On Revisiting Hoare Logic for Real-Time Systems

David Pereira and Luís Miguel Pinho

CISTER Research Centre/INESC-TEC
School of Engineering (ISEP) – Polytechnic Institute of Porto (IPP), Portugal
{dmrpe,lmp}@isep.ipp.pt

Abstract. Modern Real-Time Systems (RTSs) are complex and pervasive, and normally assume the form of software components embedded in some larger system. Several RTSs have a safety-critical behavior, *i.e.*, their failure can lead to catastrophic events. Henceforth, this class of systems must be developed with levels of rigor that are not characteristic of the development cycle of standard software. Formal specification and verification are being progressively introduced to detect flaws in the development cycle of RTSs. The most popular techniques are Model Checking and Abstract Interpreation, but reality is that most verification activities reduce to unit testing and fault-injection. Actual program verification is almost no existant. In this paper we extend Hoare logic with non-functional primitives characteristic from RTSs. We present a simple language for programming hierarquical RTSs, its operational semantics, and show the associated proof-system sound.

1 Introduction

Runtime verification is an emerging formal verification thechnique in the context of hard real-time systems. Runtime verification is a kind of *lightweight formal method* that does not have th goal of constructing a mechanically verifiable evidence of some property. Instead, runtime verification concerns with real data that is observed during a system's execution, and if such data is conformant with a prescribed formal specification.

1.1 Contributions

The main contribution of this paper is a rich language in which we can specify, implement, and verify safety-critical RTSs. Our proposal extends existing languages with a primitive notion of off-line scheduler that ensures non-interference and adequate preemption between concurrent tasks. The off-line scheduling is calculated based on the worst-execution times calculated directly from the bodies of the tasks considered, and taking into account information about their periods and deadlines.

1.2 Paper Organization

This paper is organized as follows. In Section 2 we introduce both the syntax and operational semantics of our language proposal. In Section 3 we describe the syntax of the assertion language considered, as well as the proof systems designed to capture the expected behavior of the structure of our hierarquical programming language. In **??** we give a soundness proof of the deductive system and the operational semantics defined in Seciont 2 and present some brief references to its mechanization in the Coq proof assistant. In Section **??** we detail the specification and implementation of the mine pump example of Burns [] within our language, and show we have proved its more relevant properties. Finally, we draw some conclusions and point to future work to leverage the proposed ideas to a real-work example level.

2 Syntax and Operational Semantics

The programming language we are about to introduce targets the development of simple, yet non-trivial, hierarquical RTSs. An hierarquical RTS is a RTS built from components with their own real-time constraints and taskset, as well as possibly different internal scheduling schemes. Components in a hierarquical RTS respect a global toplevel scheduling policy, but their internal task set and scheduling policy are opaque to external analysis by other components.

IMP$^{\theta}$ programs are interpreted over a fine-grained operational semantics that considers a global clock value θ capturing the evolution of time during program computation. Hence, we extend traditional small-step structural operational semantic to have a fined-grained notion of execution cost of the segments of the program under evaluation. This kind of semantics is needed in order to allow offline schedulability to determine how a concurrent, safety-critical RTS launches and preempts its composing tasks. T

3 Assertion Language and Axiomatic Semantics

4 Related Work

So far, not many approaches for runtime verification of real-time

5 Requirements for a Runtime Verification

[1]

References

1. Yassine Lakhneche and Jozef Hooman. Metric temporal logic with durations. *Theor. Comput. Sci.*, 138(1):169–199, February 1995.