Verification of Control System Software

Control systems are typically prototyped with graphical design tools such as Simulink; the actual implementation is then obtained by either compilation from these tools or via other high-level languages such as Scade. All of these steps, including early design, may result in bugs slipping into the end product. These bugs may lead to costly product recalls or, in the case of safety-critical systems (aircraft fly-by-wire, medical infusion pumps, safety-critical industrial processes, etc.), to loss of life and limb.

The traditional way to detect bugs in a computer system is through testing: run the programs or components thereof on sample inputs and check for violations of expected system behaviors—not just program crashes, but also functional properties such as actuator constraint violations and inconsistent mode settings. Although coverage criteria for testing typically guarantee that all instructions of the program have been exercised, testing cannot exercise all possible configuration executions on all possible inputs.

The limitations of testing could be overcome if we could *prove* that a program behaves correctly on all possible inputs. This is the goal of formal validation and verification research, which has resulted in practical tools and successes in this area!

The Airbus A380 has advanced fly-by-wire controls implemented in software. A380 software development benefited from static program analysis tools.

From Testing to Proving

Correctness proofs for programs were proposed in the late 1960s by Floyd and Hoare, but the limited technologies available for automating such proofs long confined them to academic examples and idealized versions of crucial algorithms.

A crucial limitation of automated program analysis is that no analysis algorithm can be guaranteed to never give false negatives (failing to point to bugs) or false positives (bugs that cannot occur in reality). This is a basic mathematical result of computability theory. Thus, all automated analysis methods effect a balance between these two kinds of errors/imprecisions.

Within these theoretical limits, though, practical tools can and have been developed and deployed.

From Academia to Industry

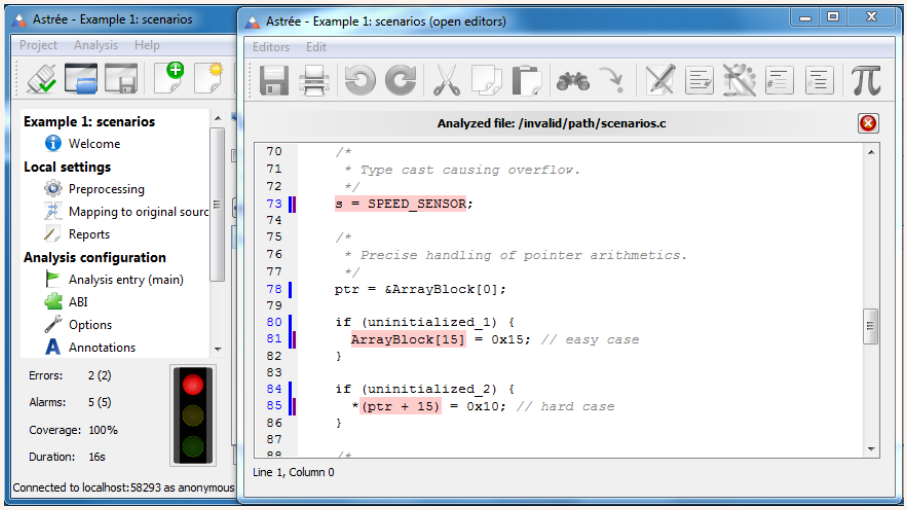
The aerospace community's interest in formal methods was renewed in 1996 by the explosion of the maiden flight of the Ariane 5 rocket due to a software bug (an arithmetic overflow). A team of researchers from INRIA was commissioned to design a static analysis system that could detect such kinds of bugs in future. The academic IABC static analyzer was later turned into the PolySpace verifier. (PolySpace, a startup, was later bought by The MathWorks.)

The Airbus A380 was the next major application of static analysis in aerospace. Researchers from École Normale Supérieure of Paris and CNRS developed new analysis techniques for avionics software (e.g., analyzing floating-point computations such as digital filters). The Astrée tool is now marketed through AbsInt GmbH, which also develops the aiT tool for proving bounds on worst-case execution time on modern embedded processors (with pipelined execution units, caches, etc.).

In the United States, the U.S. Food and Drug Administration (FDA) began an initiative in 2010 enforcing the use of static analyzers for programs running infusion pumps; the misbehavior of such programs may result in the death of patients.

Contributors: David Monniaux, VERIMAG, CNRS, France, and Antoine Miné, LIENS, CNRS, France

From: *The Impact of Control Technology*, 2nd ed., T. Samad and A.M. Annaswamy (eds.), 2014. Available at www.ieeecss.org.



The graphical user interface of the Astrée tool displays program lines that could cause runtime errors and also outputs useful information on the program, such as the range and usage of variables.

The Ariane 5 rocket had to be destroyed on its maiden flight because of a software bug (an arithmetic overflow). This incident renewed interest in formal verification and ultimately resulted in a success story for the technology.

The Astrée Static Analyzer

The Astrée static analyzer takes as input C source code and optional annotations (e.g., range of inputs). After a fully automated analysis, it provides easy-to-understand "traffic-light" indications: a green light for program instructions for which it can prove that no unsafe behavior may occur, a red light for those that it can prove will necessarily result in unsafe behavior if executed, and an orange light for those for which it cannot provide proofs of either safe or unsafe performance.

Some static analysis tools may exhibit false negatives: they may fail to flag possible runtime errors or specification violations. In contrast, Astrée is sound. It performs an exhaustive scan of the control and data space of the program, according to the user-specified inputs and the semantics of the C language (including fine points such as floating-point computations, modular integers, pointer manipulations, and memory layouts). It thus discovers all runtime errors. Such *soundness* of results is often considered to necessarily lead to many false positives (warnings about nonexistent problems), but this is not the case with Astrée when applied to its intended target: safety-critical reactive control code with neither dynamic memory allocation, recursion, nor concurrency. By concentrating on the discovery of runtime errors in such programs, Astrée solves a simpler problem than general-purpose analysis tools, but solves it well.

Astrée is specialized. It is parametrized by a set of abstractions that have been specially tuned for use on embedded control-command software, with a preference for avionic and space software. It includes very specific, mathematically sound analyses for constructs commonly found in such applications (e.g., infinite-impulse-response digital filters or quaternion computations) but not in general-purpose software. Designed to be efficient and precise (few or no false alarms) on these codes, it has also been shown to perform well in other application domains of embedded C software.

Astrée has been successfully applied to the analysis of large industrial codes. In just a few hours, it was able to prove automatically the total absence of runtime error in codes of over 1 million lines. For instance, it analyzed Airbus A380 fly-by-wire control code in 14 hours with no false positives.

For more information on the Astrée tool, visit http://www.astree.ens.fr and http://www.absint.com/astree/.
Also see the companion flyer on "Toward Verifiably Correct Control Implementations" in the Research Challenges section of this volume.