FORMAL ANALYSIS FOR REAL-TIME SCHEDULING

Bruno Dutertre and Victoria Stavridou, SRI International, Menlo Park, CA

Introduction

In modern avionics architectures, application software increasingly relies on services provided by a real-time operating system (RTOS). An application is typically structured in sets of processes that share common hardware resources via the RTOS. Such architectures present numerous advantages for software development by decoupling the application software from the specifics of the underlying hardware. However, they also present challenging certification problems. The RTOS is a highly critical component of the overall avionics system and must be certified to the highest levels of assurance. In addition, new software integration issues have to be addressed such as ensuring that the processes that share common resources will satisfy their performance requirements.

Scheduling and synchronization services are among the most critical services an RTOS must provide. These services can be particularly subtle, and it is difficult to obtain strong evidence that they will perform properly in all circumstances. Testing and inspection are not sufficient for this purpose. Aside from the pure problem of showing that the RTOS behaves as expected, a second issue is to determine whether an application that relies on the scheduling discipline and uses the communication services provided by the RTOS satisfies its timing requirements.

A well-established theory of real-time scheduling does exist, that should provide a sound foundation to the development and validation of real-time system. Yet, the abundant theoretical work may not by itself provide the degree of assurance required in safety-critical domains such as avionics. The literature has traditionally relied on informal approaches, and proofs often rely more on intuitive explanations than precise, rigorous arguments. Since the problems are complex and subtle, there are examples of erroneous results and flawed proofs in the literature. In addition, real operating systems are often richer and more complex than considered in the literature. In integrated avionics, RTOS must provide strong partitioning guarantees to prevent interference between processes sharing common hardware resources. Complex systems relying on fault-tolerant rate-monotonic scheduling are being built [1] or kernels employing a nontrivial mixing of static and priority-based scheduling are being considered [2]. The general theory rarely applies directly but often has to be adapted to a specific context.

We examine how formal methods can help address these issues. Formal methods can be used to develop very precise, complete, and rigorous proofs that theoretical results are correct and properly applied to a particular context. Formal modeling and verification can provide very strong evidence that an RTOS satisfies critical scheduling and synchronization properties. As an illustration, we discuss the formalization and verification of the *priority-ceiling protocol* [3], a scheduling and synchronization protocol used in common real-time operating systems. We then discuss extensions of this work to more complex types of kernels and the benefits of formal methods in real-time scheduling problems.

Real-Time Scheduling Issues in Safety-Critical Applications

In traditional real-time systems, an application is decomposed into a set of processes or tasks that run concurrently on one of more processors. A scheduling policy determines how the processing resources are shared between the different tasks. Typical RTOSs use fixed-priority preemptive scheduling: each task is assigned a fixed priority, and, at all time, the task of highest priority that is ready to execute is allocated the processor. Other policies are also possible, such as earliest-deadline-first scheduling, table-driven scheduling, or other more complex schemes.

The fundamental problem in this context is to determine conditions under which a given set of tasks satisfies its timing requirements. Many theoretical results provide such guarantees for different classes of systems, relying on different scheduling policies, and with different assumptions about tasks and timing constraints. In particular, fixed-priority preemptive scheduling has been extensively studied and schedulability results are known for many types of task sets and timing constraints.

A simple situation is when the tasks are independent and periodic. In such a case, n tasks $\tau_1, \cdots, \tau_n$ are each characterized by a computation cost C_i and a period T_i. Task τ_i is activated at successive times $t_0, t_0 + T_i, t_0 + 2T_i$, and so forth, and each invocation must terminate within a one-period interval. In other words, τ_i must be allocated a total amount of C_i time units of processing time in every interval $[t_0 + kT_i, t_0 + kT_i + T_i)$. In fixed-priority scheduling, it is known that, under the rate-monotonic priority assignment[1], the problem is feasible if the following condition is satisfied

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) .$$

Liu and Layland proved this property in 1973 [4]. They also established other important results, such as the fact that the rate-monotonic priority assignment is optimal. The condition above is based on a worst-case analysis and many task sets that do not satisfy the inequality are still feasible. A better analysis technique consists of computing the worst-case response time of a task. The procedure is sketched in Figure 1, where H_i is the set of tasks of higher priority than τ_i. The sequence (W_i^k) is computed until either it converges to a fixed point $W_i \leq T_i$ or it reaches a value W_i^k that is greater than T_i. In the former case, every invocation of task τ_i is guaranteed to terminate within a delay W_i. In the latter case, τ_i cannot be guaranteed to meet its deadlines.

