Verifying the Mathematical Library of a UAV Autopilot with Frama-C

FAC 2021

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

October 14, 2021

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

Context

Formal methods

- Verification techniques based on mathematical models
- Recommended in avionics with DO-178C and DO-333 standards
- Example: abstract interpretation, deductive methods, model-checking

Context

Formal methods

- Verification techniques based on mathematical models
- Recommended in avionics with DO-178C and DO-333 standards
- Example: abstract interpretation, deductive methods, model-checking

The goals of this ongoing work

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

First results

Analysis of a mathematical library of Paparazzi:

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

Frama-C

Frama-C

Frama-C is 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.

Frama-C

Software Analyzers

Frama-C is 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.

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.

Note: the tree analysis can be performed by several plugins.

Some Frama-C plugins

RTE (*RunTime Errors*):

- Adds assertions in the code,
- Allows to verify runtime errors
  ex: division by 0, overflows ...

Some Frama-C plugins

RTE (*RunTime Errors*):

- Adds assertions in the code,
- Allows to verify runtime errors
  ex: division by 0, overflows ...

WP (*Weakest Precondition*)

- Implements weakest precondition calculus,
- Interfaced with Why3 to verify goals with automatic provers (Alt-Ergo, Z3, CVC4).

Some Frama-C plugins

RTE (*RunTime Errors*):

- Adds assertions in the code,
- Allows to verify runtime errors
  ex: division by 0, overflows ...

WP (*Weakest Precondition*)

- Implements weakest precondition calculus,
- Interfaced with Why3 to verify goals with automatic provers (Alt-Ergo, Z3, CVC4).

EVA (*Evolved Value Analysis*)

- Based on static analysis by abstract interpretation methods,
- Computes domains of values for each variable in the program.

Paparazzi

Presentation

Paparazzi is an autopilot for micro-drones

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

Presentation

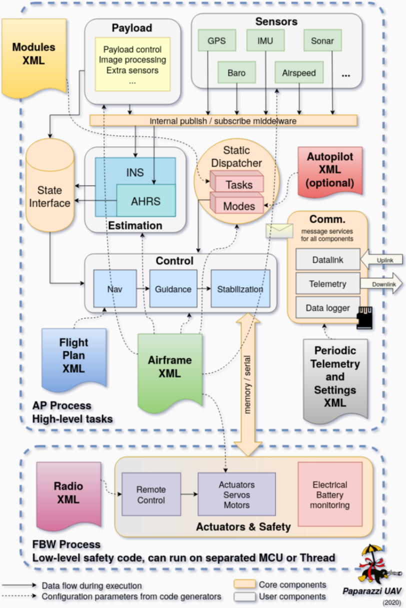
Paparazzi is an autopilot for micro-drones

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

Complete drone control system:

- Offers the control software part,
- Also offers some designs of hardware components,
- Supports for ground and aerial vehicles,
- Supports for simultaneous control of several drones.

Flight system architecture



Library studied

`pprz_algebra` : mathematical algebra library coded in C ($\sim$ 4 000 loc)

The library contains:

- The definition of a representation of `vectors`,
- Different representations of vector rotations,
  rotation matrices, Euler angles, quaternions
- Elementary operations,
  ex: addition of vectors, computation of the rotation of a vector, normalization of a quaternion . . .
- Conversion functions between these different representations.

Note: Each representation/function has a fixed point (`int`) and floating-point version (for `float` and `double`).

Absence of runtime errors

Absence of runtime errors

There are different types of runtime errors:

- Dereferencing an invalid pointer,
- Division by 0,
- Overflows,
- Non finite float value,
- ...

Absence of runtime errors

There are different types of runtime errors:

- Dereferencing an invalid pointer,
- Division by 0,
- Overflows,
- Non finite float value,
- ...

Goals: To determine the minimum contracts for the functions of the library in order to guarantee the absence of runtime errors.

