

# OPTIMIZING TIMING ANALYSIS AND VERIFICATION OF EMBEDDED SYSTEMS USING RULE-BASED-ANALYTIC TECHNIQUES

Sukhdeep Sodhi and Albert M.K. Cheng  
Real-Time Systems Laboratory, Department of Computer Science  
University of Houston, Houston, TX 77204  
ssodhi@uh.edu, cheng@cs.uh.edu

## Abstract

*In previous work we have developed very fast static techniques to determine response times of rule-based programs. This paper shows that the problem of formal verification of embedded systems can be represented as the problem of determining response time of rule-based systems. We do this by encoding the specification and safety assertion given in RTL as a rule-based EQL or MRL program. This paper goes on to show that our approach requires quadratic time while the equivalent RTL approach requires exponential time in the worst case.*

## 1. Introduction

Most formal analysis and verification techniques suffer from the combinatorial state explosion problem. Despite advances in symbolic representations of the state space of the system being analyzed, more optimizations need to be performed to further combat this problem. This project develops a strategy for optimizing the timing analysis and verification of embedded systems using analysis techniques that we have developed for real-time rule-based systems. We introduce two approaches to the problem.

In the first approach we begin with encoding the behavioral specification and safety assertion in a first order logic called RTL (Real-Time Logic) [1]. The safety assertion is then negated and encoded along with the behavioral specification as a rule-based program in EQL (Equational Logic Language) [2]. Then we show that if this EQL program has bounded response time, the specification violates the safety assertion.

The second approach requires writing the behavioral specification and safety assertion in RTL, which is subsequently converted to a rule-based MRL (Macro Rule-Based Language) program. This MRL [3] program has rules which represent firstly the behavioral specification and secondly the negation of the safety assertion. We then show that if this MRL program has bounded response time the specification adheres to the safety assertion.

Considerable work has been done on optimizing response time analysis techniques of EQL and MRL programs [4,5,6,7]. In some cases rule-based response-time analysis has been

orders of magnitude faster than equivalent state space techniques [7]. In this paper we introduce techniques, which represent the problem of embedded-systems verification as the problem of response time analysis of rule-based systems.

The rest of the paper is organized as follows. Section 2 briefly introduces the reader to RTL. Section 3 covers concepts of EQL and MRL. Section 4 describes our rule-based analytic techniques for formal verification. Section 5 concludes the paper.

## 2. RTL

RTL is a first-order logic language that was developed for formally specifying real-time systems and their absolute timing properties. It easily captures the timing requirements of the specified system while making it easy to manipulate the specification mechanically. It is based on the event action model, augmented with several features such as the occurrence function @, which assigns time values to event occurrences.

There are three types of RTL constants: actions, events and integers. An action is a schedulable unit of work that can be primitive or constant. Event constants serve as temporal markers and are classified as (a) start events indicating the beginning of actions, preceded by  $\uparrow$ ; (b) stop events indicating the end of actions, preceded by  $\downarrow$ ; (c) transition events indicating the change in certain attributes of the system state; and (d) external events, preceded by  $\Omega$ .

Mok et. al introduced a restricted class of RTL formulae [8]. This restricted class was motivated by the fact that in the specification of many real-time systems. Its general representation is as follows:

*occurrence function*  $\pm$  *integer constant*  $\leq$  *occurrence function*

Such a restricted RTL class allows the use of a graph-theoretic approach for analysis. The set of inequalities is represented as a constraint graph. An inequality  $X_i \pm A_{ij} \leq X_j$  is represented by a weight  $A_{ij}$  connecting the node  $X_i$  to the node  $X_j$ . Then a set of inequalities represented by such a graph is unsatisfiable iff there is a cycle in this graph with a positive total weight on it. This RTL subclass is sufficiently

expressive to represent a broad range of real-time systems and has been used successfully [9]. Formulae belonging to this restricted RTL class can be easily represented as rule-based production systems.

### 3. EQL and MRL

EQL allows one to write zero-order rule-based programs. An EQL program has a set of rules for updating variables, which denote the state of the physical system under control. The firing of a rule computes a new value for one or more state variables to reflect changes in the external environment as detected by sensors. Intuitively, rules in an EQL program are used to express the constraints on a system and also the goals of the controller.

MRL is a first-order rule-based production language. It is designed as a more expressive superset of EQL to serve as a general-purpose domain-independent language with predictable response time characteristics. An MRL program comprises of two parts: a set of rules and a structured database called working memory (wm). A rule contains an enabling condition (EC) and an action part that contains a set of assignments. The working memory contains two types of variables: primitive variables and macro variables. Primitive variables are similar to variables in a regular programming language like Pascal. Each primitive variable has an associated type. Each macro variable corresponds to a set of primitive variables. Rules that contain macro variables are referred to as macro rules. Macro variables can be referred to only by special symbols called pattern variables, which are not real variables since they do not occupy any memory. The scope of pattern variables is restricted to one rule only. In MRL, pattern variables are always quantified. Pattern variables with the prefix question mark “?” are existentially quantified variables, and those prefixed with an asterisk “\*” are universally quantified variables. Existentially quantified variables may appear anywhere in a rule while universally quantified variables can appear only in the enabling condition. This restriction poses a problem to converting specifications written in RTL to a MRL rule-based program. However, this restriction is not impossible to overcome [3].

We conceptually model the execution of MRL programs as follows. Before execution, a macro rule is expanded to a set of plain rules each of which contains one of the possible combinations of macro variable expansions, i.e., each macro variable is instantiated to one of the

corresponding expanded variables. The result of the expansion is an equivalent EQL program which has the same behavior as the MRL program. As a consequence EQL analysis techniques can be used to analyze MRL programs [10].

### 4. Rule-Based Verification Techniques

To show that a system or program meets certain safety properties, we can relate the specification of the system to the safety assertion representing the desired safety properties. One of the following three cases may result from the analysis relating the specification and the safety assertion: (1) Safety assertion is a theorem derivable from the specification, thus the system is safe with respect to the behavior denoted by the safety assertion; (2) Safety assertion is unsatisfiable with respect to the specification, so the system is inherently unsafe; and (3) Negation of the safety assertion is satisfiable under certain conditions meaning additional constraints must be added to the system to ensure its safety.

Our first approach uses analysis techniques developed for determining response times of EQL programs to determine whether the system being analyzed satisfies case 1 mentioned above. The second approach uses reachability analysis of MRL programs to check whether the system satisfies case 2. We now look at both approaches in more detail.

#### 4.1 Verification Using Bounded Response Time of EQL

In this technique we begin with the system specification (SP) and the safety assertion (SA) expressed as restricted RTL subclass formulae. Our goal now is to show that SA is a theorem derivable from SP, i.e.,  $SP \rightarrow SA$ . This is equivalent to showing the negation of  $SP \rightarrow SA$ , i.e.,  $\neg(SP \rightarrow SA)$ , is unsatisfiable. Since  $SP \rightarrow SA$  can be rewritten as  $\neg SP \vee SA$ , our goal is to prove that the formula  $SP \wedge \neg SA$  is unsatisfiable.

We begin by converting SP and  $\neg SA$  to Skolem standard form. In the Skolem standard form existential quantifiers are replaced by Skolem constants [11]. Once existential quantifiers are removed it is trivial to remove the universal quantifiers as well. The second step is unification of the different occurrence functions in the RTL inequalities. After unification different instances of the same occurrence function accepting different variables or constants are treated as the same entity.

After the previous two steps we have removed all quantifiers, so the RTL formula SP

$\wedge \neg SA$  can be easily converted to a zero order rule-based program in EQL. This EQL program is not semantically equivalent to the specification and safety assertion written in RTL. It is however equivalent to the constraint graph representing the formula  $SP \wedge \neg SA$ . So now a cycle in the constraint graph can be equated to the EQL program having unbounded response time. Therefore we can conclude that if the EQL program has bounded response time there are no cycles in the constraint graph. Recall that a positive cycle in the constraint graph represents unsatisfiability of the RTL formula  $SP \wedge \neg SA$ . Therefore a lack of any cycles (i.e. bounded response time) represents the fact that  $SP \wedge \neg SA$  is not unsatisfiable, in other words the safety assertion is not a theorem derivable from the specification. Readers interested in more details and examples regarding unification and constraint graph analysis are referred to [11]. An example is given in the appendix.