The approach sketched in Figure 1 (as well as other approaches) generalize to many other task models, including models where tasks can communicate with each other and where deadlines are different from the task periods [5,6]. With such analysis techniques, the theory of real-time scheduling, especially in the fixed-priority context, is now mature and applicable to real-world systems.

$$\boxed{\begin{aligned} W_i^0 &= C_i \\ W_i^{k+1} &= C_i + \sum_{j \in H_i} C_j \left\lceil \frac{W_j^k}{T_j} \right\rceil \end{aligned}}$$

Figure 1. Computing the Worst-Case Response Time of a Task

However, in safety-critical applications, high assurance is required, and the following issues have to be examined:

- Which guarantees do we have that the results published in the literature are actually correct?
- Which guarantees do we have that a given RTOS actually satisfies the assumptions of an abstract scheduling model?

Correctness Issues

Although the theory of real-time scheduling can be developed using rigorous mathematics, results are often presented less formally. The assumptions made are not always carefully specified. The proofs are often imprecise and based on intuitive explanations rather than rigorous arguments. As the problems are subtle and complex, there are examples of erroneous results, and incomplete or flawed proofs in the literature.

Even Liu and Layland's essential results rely on many properties that all look intuitively reasonable but are not rigorously proven [4]. More precise proofs of the same results have been given since, but there is an obvious risk in relying on properties whose proof is based on their authors' intuition. This risk is exacerbated when one moves away from the simplest scheduling models. In many systems that are relevant for avionics, process scheduling interacts with other critical mechanisms, such as those supporting inter-process communication, partitioning, or fault-tolerance. In such contexts, our intuition is much more likely to be wrong than in the basic, fixed-priority model.

[1] The tasks with lower period are given higher priority.

For example, Ghosh et al. [7] present a scheduling approach intended to tolerate transient faults by reexecuting faulty tasks. Generalizing Liu and Layland's results, they give conditions under which periodic tasks are feasible using this scheme. In a subsequent paper, the same authors found a flaw in their initial analysis and proposed a different fault-tolerant scheduling scheme [8]. Unfortunately, as discovered by Sinha and Suri [9], their analysis of the new scheme is also flawed; there are task sets that satisfy the conditions in Ghosh et al. [8], but where deadlines are not met if a fault occurs.

This example illustrates the difficulty of correctly analyzing complex scheduling policies using only informal models and proofs. It also shows that the informal review process used in academic research may fail to spot serious errors. The example is also very relevant to the avionics community since the fault-tolerant scheduling scheme of Ghosh et al. has been implemented in an RTOS specifically designed for avionics applications [1]. (The approach might work in this case because the RTOS in question only supports harmonic task sets.)

Validity of Models

Most results in real-time scheduling are developed using very abstract scheduler models. Typically, only a few assumptions are made about the scheduler, such as which job is active when several are ready to execute. To apply theoretical results in critical applications, one needs to ensure that the relevant assumptions are satisfied by the system at hand. This can be difficult as there is usually a large gap between the abstract models used to derive scheduling results and the real scheduling algorithms used in an RTOS. Part of the difficulties is due to the complex interaction between scheduling and other services provided by the RTOS. As a minimum, inter-process communication requires synchronization services that may cause a process to wait and be delayed by other processes. Other features of an RTOS, such as interrupt handling or its clock frequency also have an impact on task execution.

Because of fault-tolerance requirements, RTOS for critical applications may also provide features, such as temporal partitioning [2], that impose constraints on how a processing time is allocated to tasks. Such systems do not follow exactly the simple priority-based allocation policy that is the most commonly studied in theory. Furthermore, because of limitations of the simple approaches, more complex scheduling strategies are being proposed. For example, maximal-urgency-first scheduling [10] is a complex combination of priority-based and deadline-based scheduling. The approach intends to ensure that critical tasks never miss a deadline, while achieving high processor utilization. This new approach has undeniable performance advantages over more classic scheduling strategies and may be adopted by practical systems, but it is also more complex to implement and to analyze. To the best of our knowledge, this approach has been evaluated only empirically [10,11].

Even though many types of scheduling disciplines and task sets have been investigated theoretically, real systems may not always match any of the theoretical models. Still, the literature provides general analysis methodologies that can often be adapted to new contexts. The problem is to obtain high assurance that such general methods have been properly applied to the given system.