Absence of runtime errors

There are different types of runtime errors:

- Dereferencing an invalid pointer,
- Division by 0,
- Overflows,
- Non finite float value,
- ...

Goals: To determine the minimum contracts for the functions of the library in order to guarantee the absence of runtime errors.

Process :

- Analyze the code with Frama-C using RTE and WP plugins.
- Deduce the missing information in the contract.

Analysis with Frama-C and the RTE plugin

Analysis of the instruction:

```
c->x = a->x * b->x;
```

Analysis with Frama-C and the RTE plugin

Analysis of the instruction:

```
c->x = a->x * b->x;
```

Frama-C finds 2 potential errors!

Analysis with Frama-C and the RTE plugin

Analysis of the instruction:

```
c->x = a->x * b->x;
```

Frama-C finds 2 potential errors!

- Pointers might not be valid.

```
/*@ assert rte: mem_access: \valid(&c->x); */
/*@ assert rte: mem_access: \valid_read(&a->x); */
/*@ assert rte: mem_access: \valid_read(&b->x); */
```

Analysis with Frama-C and the RTE plugin

Analysis of the instruction:

```
c->x = a->x * b->x;
```

Frama-C finds 2 potential errors!

- Pointers might not be valid.

```
/*@ assert rte: mem_access: \valid(&c->x); */
/*@ assert rte: mem_access: \valid_read(&a->x); */
/*@ assert rte: mem_access: \valid_read(&b->x); */
```

$\Longrightarrow$ Require the validity of pointers as a precondition.

Analysis with Frama-C and the RTE plugin

Analysis of the instruction:

```
c->x = a->x * b->x;
```

Frama-C finds 2 potential errors!

- Pointers might not be valid.

```
/*@ assert rte: mem_access: \valid(&c->x); */
/*@ assert rte: mem_access: \valid_read(&a->x); */
/*@ assert rte: mem_access: \valid_read(&b->x); */
```

$\Longrightarrow$ Require the validity of pointers as a precondition.

- The values are not bounded.

```
/*@ assert rte: signed_overflow: -2147483648 ≤ a->x * b->x; */
/*@ assert rte: signed_overflow: a->x * b->x ≤ 2147483647; */
```

$\Longrightarrow$ Determine bounds which guarantee the absence of overflows.

Example of final contract for the function `int32_quat_comp`

```c
#define SQRT_INT_MAX4 23170 // 23170 = SQRT(INT_MAX/4)

/*@
  requires \valid(a2c);
  requires \valid_read(a2b);
  requires \valid_read(b2c);
  requires bound_Int32Quat(a2b, SQRT_INT_MAX4);
  requires bound_Int32Quat(b2c, SQRT_INT_MAX4);
  requires \separated(a2c, a2b) && \separated(a2c, b2c);
  assigns *a2c;
*/
void int32_quat_comp(struct Int32Quat *a2c,
                     struct Int32Quat *a2b,
                     struct Int32Quat *b2c)
```

Notes

EVA and WP had to be associated to verify the absence of RTE.

- WP is overloaded when accessing values by reference,
- EVA cannot verify loop variants and invariants.

$\Longrightarrow$ The same problem has been raised in the thesis of V. Todorov.

Notes

EVA and WP had to be associated to verify the absence of RTE.

- WP is overloaded when accessing values by reference,
- EVA cannot verify loop variants and invariants.

$\Longrightarrow$ The same problem has been raised in the thesis of V. Todorov.

The `real` **arithmetic model** (real in the mathematical sense) has been used to verify floating-point version of the functions.

The `real` model guarantees :

- The absence of division by 0,
- The lack of dereference of invalid pointers.

But the absence of overflows is not verified.

Functional verification

Functional verification

Offer guarantees on the behavior or the result of a function.

Example: *Functional properties for square root function*

```
/*@
  requires x >= 0;
  ensures \result >= 0;
  ensures \result * \result == \old(x);
  assigns \nothing;
*/
float sqrt(float x);
```

Functional verification

Functional verification

Offer guarantees on the behavior or the result of a function.