#### 4.2 Verification Using Bounded Response Time of MRL

This technique also begins with the behavioral specification (SP) and the safety assertion (SA) represented as restricted RTL formulae. We then convert the conjunction of the specification and the negated safety assertion, i.e.,  $SP \wedge \neg SA$  to a rule-based MRL program. MRL is a first order language so this conversion can be done without any loss of information. The corresponding MRL program preserves the semantics of the RTL formula  $SP \wedge \neg SA$ . Specifically the program will have a set of rules that represent SP and a set of rules the represent  $\neg SA$ .

A simple approach to verification would be using reachability analysis to determine whether a rule belonging to the set representing  $\neg SA$  is directly or indirectly enabled due to the firing of a rule, which belongs to the set representing SP. This scenario would represent that a condition violating the safety assertion is reachable from the behavioral specification therefore the system is unsafe. However, this approach would be very inefficient.

Making a small modification to the rules representing  $\neg SA$ , can lead to a more efficient approach for a subset of the systems. Changes can be made to the enabling condition and the action part of MRL rules so that once a rule is fired it would enable itself. This would result in a loop whenever any of the rules representing  $\neg SA$  is fired, leading to the MRL program having unbounded response time. Once

these changes have been made, MRL response time analysis can be used to determine whether the program has bounded or unbounded response time. Unbounded response time is equivalent to having a path from one of the SP rules to one of the  $\neg SA$  rules, i.e., the specification violates the safety assertion. This approach would only work iff a rule-based program representing SP alone does not have unbounded response time. We are currently working towards removing this limitation.

#### 4.3 Timing Requirements

Determining response time of EQL programs requires the creation of a high-level dependency (HLD) graph [12], which can done in  $O(n^2)$  time where  $n$  is the number of rules in the program. The second step, which checks for special forms can also be done in  $O(n^2)$  time. Our procedure for converting from RTL to EQL as well as MRL requires  $O(m)$  time, where  $m$  is the number of clauses in the RTL. So given an RTL specification we can determine whether the SA is derivable from SP in quadratic time. Response time analysis of OPS5 programs, which are similar to MRL can also be done in quadratic time [13]. Checking the RTL constraint graph for positive cycles requires exponential time in the worst case i.e.,  $O(2^k)$  where  $k$  is the number of positive cycles. Since both the approaches shown in this paper require quadratic time we expect them to be faster than the RTL constraint-graph approach.

#### 5. Conclusion

We have introduced two approaches that use response time analysis of rule-based programs for formal verification of real-time systems. The first approach uses response time analysis of EQL programs to determine whether the behavioral specification violates the safety assertion and the second approach uses response time analysis of MRL programs to achieve the same objective. Very fast static techniques to determine response times of EQL and MRL programs have been developed, in some cases analysis time is many orders of magnitude faster compared to approaches that use state-space techniques for analysis. Our goal in introducing these two approaches is to exploit these rule-based techniques to develop efficient techniques for verification of embedded systems. A more detailed description would be the subject of a separate paper.

#### References

- [1] F. Jahanian, A.K. Mok, "Safety Analysis of Timing Properties in Real-Time Systems," *IEEE*

*Trans. Software Engineering*, Vol. SE-12, No. 9, pages 890-904, Sept. 1986.

[2] A.K. Mok, "Formal analysis of real-time equational rule-based system," *Proc. of the 1989 IEEE RTSS*, pages 308-318, 1989.

[3] C.K. Wang, A.K. Mok, "MRL: The Language," Univ. of Texas, Austin, Comp. Sci. Dept.

[4] B.Zupan, A.M.K. Cheng, "Optimization of Rule-Based Systems Using State Space Graphs," *IEEE Trans. on Knowledge and Data Engg.*, Apr. 1997.

[5] C.K. Wang, A.K. Mok, "Timing analysis of MRL: A Real-Time Rule-Based System," *J. of Real-Time Systems*, 5(1), Mar. 1993, 89-128.

[6] J. R. Chen, A.M.K. Cheng, "Response time analysis of EQL real-time rule-based systems," *IEEE Trans. on Knowledge and Data Engg.*, vol. 7, np. 1, pages 26-43, Feb. 1995.

[7] J.C. Wang, A. M. K. Cheng, "A State-Space-Based Approach for Optimizing MRL Rule-Based Programs," *Proc. Intl. Conf. on Parallel Distributed Computing Systems*, Cambridge, MA, Nov. 1999.