A Formal Analysis Methodology for Real-Time Scheduling

The literature on real-time scheduling contains many results that should provide sound bases for the engineering of real-time systems. For such results to be applied in safety-critical applications, one must obtain very high confidence that they are correct. Furthermore, as new scheduling strategies are introduced, there is a need for rapidly adapting scheduling results to complex systems that do not necessarily match traditional models.

As discussed previously, very rigorous approaches are necessary to catch potential errors or omissions in models and in proofs. Formal methods and tool-assisted verification can provide the required degree of precision and rigor, and as discussed in the sequel, can be applied effectively to nontrivial examples. We propose a general methodology for modeling real-time schedulers, proving that they satisfy critical properties, and deriving scheduling properties for different types of tasks. Our objective is an approach that supports both the verification of qualitative properties related to synchronization (e.g., the absence of deadlocks) and the derivation of high-level real-time results (e.g., conditions under which deadlines are met).

To support the approach, we use the PVS theorem prover [12]. In particular, we have developed a large library of basic notions and results that are useful for modeling systems and verifying high-level scheduling properties. This PVS library also introduces the basic concepts that are central to real-time scheduling analysis. Application to a specific scheduling protocol consists of building a state-machine model of the scheduler, examining the execution traces of this model, and relating these traces to the basic notions and results provided by the library.

Support for Timing Analysis

Although scheduling algorithm can vary a lot, many generic notions and basic facts are useful for reasoning about different approaches. The literature also provides some general analysis techniques, such as computing worst-case response times, that are applicable in many contexts. To take advantage of these existing results, we have developed a library of PVS theories that formalize and prove many useful properties.

The library is based on the basic concept of a discrete-time *abstract schedule* that represents the allocation of a single resource (e.g., a processor) to different jobs. Given a set of jobs *J*, a schedule is an infinite sequence

$$u = u_0, u_1, \ldots, u_t, \ldots$$

where u_t indicates which job, if any, owns the resource at time *t*. Either $u_t = j$ and $j \in J$ is the job that is active at time *t*, or $u_t = \bot$ if there is no job active at that time. Analysis of timing properties can then be based on measuring how much time has been allocated to a particular job or sets of jobs during. Given a set $E \subseteq J$, the PVS function

process_time(u, t1, t2, E)

gives the amount of time allocated to jobs of *E* in the interval $[t_1, t_2)$. A similar function gives the amount of idle time in an interval. These basic notions and their properties are implicitly used in many theoretical works on real-time scheduling. A set of PVS theories contains many useful properties of these basic functions.

Another part of our PVS libraries introduces results that support timing analysis via the computation of worst-case response times. For example, we have formalized in PVS a generalization of the algorithm described in Figure 1. These results are very useful since they can be applied in general contexts. In particular they are applicable to many priority-based preemptive schemes.

Modeling Schedulers

To apply the results from the library to a particular system, the first step is to build a model of the scheduling algorithm used. A natural approach is to use state-machine models. Such models have the advantage of providing operational specifications and are easy to understand, thus reducing the gap between implementation and model. There are also well-known verification techniques (some of which can be fully automated) for proving properties of such automata-based models.

It is also important for our purpose to obtain models that conveniently support timing analysis. Since timing results depend as much on task characteristics as on scheduler properties, we need the ability to introduce assumptions about sets of tasks. A common approach is to decompose a task into a succession of jobs, each job being characterized by attributes such as its dispatch time and duration. Timing characteristics of tasks can then be specified as constraints on the length of jobs and the delays between successive jobs.

Our modeling approach follows the same general principles. We build a state-machine model that is parameterized by a fixed but arbitrary collection of jobs. All the job attributes are assumed fixed and known in advance to simplify the modeling. To obtain general models, we assume as little as possible about the jobs. This allows us to obtain results that are valid for very general classes of jobs and then largely independent of the type of tasks considered. For example, synchronization properties that are valid whatever the task characteristics can be established in this way.

For obtaining higher-level scheduling results that depend on the timing characteristics of tasks, the general models can be specialized by restricting attention to jobs that satisfy certain properties. This amounts to instantiating the state-machine model with specific parameters that satisfy adequate assumptions. Analysis of these specialized instances can still rely on the properties that are inherited from the generic model.

From State Machines to Abstract Schedules

Once a state-machine specification is constructed, we can study its real-time properties by examining its execution traces. More precisely, the state-machine model is parameterized by a set of jobs *J* with attributes such as their starting time, their priority, and their duration. The traces of the machine are sequences of state-transitions performed by the system for this particular set of jobs. From such traces, it is trivial to construct abstract schedules that can be analyzed using the support library. As previously, timing properties can be very general or valid only for special instances of the job parameters that satisfy relevant assumptions.

An Example: The Priority-Ceiling Protocol

We have performed a full PVS formalization and verification of a nontrivial scheduling algorithm. This shows that precise and rigorous analysis of fairly complex scheduling approaches can be performed with modern theorem provers. Our case study was the priority-ceiling protocol, introduced by Sha et al. [3]. This protocol is a synchronization algorithm for real-time systems that employ fixed-priority preemptive scheduling. In its basic form, the protocol is used to control access to common resources, such as shared variables or I/O devices, in mono-processor systems. Access to critical sections is controlled via binary semaphores.

This protocol has many interesting properties: it ensures mutual exclusion and absence of deadlocks, and it guarantees that blocking delays are minimal. The latter property is essential in real-time applications. It means that a process *P* requesting access to a resource *S* will wait only for a bounded time before *S* becomes available. The protocol is also important in practice: the priority-ceiling protocol or one of its variant is used in most commercial RTOSs.

As is often the case in the literature, the description of the protocol given by its authors is quite informal and imprecise. The key properties are also proved in a fairly informal manner. As the protocol is subtle and relies on complex priority-adjustment rules, an informal specification is open to misinterpretation. The priority-adjustment rules and the interactions between mutual exclusion and scheduling make the details of the protocol difficult to understand and quite challenging to verify.

Our objective was to develop complete and precise specifications of the protocol, and obtain rigorous proofs that the associated results are valid. We wanted to prove both key synchronization properties such as mutual exclusion and absence of deadlocks, and significant schedulability results. Details of the PVS developments and verifications are available elsewhere [13]. We used the general methodology described previously:

We constructed a parameterized state-machine model of the priority-ceiling protocol that specifies how semaphores are allocated to jobs and how jobs can be activated in a given state. We then proved that all reachable states of the protocol satisfy invariant properties that ensure mutual exclusion, absence of deadlocks, and imply that at most one job can block another. The proofs use standard techniques based on induction.

In a second step, we examined the execution traces and schedules that the protocol model can produce. From the basic state-invariant properties, we obtained general bounds on blocking delays and on the processing time allocated to jobs. These properties are valid for arbitrary collections of jobs and are the basis for further scheduling analysis. The proofs used the notion of schedules and the various results developed in the support library.

We then applied the general results to derive a well-known schedulability criterion for sporadic tasks [5,6]. A sporadic task can be thought of as a sequence of jobs of the same priority, separated by a minimal interarrival delay. Sporadic tasks are useful for modeling both strictly periodic computations and interrupt-driven activities with a limited interrupt frequency. Knowing bounds on the length of each job of a task and the length of critical sections of other jobs, one can compute the worst-case response time for jobs of a task τ_i, using an iterative algorithm similar to the one of Figure 1. From this worst-case execution time, it is easy to determine if a task meet its deadlines.

The case study shows that a full formal specification and verification of nontrivial scheduling mechanisms and properties is feasible. In the same framework, we established both low-level synchronization properties and high-level schedulability properties. All the developments were performed with tool support, using the PVS specification and verification system. This provides very high assurance of correctness and ensures a complete and rigorous analysis. The whole PVS formalization required between two and three person months of effort, part of which was spent on developing the general support libraries. The formalization contains 417 lemmas and theorems, for around 2500 lines of specifications. The support library and other reusable results that are independent of the priority-ceiling protocol represent around 40% of the developments (1000 lines).

In addition to giving very strong evidence that the protocol and the associated scheduling test are correct, the formalization also provided other useful insights:

- Our specification of the protocol is more precise and simpler than the traditional informal description. In particular, the protocol can be entirely specified without any priority adjustment by using a slightly modified rule for selecting active jobs. This may point to new ways of implementing the protocol and was essential for simplifying the verification.
- Some new results emerged from the analysis, such as the fact that the protocol works under weaker assumptions than originally made by its authors. Sha et al. assume that critical section are properly nested [3], but this requirement is actually unnecessary. All the important properties are satisfied even if critical sections overlap.
- Our PVS proof of the schedulability result for sporadic tasks is based on the notion of busy periods, but is simpler and more direct than the classic proofs. Many informal proofs of similar results refer to Liu and Layland's theorem 1, which states that the worst-case response time for a task τ occurs when all tasks of higher priority than τ are dispatched at the same time as τ [4]. The worst-case scenario is then when all tasks are released at the same time, called a critical instant. Informal proofs are usually based on finding the worst-case scenario for a given context and determining response times in this worst case. However, proving that the identified scenario is actually the worst possible case can be difficult[2]. Our proof shows that looking for such worst-case scenarios is actually an unnecessary detour.

