AstréeA: A Static Analyzer for Large Embedded Multi-Task Software

Antoine Miné

AstréeA: A Static Analyzer for Large Embedded Multi-Task Software

Antoine Miné

CNRS & École Normale Supérieure
45, rue d'Ulm
75005 Paris, France
mine@di.ens.fr

Embedded critical systems, such as planes and cars, cannot be easily fixed during missions and any error can have catastrophic consequences. It is thus primordial to ensure the correctness of their controlling software before they are deployed. At the very least, critical embedded software must be exempt from runtime errors, including ill-defined operations according to the specification of the language (such as arithmetic or memory overflows) as well as failure of programmer-inserted assertions. Sound and approximate static analysis can help, by providing tools able to analyze the large codes found in the industry in a fully automated way and without missing any real error. Sound and scalable static analyzers are sometimes thought to be too imprecise and report too many false alarms to be of any use in the context of verification. This claim was disproved when, a decade ago [2], the Astrée static analyzer [1] successfully analyzed the runtime errors in several Airbus control flight software, with few or no false alarm. This result could be achieved by employing abstract interpretation [4], a principled framework to define and compose modular sound-by-construction and parametric abstractions, but also by adopting a design-by-refinement development strategy. Starting from an efficient and easy to design, but rather coarse, fully flow- and context-sensitive interval analyzer, we integrated more complex abstractions (carefully chosen from the literature, such as octagons [10], adapted from it, such as trace partitioning [9], or specifically invented for our needs, such as digital filter domains [6]) to remove large sets of related false alarms, until we reached our precision target.

In this presentation, we discuss our *on-going* efforts towards a similar goal: the efficient and precise sound verification of the absence of run-time errors, but targeting another, more complex class of software: *shared-memory concurrent embedded C software*. Such software are already present in critical systems and will likely become the norm with the generalization of multi-core processors in embedded systems, leading to new challenging demands in verification. Our analyzer is named AstréeA [5], in reference to Astrée on which it takes its inspiration and on the code base of which it elaborates. AstréeA's specialization target is a family of several embedded avionic codes, each featuring a small fixed set of a dozen threads, more than 1.5 Mlines of C code, implicit communication through the shared memory, and running under a real-time OS based on the ARINC 653 specification.

One major challenge is that a concurrent program execution does not follow a fixed sequential order, but one of many interleavings of executions from different tasks chosen by the scheduler. A sound analysis must consider all possible interleavings in order to cover every corner case and race condition. As it is impractical to build a fully flow-sensitive analysis by enumerating explicitly all interleavings, we took inspiration from *thread-modular* methods: we analyze each thread individually, in an environment consisting of (an abstraction of) the effect of the other threads. This is a form of *rely-guarantee* reasoning [8], but in a fully automatic static analysis settings formalized as abstract interpretation. Contrary to Jones' seminal rely-guarantee proof method or its more recent incarnations [7], our method does not require manual annotations: thread interferences are automatically inferred by the analysis (including which variables are actually shared and their possible values). Following the classic methodology of abstract interpretation [4,3], a thread-modular static analysis is now viewed as a computable abstraction of a *complete* concrete thread-modular semantics. This permits a fine control between precision and efficiency, and opens the way to analysis specialization: any given safety property of a given program can be theoretically inferred given the right abstract domain.

Following the design-by-refinement principle of Astrée, our first prototype AstréeA [11] used a very coarse but efficient flow-insensitive and non-relational notion of thread interference: it gathered independently for each variable and each thread an interval abstraction of the values the thread can store into the variable along its execution, and injected these values as non-deterministic writes into other threads. This abstraction allowed us to scale up to our target applications, in efficiency (a few tens of hours of computation) if not in precision (a few tens of thousands alarms).

This presentation will describe our subsequent work in improving the precision of AstréeA by specialization on our target applications, and the interesting abstractions we developed along the way. For instance, we developed new interference abstractions enabling a limited but controllable (for efficiency) degree of relationality and flow-sensitivity [12]. We also designed abstractions able to exploit our knowledge of the real-time scheduler used in the analysis target: *i.e.*, it schedules tasks on a single core and obeys a strict priority scheme.[1] The resulting analysis is less general, but more precise on our target applications, which was deemed necessary as the correctness of the applications relies on these hypotheses on the scheduler.[2] Finally, not all false alarms are caused by our abstraction of concurrency; we also developed numeric and memory domains to handle more precisely some programming patterns which we did not encounter in our previous experience with Astrée and for which no stock abstract domain was available.

[1] The scheduler remains fully preemptive: a low-priority thread may be interrupted at any point by a higher-priority thread whose request to an external resource has just been granted, resulting in a large number of possible thread interleavings.

[2] It is important not to confuse here specialization with restriction: the scheduler abstraction is optional and can be omitted to achieve a more general, but less specialized analysis.

The end-result is a more precise analyzer on our target applications, with currently around a thousand alarms. We stress that AstréeA is a work in progress and that its results, although they are not yet as impressive as those of Astrée, are likely to improve through further specialization. We also believe that, thanks to the intrinsic modularity of the abstract interpretation framework, the analysis performed by AstréeA can be adapted to other settings (other families of applications, other schedulers, other concurrency models) by developing new abstractions, while the abstractions we designed along the journey may also be of use in similar or different static analyses.

References

1. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The Astrée static analyzer. `http://www.astree.ens.fr`.
2. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI 2013*, pages 196–207. ACM, 2003.
3. P. Cousot and R. Cousot. Invariance proof methods and analysis techniques for parallel programs. In *Automatic Program Construction Techniques*, chapter 12, pages 243–271. Macmillan, New York, NY, USA, 1984.
4. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, Aug. 1992.
5. P. Cousot, R. Cousot, J. Feret, A. Miné, and X. Rival. The AstréeA static analyzer. `http://www.astreea.ens.fr`.
6. J. Feret. Static analysis of digital filters. In *ESOP 2004*, volume 2986 of *LNCS*, pages 33–48. Springer, Mar. 2004.
7. C. Flanagan, S. N. Freund, and S. Qadeer. Thread-modular verification for shared-memory programs. In *ESOP 2012*, volume 2305 of *LNCS*, pages 262–277. Springer, 2002.
8. C. B. Jones. Tentative steps toward a development method for interfering programs. *ACM TOPLAS*, 5:596–619, 1983.
9. L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzer. In *ESOP 2005*, volume 3444 of *LNCS*, pages 5–20. Springer, Apr. 2005.
10. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
11. A. Miné. Static analysis of run-time errors in embedded critical parallel C programs. In *ESOP 2011*, volume 6602 of *LNCS*, pages 398–418. Springer, 2011.
12. A. Miné. Relational thread-modular static value analysis by abstract interpretation. In *VMCAI 2014*, volume 8318 of *LNCS*, pages 39–58. Springer, 2014.