[8] F. Jahanian, A.K. Mok, "A Graph-theoretic Approach for Timing Analysis and its Implementations," *IEEE Trans. Computers*, Vol C-36, No. 8, pages 933-947, Dec. 1987.

[9] L.E.P. Rice, A.M.K. Cheng, "Timing Analysis of the X-38 Space Station Crew Return Vehicle Avionics," *Proc. IEEE RTAS*, Canada, Jun. 1999.

[10] C. K. Wang, A. K. Mok, A. M. K. Cheng, "MRL: A Real-Time Rule-Based Production System," *Proc. 11th IEEE RTSS*, Orlando, FL, Dec. 1990

[11] A.M.K. Cheng, *Real-Time Systems: Scheduling, Analysis, and Verification*, Jul. 2002

[12] A.M.K. Cheng, J.C. Browne, A.K. Mok, R.H. Wang, "Analysis of Real-Time Rule-Based Systems with Behavioral Constraint Assertions Specified in Estella," *IEEE Trans. Software Engg.*, vol 19, pages 863-885, Sep 1993.

[13] A.M.K. Cheng, J.R. Chen, "Response Time Analysis of OPS5 Production Systems," *IEEE Trans. on Knowledge and Data Engg.*, May/June 2000.

## Appendix

### A. Converting RTL to EQL

Refer the RTL specification in [11]. SP is represented in clausal form as

$$\begin{aligned} f(x) &\leq g_1(x) \\ g_2(x) - 30 &\leq f(x) \\ g_1(y) + 15 &\leq g_2(y) \end{aligned}$$

Rewriting the  $\neg$ SA in clausal form we get

$$\begin{aligned} f(T) + 45 &\leq h(U) \\ h(U) - 59 &\leq f(T) \\ h(U) + 1 &\leq g_2(T) \vee g_2(T) + 46 \leq h(U) \end{aligned}$$

where, f, g, h are integer functions

x is a free variable

X is a skolem constant

In [8] f(x) and f(T) are represented by the same node in the graph. Applying clustering  $SP \wedge \neg SA$  to the above formulae we can get the formula, which is equivalent to the graph constructed using the process in [8].

$$\begin{aligned} f &\leq g_1 \\ g_2 - 30 &\leq f \\ g_1 + 15 &\leq g_2 \\ f + 45 &\leq h \\ h - 59 &\leq f \\ h + 1 &\leq g_2 \vee g_2 + 46 \leq h \end{aligned}$$

The above formula can be easily converted to the EQL program shown below.

```
INPUTVAR
f, g1, g2, h
VAR
time_f, time_g1, time_g2, time_h, a
INIT
a = f = g1 = g2 = h = 0
RULES
(1) time_f = time; f = 2 IF (f == 1)
(2) time_g1 = time; g1 = 2 IF (g1 == 1)
(3) time_g2 = time; g2 = 2 IF (g2 == 1)
(4) time_h = time; h = 2 IF (h == 1)
(5) time_f = time_g1; f = 2 IF (g1 == 2 AND NOT f)
(6) time_g2 = time_f + 30; g2 = 2 IF (time == time_f + 30 AND NOT g2)
(7) time_g1 = time_g2 - 15; g1 = 2 IF (g2 == 2 AND NOT g1)
(8) time_f = time_h - 45; f = 2 IF (h == 2 AND NOT f)
(9) time_h = time_f + 59; h = 2 IF (time == time_f + 59 AND NOT h)
(10) a = 0; time_h = time_g2 - 1; h = 2 IF (g2 == 2 AND a == 1 AND NOT h)
(11) a = 1; time_g2 = time_h - 46 IF (time >= time_h AND a == 0)
(12) f = 0 IF (f == 2)
(13) g1 = 0 IF (g1 == 2)
(14) g2 = 0 IF (g2 == 2)
(15) h = 0 IF (h == 2)
```

Bounded response time of graph is equivalent to a lack of cycles in the RTL constraint graph. Unbounded response time represents both positive and negative cycles in graph. We are currently exploring modifications to the rule-based program, to allow for distinguishing between positive cycles and negative cycles. This program requires a simple priority mechanism to be used while rule firing.

Converting from RTL to MRL is very similar to the process shown above.