Conclusion

A sound theory of real-time scheduling is necessary to support the engineering of real-time software applications. Developing scheduling results using formal methods helps provide the high level of assurance required in safety-critical domains. We propose a formalization methodology relying on state-machine models and on a library of commonly applicable properties. A case study has shown that rigorous and a detailed verification of nontrivial scheduling results can be performed within reasonable time limits, using modern theorem-proving tools such as PVS.

It is still necessary to develop and extend such verification efforts to the increasingly complex scheduling algorithms that may one day be used in safety-critical applications. Other aspects relevant to the avionics domain remain insufficiently explored by the formal methods community. For example, little has been done in the analysis of distributed real-time systems, where difficult problems mixing communication, processing, fault-tolerance must be addressed. Formal methods are especially valuable in such complex settings, where informal proofs guided by intuition are insufficient and where other validation approaches such as testing are incomplete. Formal methods should become an essential tool for validating the most critical aspects of real-time systems, such as the

[2] The proof of theorem 1 given by Liu and Layland is particularly unconvincing.

partitioning mechanisms required for fault isolation in integrated avionics.

Real-time scheduling problems are an ideal application area for formal methods since they are subtle and complex, must be certified to the highest degrees of assurance for supporting critical applications, and thus require very precise, detailed, and rigorous verification.

Acknowledgements

The work described in this paper was partially funded by the Defense Advanced Research Project Agency and by the Air Force Research Laboratory under contract F30602-97-C-0040.

References

[1] Dong, Libin, et al., 1999, Implementation of a Transient-Fault-Tolerance Scheme on DEOS, In *Proceedings of the Real-Time Technology and Application Symposium (RTAS)*, Vancouver, Canada.

[2] ARINC Specification 653, Avionics Application Software Interface, Annapolis, MD.

[3] Sha, L., R. Rakjumar, and J. P. Lehoczky, 1990, Priority Inheritance Protocols: An Approach to Real-Time Synchronization, *IEEE Transactions on Computers*, Vol. 39, No. 9, pp. 1175-1185.

[4] Liu, C. L. and James W. Layland, 1973, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM*, Vol. 20, No 1, pp. 46-61.

[5] Audsley, N. C., A. Burns, M. F. Richardson, and A. J. Wellings, 1991, Hard Real-Time Scheduling: The Deadline Monotonic Approach, *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, GA, pp. 127-132.

[6] Sha, L., R. Rajkumar, and S. Sathaye, 1994, Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems, *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 68-82.

[7] Ghosh, S., R. Melhem, and D. Mossé, 1997, Fault-Tolerant Rate-Monotonic Scheduling, In *Dependable Computing for Critical Applications (DCCA-6)*, IEEE Computer Society.

[8] Ghosh, S., R. Melhem, D. Mossé, and J. Sarma, 1998, Fault-Tolerant Rate-Monotonic Scheduling, *Real-Time Systems*, Vol. 15, Kluwer Academic Publisher, pp. 149-181.

[9] Sinha, P. and N. Suri, 1999, On the Use of Formal Techniques for Analyzing Dependable Real-Time Protocols, In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS-99)*, Phoenix, AZ.

[10] Steward, D. and P. Khosla, 1991, Real-Time Scheduling of Sensor-Based Control Systems, *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, GA, pp. 144-150.

[11] Gill, C., D. Levine, and D. Schmidt, 2000, The Design and Performance of a Real-Time CORBA Scheduling Service, *International Journal of Time-Critical Computing Systems, Special Issue on Real-Time Middleware*, http://www.cs.wustl.edu/~schmidt/dynamic.ps.gz.

[12] Owre, S., J. Rushby, N. Shankar, and F. von Henke, 1995, Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS, *IEEE Transactions on Software Engineering*, Vol. 21, No 2, pp. 107-125.

[13] Dutertre, B., 1999, *The Priority-Ceiling Protocol: Formalization and Analysis using PVS*, Technical Report, System Design Laboratory, SRI International, http://www.sdl.sri.com/dsa/publis/prio-ceiling.html.