Static Analysis of Embedded Real-Time Concurrent Software with Dynamic Priorities

Antoine Miné

Sorbonne Universités, UPMC Univ. Paris 06, CNRS, LIP6, Paris, France

Abstract

In previous work, we developed a sound static analysis by abstract interpretation to check the absence of run-time errors in concurrent programs, focusing on embedded C programs composed of a fixed set of threads in a shared memory. The method is thread-modular: it considers each thread independently, analyzing them with respect to an abstraction of the effect of the other threads, so-called interference, which are also inferred automatically as part of analyzing the threads. The analysis thus proceeds in a series of rounds that reanalyze all threads, gathering an increasing set of interference, until stabilization. We proved that this method is sound and covers all possible thread interleavings. This analysis was integrated into the Astére industrial-scale static analyzer, deployed in avionics and automotive industries.

In this article, we consider the more specific case of programs running under a priority-based real-time scheduler, as is often the case in embedded systems. In such programs, higher priority threads cannot be preempted by lower priority ones (except when waiting explicitly for some resource). The programmer exploits this property to reduce the reliance on locks when protecting critical sections. We show how our analysis can be refined through partitioning in order to take into account the real-time hypothesis, remove spurious interleavings, and gain precision on programs that rely on priorities. Our analysis supports in particular dynamic priorities: we handle explicit modifications of the priorities by the program, as well as implicit ones through the priority ceiling protocol.

We illustrate our construction formally on an idealized language. Following previous work, we first provide a concrete semantics in thread-modular denotational form that is complete for safety properties, and then show how to apply classic abstractions to obtain an effective static analyzer, able to detect all run-time errors, data-races, as well as deadlocks. Finally, we briefly discuss our implementation inside the Astére analyzer and on-going experimentation, with results limited for now to small programs.

Keywords: static analysis, abstract interpretation, verification, safety, concurrency, run-time errors, data-races, deadlocks, real-time scheduling, priority ceiling protocol

1 Introduction

Program verification is an important part of software development, and has a significant cost in industry, hence the need to research more cost-effective methods. It is particularly important to ensure that critical embedded software that control planes or cars are free from errors. Testing, the main established method, is more
and more often supplemented with formal methods. Unlike testing, they can provide strong mathematical guarantees about the behaviors of programs. Semantic-based static analysis by abstract interpretation [12] is particularly attractive: it infers statically properties of the dynamical behaviors of a program, reasons directly on the original, unmodified source (or binary) code, it is fully automated, and it is sound. Soundness is a key property: it states that no behavior of the program is omitted by the analysis, so that any property derived by the analysis is true on all possible executions. In the avionics field, for instance, certification authorities require static analyses to be sound in order to be considered part of the certification process [2]. To scale up, and to produce effective results on what is in general an undecidable problem, abstract interpretation performs approximations which, to maintain soundness, must be conservative and over-approximate the set of possible behaviors. This may result in false alarms: situations where the approximation considers spurious erroneous executions and fails to establish the correctness of a correct program. Nevertheless, through a manual process of specialization of abstractions targeting specific classes of programs and properties to prove, we can hope to achieve a precise and efficient enough analysis, at least on the target programs. For instance, in previous work, we participated to the design of Astrée [6], one such analyzer aiming to prove the absence of run-time errors (such as integer or float overflows, invalid pointer accesses, etc.) on embedded critical sequential C code, and specialized it for control-command avionics software. Specialization took the form of developing new, sophisticated abstract domains, mainly for numeric properties, and heuristic to choose wisely at each program point the desired cost versus expressiveness among to abstractions available. Astrée is being used in an industrial context, initially at Airbus [14] and, following its commercialization by AbsInt [20], in other embedded critical industries, such as the automotive industry.

We consider here the static analysis of concurrent embedded software. On the one hand, more and more software are concurrent, either to exploit the parallel execution offered by current multi-core processors, or simply for ease of programming. This trend also affects critical software. For instance, Integrated Modular Avionics [38] aims at replacing sets of processors, running sequential programs communicating through a bus, with a single processor, running a multi-threaded program in a shared memory implementing the same set of functionalities. On the other hand, concurrent program verification is challenging for traditional methods. Indeed, concurrent programs generally feature a large number of possible executions due to highly non-deterministic control flows, causing test coverage to drop dramatically. Full control and data coverage, i.e., soundness, is nevertheless very important as effective errors often occur in rare but possible situations (e.g., scheduling corner cases or specific interleavings). In such a setting, sound static analysis becomes even more attractive. We have been working on an extension of Astrée to concurrent embedded C software [27]. We report here on a recent improvement concerning the specific case of programs running under a priority-based real-time scheduler.
دریافت فوری متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات