties that can be dynamically checked either on-line throughout an execution of the system or off-line over an execution trace of the system. Some form of linear-time temporal logic usually forms the basis for the specification logic. This large variety of logics prompted the search for a small and general framework for defining monitoring logics, which would be powerful enough to capture most of the existing logics, thus supporting future and past time logics, interval logics, extended regular expressions, state machines, real-time and data constraints, and stochastic behaviour. The framework should support the definition of new logics easily and should support the monitoring of programs with their complex program states. EAGLE was the result.

The EAGLE logic is a restricted first order, fixed-point, linear-time temporal logic with chop (concatenation) over finite traces. As such, the logic is highly expressive and, not surprisingly, EAGLE's satisfiability (validity) problem is undecidable; checking satisfiability in a given model, however, is decidable and that is what's required for run-time verification. The syntax and semantics of EAGLE is succinct. There are four primitive temporal operators: $\bigcirc$ — next, $\bigodot$ — previously, $\cdot$ — concatenation, and ; — chop (overlapping concatenation, or sequential composition). Temporal equations can be used to define schema for temporal formulae, where the temporal predicates may be parameterized by data as well as by EAGLE formulas. The usual boolean logical connectives exist. For example, the linear-time $\Box$, $\mathcal{U}$ and $\mathcal{S}$ temporal operators can be introduced through the following equational definitions.

$$\textbf{max } \texttt{Always}(\textbf{Form } F) = F \wedge \bigcirc \texttt{Always}(F)$$

$$\textbf{min } \texttt{Until}(\textbf{Form } F_1, \textbf{Form } F_2) = F_2 \vee (F_1 \wedge \bigcirc \texttt{Until}(F_1, F_2))$$

$$\textbf{min } \texttt{Since}(\textbf{Form } F_1, \textbf{Form } F_2) = (F_2 \vee (F_1 \wedge \bigodot \texttt{Since}(F_1, F_2)))$$

The qualifiers **max** and **min** indicate the positive and, respectively, negative interpretation that is to be given to the associated temporal predicate at trace boundaries — corresponding to maximal and minimal solutions to the equations. Thus $\bigcirc\texttt{Always}(p)$ is defined to be true in the last state of a given trace, whereas $\bigcirc\texttt{Until}(p, q)$ is false in the last state.

Even without data parametrization, the primitive concatenation temporal operators in conjunction with the recursively defined temporal predicates takes the logic into the world of context-free expressivity. Parametrization of temporal predicates by data values allows us to define real-time and stochastic logical operators.

We will reflect on two related questions: Is EAGLE too expressive for run-time monitoring? If not, is EAGLE expressive enough? For example, there are arguments to use deterministic versions of temporal concatenation and chop for run-time monitoring — and there are several different forms of deterministic cut, e.g. left minimal, left maximal, right minimal, right maximal, etc..

What can be said about the computational effectiveness of algorithms for EAGLE trace-checking? Firstly, trace-checking of full EAGLE can be under-

*School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, UK

†Columbus Technologies, Laboratory for Reliable Software, NASA's Jet Propulsion Laboratory, Pasadena, CA 91109, USA

taken on a state-by-state basis, even though the logic has the same temporal expressiveness over the past as over the future; basically, our published trace-check algorithm maintains sufficient knowledge about the past in the evolving monitor formulas. Unfortunately, given the presence of data arguments in temporal predicates, an explosion in the size of the evolving monitor formula may occur.

What was clear to us at the time was that there were some practically useful and efficiently executable subsets of EAGLE. One such fragment for which we computed complexity results was the LTL (past and future) fragment of EAGLE [3]. Despite the nice features of EAGLE, we still believed we should continue to search for a powerful and simpler "core" logic, one that is easy and efficient to evaluate for monitoring purposes.

Introducing RULER

The EAGLE trace-checking algorithm is essentially interpretative. Given a monitor formula and an input state, the trace-checker computes a new monitor formula that will need to hold in the next state for the original monitor formula to hold on the current input; recursively defined temporal predicates are replaced by their definitions and separated into what has to hold now and in the future. Considerable formula rewriting, i.e. data structure manipulation, is required. The question thus arises: what compilation strategy might be possible in order to optimize the interpretation process? Perhaps some form of predicated automata can be compiled. Similar issues arose when interpretation improvements were being sought for the executable logic METATEM [1]. Fisher's separated normal form was developed [4], leading to improved temporal resolution-based theorem-proving techniques.