Example: *Functional properties for square root function*

```
/*@
  requires x >= 0;
  ensures \result >= 0;
  ensures \result * \result == \old(x);
  assigns \nothing;
*/
float sqrt(float x);
```

Note : Verifying these properties is only possible with the *real* model.

Functional verification

Functional verification

Offer guarantees on the behavior or the result of a function.

Example: *Functional properties for square root function*

```
/*@
  requires x >= 0;
  ensures \result >= 0;
  ensures \result * \result == \old(x);
  assigns \nothing;
*/
float sqrt(float x);
```

Note : Verifying these properties is only possible with the *real* model.

The functional verification was only done for some floating-point functions.

How to specify the functional properties?

How to specify the functional properties?

Functional properties must be expressed in the ACSL logic.

How to specify the functional properties?

Functional properties must be expressed in the ACSL logic.

First, it is necessary to define:

- **Types**,
  ex : `RealVect3`, `RealRMat`, `RealQuat`.
- **Elementary functions**,
  ex: addition of vectors, rotation of a vector...
- **Conversion functions** between certain representations,
  ex: Definition of the function `rmat_of_quat` : $\mathbb{H} \rightarrow M_{3,3}(\mathbb{R})$,

```
/*@
    logic RealRMat l_RMat_of_FloatQuat(struct FloatQuat *q) =
    [...]
*/
```

How to specify the functional properties?

Functional properties must be expressed in the ACSL logic.

First, it is necessary to define:

- **Types**,
  ex : `RealVect3`, `RealRMat`, `RealQuat`.
- **Elementary functions**,
  ex: addition of vectors, rotation of a vector...
- **Conversion functions** between certain representations,
  ex: Definition of the function `rmat_of_quat` : $\mathbb{H} \rightarrow M_{3,3}(\mathbb{R})$,

```
/*@
   logic RealRMat l_RMat_of_FloatQuat(struct FloatQuat *q) =
    [...]
*/
```

- **Lemmas...**

Specifying the functional properties of the library

Lemmas: Verify that the mathematical definitions are correct.

Ex: The conversion function produces the same rotation,

Specifying the functional properties of the library

Lemmas: Verify that the mathematical definitions are correct.
Ex: The conversion function produces the same rotation,

- Mathematically,

$$\forall q \in \mathbb{H},\ \forall v \in \mathbb{R}^3,\quad q(0,v)q^* = (0, \texttt{rmat\_of\_quat}(q).v)$$

Specifying the functional properties of the library

Lemmas: Verify that the mathematical definitions are correct.
Ex: The conversion function produces the same rotation,

- Mathematically,

$$\forall q \in \mathbb{H},\ \forall v \in \mathbb{R}^3, \quad q(0, v)q^* = (0, \texttt{rmat\_of\_quat}(q).v)$$

Finally, the functional properties are expressed in the form of predicates:

- M is a rotation matrix: $M.M^t = I$
- ...

Example

Spécification of the function `float_rmat_of_quat`.

```
/*@
    requires \valid(rm);
    requires \valid_read(q) && finite_FloatQuat(q);
    requires unitary_quaternion(q);
    requires \separated(rm, q);
    ensures rotation_matrix(l_RMat_of_FloatRMat(rm));
    ensures special_orthogonal(l_RMat_of_FloatRMat(rm));
    ensures l_RMat_of_FloatRMat(rm) ==  l_RMat_of_FloatQuat(q);
    assigns *rm;
*/
void float_rmat_of_quat(struct FloatRMat *rm,
                        struct FloatQuat *q)
```

Problems faced

Contracts of the trigonometric functions from the libc do not provide mathematical results.

Problems faced

Contracts of the trigonometric functions from the libc do not provide mathematical results.

