Formal verification of an UAV autopilot

JDD @ DISC

B. Pollien[1], C. Garion[1], G. Hattenberger[2], P. Roux[3], X. Thirioux[1]

June 23, 2023

[1]ISAE-SUPAERO, [2]ENAC and [3]ONERA

Context

The development of a system can be divided into 3 steps:

1. **Specification** of the functional needs and constraints.
2. **Implementation** of the system.
3. **Verification** that the implementation respects the specification.

Verification methods:

- Code reviews,
- Tests,
- Formal methods.

Context

Formal methods

- Verification techniques based on mathematical techniques and tools
- Provides stronger guarantees but with some cost
- Recommended in avionics with DO-178C and DO-333 standards
- Examples: abstract interpretation, deductive methods, model-checking

The goals of my PhD

- Define verification processes that use formal methods,
- Apply these methods to a drone autopilot: Paparazzi.

Paparazzi

Paparazzi is an autopilot for micro-drones

- Developed at ENAC since 2003,
- Open-Source under GPL license.

Complete drone control system:

- Control software part,
- Design of some hardware components,
- Support for ground and aerial vehicles,
- Support for simultaneous control of several drones,
- User can define their own mission using **flight plans**.

Paparazzi as a formal method subject of study?

Paparazzi is a good candidate for testing if formal methods are usable/efficient:

- The autopilot has been developed:
  - without verification purpose,
  - by good programmers,
  - classic C idioms used in the code (pointers etc).
- The code base is consequent ($\sim$ 350k loc).

My PhD focuses on 2 critical components:

- a mathematical library used by the control system,
- a flight plan generator producing embedded C code.

Mathematical Library

Formal Verification of a Mathematical Library

Analysis of a mathematical library of Paparazzi:

- Checking for the absence of runtime errors,
- Verification of some functional properties,
- Without modifying the code.

Verification done using **Frama-C**, a C code analysis tool

- Developed by CEA and Inria,
- Modular, which supports different analysis methods
  ex: static analysis with EVA or dynamic analysis with E-ACSL.

Note: We used the WP or EVA plugins implementing formal methods technics.

Formal verification with Frama-C

Verification process of a C program using Frama-C:

1. Code specification with **ACSL** (*ANSI C Specification Language*),
2. Generation of the abstract syntax tree of the analyzed code,
3. Analysis of the tree by the plugins
   $\Longrightarrow$ Verify if the specification is respected.

Our **goals** were to determine the *minimal functionnal contracts* to guarantee properties:

- No runtime errors: Dereferencing an invalid pointer, division by 0, overflows...
- Functional properties: Offer guarantees on the behavior or the result of a function.

$\Longrightarrow$ Approximately 3,500 lines of annotation to verify 3,000 lines of code.

gitlab.isae-supaero.fr/b.pollien/paparazzi-frama-c

Flight Plan Generator

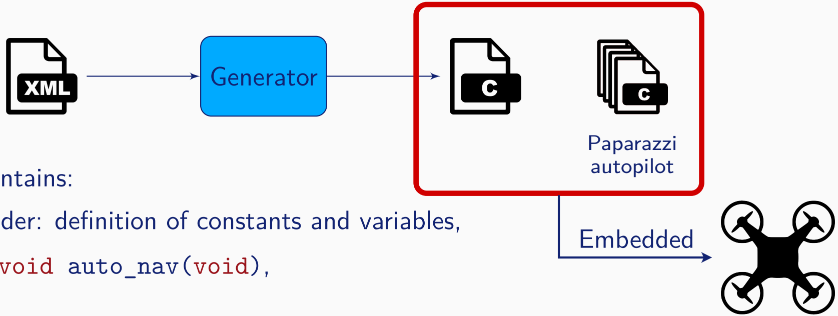
Flight Plan

The flight plan (FP)

- describes how the drone might behave when launched,
- is defined in a XML configuration file.

Example:

1. Wait until the GPS connection is set,
2. Take off,
3. Do a circle around a specific GPS position.
4. If battery is less than 20%: Go *home* and *land*.



The **generated C file** contains:

- The Flight Plan Header: definition of constants and variables,
- The main function: `void auto_nav(void)`,
- Auxiliary functions:
  `pre_call_block`, `post_call_block` and `forbidden_deroute`.

$\Longrightarrow$ Compiled with the autopilot and embedded on the drone.

Function `auto_nav`:

- Called at 20 Hz,
- Sets navigation parameters for actuators.

Motivation

Problems:

- The behaviour of flight plans is not formally defined.
- Does the `auto_nav` function always terminate?
- The generator is a complex software that generates embedded code.

$\Longrightarrow$ **Certified Compilation problem**

Solutions to similar problems

- CompCert: C compiler proved in Coq.
- Vélus: Lustre compiler proved in Coq.

Coq

Coq is a proof assistant

- Developed by Inria,
- Based on the Gallina language.

Software for writing and verifying formal proofs

- Proofs of mathematical theorems,
- Proofs of properties on programs.

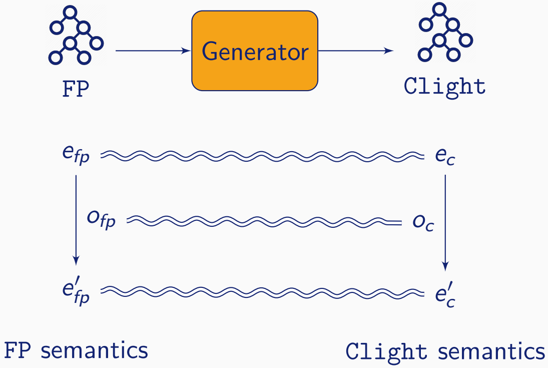
  $\Longrightarrow$ Coq code can be extracted into OCaml code with guarantees.

Our solution: New flight plan generator with

- a minimal front-end in Ocaml,
- a main generator developed and verified in Coq.

Process to develop and verify the new generator

1. **Generator development** in Gallina
   - Input: `FP`
   - Ouput: `Cligt` from CompCert
2. **Formalisation** of the `FP` semantics
3. Use the already defined semantics of `Clight`
4. **Prove** the semantics preservation property



$\Longrightarrow$ **1.3k loc of Ocaml and 17k loc of Coq (12% of working code)**

Conclusion

Formal verification of an UAV autopilot

Study case: Paparazzi

Work done:

- Technical report:
  - Formal verification for autopilots: preliminary state of the start
  - A gentle introduction to C code verification using the Frama-C platform
- Verification of some parts of Paparazzi mathematical library

  Publications: AFADL 2021, FMICS 2021
- Development of a verified flight plan generator

  Publications: FormaliSE 2023

Current work:

- Writting my thesis