As an experiment, we have constructed a simple rule system into which one can compile various forms of linear-time temporal logic. The rules bear a strong resemblance to the step rules used in graph-based temporal resolution. Let us give a flavour for the propositional case of RULER. Let the letters a, b, c, etc., denote propositional atoms that can be evaluated in a given input state, and the letters r_1, r_2, r_3, etc., denote rule names, which in turn are associated with conditional monitoring step rules. A rule name is also treated as a propositional atom. Rule definitions are of the form:

$$\text{ruleName} : \text{antecedent} \multimap \text{consequent}$$

where the antecedent is a conjunctive list of atoms, and the consequent is a disjunctive list of conjunctive lists of atoms. Here's an example of a set of rules representing the temporal monitoring formula $\Box((\bigodot(a \,\mathcal{S}\, b)) \Rightarrow \bigcirc(a \vee \bigcirc c))$, assuming we have r_0, r_1 and r_3 initially active.

$$\begin{array}{ll} r_0: & \multimap\, r_0, r_1, r_3 \\ r_1: & b \multimap r_2 \\ r_2: & a \multimap r_2 \end{array} \;\Big\|\; \begin{array}{ll} r_3: & r_2 \multimap a \,|\, r_4 \\ r_4: & \multimap\, c \end{array}$$

The evaluation of a rule name in a state determines the associated rule's activity status. Only active rules are applied, and the consequent of a rule is applied only if the rule's antecedent holds (an empty antecedent is always true). Thus when the rule r_0 is applied, the next rule activation state must contain rules r_0, r_1 and r_3. The rule r_1 requires, however, that the atom b is true in order for the consequent to apply in the next activation state. The rule r_3 has an antecedent of r_2, which means that r_2 must be a currently active rule in order for the consequent of r_3 to be applied. The latter gives a choice: the next state must have atom a true or rule r_4 must be active. Monitoring a sequence of states with such rule sets proceeds as follows.

```
create an initial set of initial rule
                          activation states
WHILE observations exist DO
  obtain next observation state
  merge observation state with the set of
                     rule activation states
  raise monitoring exception if there's
                              total conflict
  for each of the current merged states,
    apply activated rules to generate a
              successor set of activation states
  union successor sets to form the new frontier
                       of rule activation states
OD
```

The merge of observation state with the set of rule activation states results in a set of consistent rule activation sets. If no consistent sets results, we say a total conflict with given rule set has occurred, i.e. the observation trace has failed to satisfy the given rule set. For rule set satisfaction, we need to state which rule names are allowed to be active once the whole observation trace has been monitored — similar to **max** and **min** in EAGLE. Whilst we can't show any details of this working, we assert that this primitive rule system can be more efficiently executed than the direct interpretation of EAGLE.

Naturally, our full paper will provide semantic details for RULER and translations from various temporal logic subsets. In addition, we will discuss other variations on these primitive rules, such as universality (in the above example, rule r_0 acted as a generator for r_1 and r_3), interpretations for negation of rules (forced non activation) and giving rules priorities for use in conflict resolution.

A rather fuller bibliography will be provided in the full paper!

References

[1] H. Barringer, M. Fisher, D. Gabbay, R. Owens and M. Reynolds. *The Imperative Future: Principles of Executable Temporal Logic*. Research Studies Press. 1996.

[2] H. Barringer, A. Goldberg, K. Havelund and K. Sen. Rule-Based Runtime Verification. Proceedings of the *VMCAI'04, 5th International Conference on Verification, Model Checking and Abstract interpretation*, Venice. Volume 2937, Lecture Notes in Computer Science, Springer-Verlag, 2004. 2004.

[3] H. Barringer, A. Goldberg, K. Havelund and K. Sen. Run-time Monitoring in Eagle. Proceedings of *PADTAD '04, Santa Fe, New Mexico*, IEEE Computer Society, IDPDS'04, Volume 17, Number 17, pp 264b, 2004.

[4] M.D. Fisher. A Normal Form for Temporal Logics and its Applications in Theorem-Proving and Execution. Journal of Logic and Computation, Volume 7, Number 4, pp 429-456, 1997.