$\Longrightarrow$ Extend the contracts.

ex: Extension of the contract for the function $sinf$.

```
/*@ requires finite_arg: \is_finite(x);
    assigns \result \from x;
    ensures finite_result: \is_finite(\result);
    ensures result_domain: -1. <= \result <= 1.;
    ensures result_value: \result == \sin(x);
*/
extern float sinf(float x);
```

Problems faced

Contracts of the trigonometric functions from the libc do not provide mathematical results.

$\Longrightarrow$ Extend the contracts.

ex: Extension of the contract for the function `sinf`.

```
/*@ requires finite_arg: \is_finite(x);
    assigns \result \from x;
    ensures finite_result: \is_finite(\result);
    ensures result_domain: -1. <= \result <= 1.;
    ensures result_value: \result == \sin(x);
*/
extern float sinf(float x);
```

Some lemmas could not be proved by the SMT solvers.

Problems faced

Contracts of the trigonometric functions from the libc do not provide mathematical results.

$\Longrightarrow$ Extend the contracts.

ex: Extension of the contract for the function `sinf`.

```
/*@ requires finite_arg: \is_finite(x);
    assigns \result \from x;
    ensures finite_result: \is_finite(\result);
    ensures result_domain: -1. <= \result <= 1.;
    ensures result_value: \result == \sin(x);
*/
extern float sinf(float x);
```

Some lemmas could not be proved by the SMT solvers.

$\Longrightarrow$ Enable interactive mode of Frama-C to use Coq.

Summary of the functional verification

Functional verification offers guarantees on the behavior of a function.

- Functional properties are expressed using ACSL.
- The verification can be done automatically or interactively.

Using the `real` model:

- Offers no functional guarantee during execution.
- Used to verify that the code is correct in a mathematical sense,

Conclusion

Conclusion

Summary :

- Verification of the absence of runtime errors in the library,
- Verification of functional properties on some floating-point functions.

$\Longrightarrow$ Approximately 3,500 lines of annotation.

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

Perspectives:

- Verification of calls to library functions,
- Verifying the floating-point library without the `real` model,
- Verifying the Paparazzi flight plan generator.

This work is supported by the Defense Innovation Agency (AID) of the French Ministry of Defense (research project CONCORDE N 2019 65 0090004707501)

Thank you

References i

S. Sarabandi and F. Thomas.
Accurate computation of quaternions from rotation matrices.
In J. Lenarcic and V. Parenti-Castelli, editors, *Advances in Robot Kinematics 2018*, pages 39–46, Cham, 2019. Springer International Publishing.

S. W. Shepperd.
Quaternion from rotation matrix.
Journal of Guidance and Control, 1(3):223–224, 1978.

V. Todorov.
Automotive embedded software design using formal methods.
Phd thesis, Université Paris-Saclay, Dec. 2020.

Examples of lemmas proved with Coq

- Lemma to verify the correctness of the function `quat_of_rmat`

$$\forall R \in SO_3(\mathbb{R}),\ \forall q \in \mathbb{H},$$
$$||q|| > 0 \ \wedge\ Tr(R) > 0$$
$$\rightarrow (R = \texttt{rmat\_of\_quat}(q) \leftrightarrow q = \texttt{quat\_of\_rmat}(R))$$

Examples of lemmas proved with Coq

- Lemma to verify the correctness of the function `quat_of_rmat`

$$\forall R \in SO_3(\mathbb{R}),\ \forall q \in \mathbb{H},$$
$$||q|| > 0 \ \wedge\ Tr(R) > 0$$
$$\rightarrow (R = \texttt{rmat\_of\_quat}(q) \leftrightarrow q = \texttt{quat\_of\_rmat}(R))$$

- Lemma used to verify that `rmat_of_euler` compute rotation matrix:

$$\forall a, b, c \in \mathbb{R},$$
$$\sin(a)^2 \cos(b)^2$$
$$+ (\sin(a)\sin(b)\cos(c) - \sin(c)\cos(a))^2$$
$$+ (\cos(c)\cos(a) + \sin(a)\sin(b)\sin(c))^2 = 1$$