Management Summary

Embedded Flight Software Verification (ESOVER)

Introduction and Background

The Absence of Evidence is not Evidence of Absence

The well-known saying in software engineering, **"Software testing proves the existence of bugs, not their absence."** underlines the difficulties of producing robust and defect-free software within the standard software development lifecycle. Developers commonly try to achieve a high software quality by establishing reasonable software engineering procedures following a Verification and Validation "V&V" Model. To ensure the correct implementation of software requirements, the V&V-Model heavily relies on the design of good unit and integration tests. However, even if all software tests succeed, the software is not guaranteed to be free of defects since the **absence of evidence is not evidence of absence**.

The project ESOVER was a collaboration between Viktor Kunčak and his team at the Lab for Automated Reasoning and Analysis (LARA) at EPFL and Simon Felix and his team at Ateleris GmbH. The project's main objective was to **demonstrate the feasibility of applying formal software verification methods to embedded C-code to generate defect-free flight software components**. The study extended upon the existing formal verification platform Stainless and identified the filesystem of the operational solar instrument STIX as a suitable software module to execute the proof-of-concept.

We address two main limitations of developing defect-free (embedded) software with our approach. One limitation is technology-dependent and inherent to the C-language. The other limitation is process-dependent and due to the fallibility of developers.

The difficulty of writing good C code. C is a powerful and versatile language used for embedded software development. Almost anything can be done in C elegantly and efficiently. However, the performance and flexibility come at a price, and thus, preventing coding errors that lead to buffer overflows or illegal memory operations is challenging.

The difficulty of writing good software tests. With the requirements in mind, it is the responsibility of a software test engineer to write good unit and integration tests that verify the requirements and ensure the correctness of the software. The software test engineer has to anticipate good inputs to the unit and integration tests. However, not knowing all future scenarios and edge cases, it is likely that some combinations of input values and state are overlooked, thus potentially leaving parts of the code untested and vulnerable to failures during operations.

Methodology and Approach

Our Approach to Translating "Stainless" Scala to Embeddable C Code

In our approach, we address the limitations above by formally verifying software using the EPFL Stainless platform. In formal software verification, as illustrated in Figure 1, additional specifications are added to the code, which can be mathematically proven correct, i.e., free of any software defects. If the verification fails, counterexamples are produced by Stainless, which can be used to correct the software and to improve the unit and integration tests.

The Stainless platform accepts Scala code and specifications as input for formal verification. As a high-level programming language, Scala avoids many program safety and reliability limitations we encounter with C. It is by default type-safe and provides higher-level abstractions such as data types, pattern matching, and many more.

With one of the Stainless extensions developed in the frame of the ESOVER project, the verified Scala code can be transpiled, i.e., translated to executable and embeddable C code to be run within the existing STIX flight software.

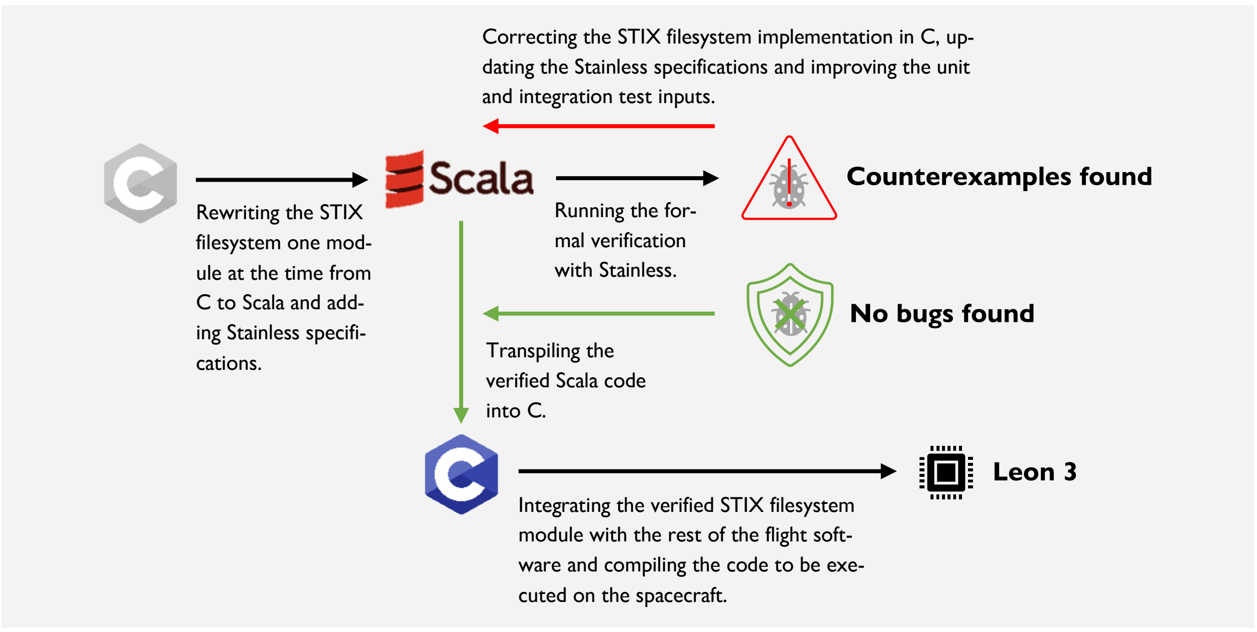
Correcting the STIX filesystem implementation in C, updating the Stainless specifications and improving the unit and integration test inputs.
Scala
Counterexamples found
Rewriting the STIX filesystem one module at the time from C to Scala and adding Stainless specifications.
Running the formal verification with Stainless.
No bugs found
Transpiling the verified Scala code into C.
Leon 3
Integrating the verified STIX filesystem module with the rest of the flight software and compiling the code to be executed on the spacecraft.

Figure 1: The STIX filesystem's formal software verification lifecycle using the Stainless platform and Scala.

Achievements and Results

A Publication and a few Bugfixes in the Operational STIX Filesystem

The LARA team made several extensions to the Stainless-supported Scala version, adding unsigned and varied bit-width machine integers and compilation options to support static and global parameters. These modifications made it feasible to write and verify embedded code in Stainless. The team also substantially improved the C code generator for the Leon processor to enable the integration of generated code into larger C projects, thus allowing for gradually merging the existing STIX filesystem with the verified components.

The Ateleris team rewrote the STIX filesystem in Scala and added specifications for the verification in Stainless. In a joint effort, the software components were verified to guarantee the absence of defects (under the given conditions). Ateleris also gradually integrated the verified components into the STIX filesystem. Some minor bugs in the STIX filesystem were detected during the study, fixed, and uploaded to the instrument during the most recent software upload.

Jad Hamza, Simon Felix, Viktor Kunčak, Ivo Nussbaumer, and Filip Schramka will present their scientific report titled "From Verified Scala to STIX File System Embedded Code using Stainless" at the NASA Formal Methods Conference in May 2022.

Acknowledgments

The ESOVER project was funded by the Swiss Space Office of SERI in the frame of the 2020 "Measure de Positionnement" and supported by the European Space Agency and Space Innovation. We thank all involved parties for the support and the provided collaboration opportunity.