

A Provably Correct Floating-Point Implementation of Well Clear Avionics Concepts

Mariano M. Moscato (AMA)

Jointly with: Nikson Bernardes Fernandes Ferreira (UnB)
Laura Titolo (AMA)
Mauricio Ayala-Rincón (UnB)

AMA: Analytical Mechanics Associates Inc., Hampton, VA, USA
UnB: University of Brasilia, Brasilia, Brazil

FMCAD 2023

Motivation

- Floats are the most widely used representation of real numbers
- Problems:
 - **Round-off errors**
 - Runtime-exceptions (division by zero, not a number, etc.)
- Writing correct floating-point code is tricky
- Particularly dangerous for **safety-critical** systems

Motivation

- Round-off error accumulation:

```
>>> (4/3 - 1) * 3 - 1
```

```
.
```

```
>>> (4/3 - 1) * 3 - 1  
-2.220446049250313e-16
```

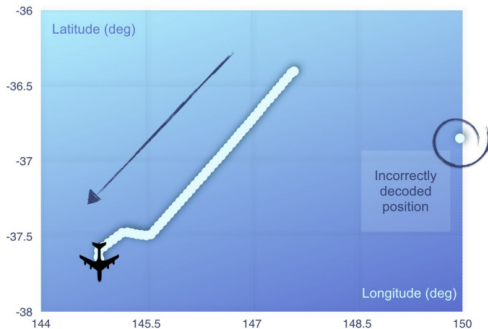
```
>>>
```

Unexpected Numerical Errors in Critical Systems

- Phased Array TRacking Intercept Of Target - PATRIOT missile (1991)
- ADS-B Compact Reporting Position Algorithm (2007)



*



*<https://api.army.mil/e2/c/images/2022/08/23/8fa857ff/size0-full.jpg>

Work of the US government

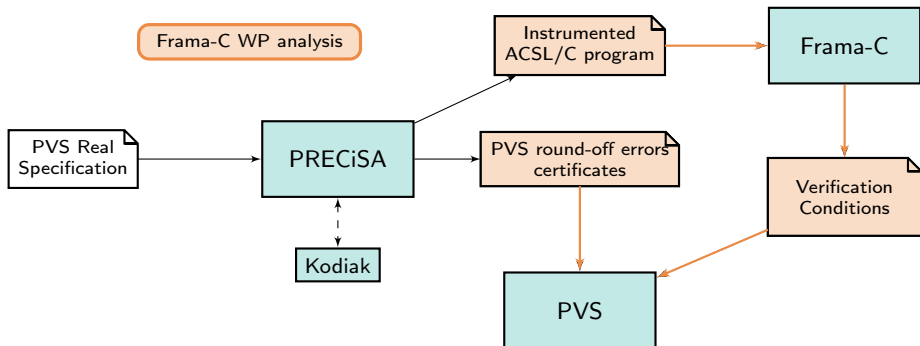
Daidalus

- Detect and Avoid (DAA) concept emerged as an effort to support the integration of UAVs into civil airspace
- **D**etect and **A**void **A**lerting **L**ogic for **U**nmanned **S**ystems
 - Correctness and safety properties formally assured
 - PVS theorem prover
 - Reference implementation — RTCA/FAA MOPS DO365
 - Main features: Conflict Detection, Maneuver Guidance, and Alerting
- Numerical issues may affect the computed result
- A testing-based approach was used to check the implementation
- DAIDALUS requires a higher level of assurance

Our Approach

- Automatic extraction of C code from the ideal real number specification
- Features of the extracted code:
 - Instrumented to detect divergences w.r.t. the *ideal* control-flow
 - Annotated with contracts on the rounding error
 - Externally verifiable
- We use a combination of different formal-methods techniques
 - PVS Interactive theorem prover
 - PRECiSA analyzer and code generation for floating-point functions
 - FRAMA-C collaborative framework for the static analysis of C programs

Integrated Toolchain[†]



- Formal guarantees on the rounding-error
| specification - FP implementation | $\leq \epsilon$ computed by PRECiSA
- Control flow divergence \Rightarrow warning
- Automatic (no expertise on theorem proving or FP required)

[†]Titolo, Moscato, Feliu, and Muñoz, *Automatic Generation of Guard-Stable Floating-Point Code Integrated Formal Methods* (iFM 2020).

Can This Toolchain be Applied to DAIDALUS?

- Not as is!
- Complexity of the Well-clear specification:
 - Higher-order features
 - Intensive use of predicates
 - Number of function calls
- We could not use PRECiSA directly for the Well-Clear library

Our Solution

- Extending the integrated toolchain introduced in iFM2020
- Restating specification using only first-order constructs
- Conditionally slicing the specification to make it manageable to PRECiSA

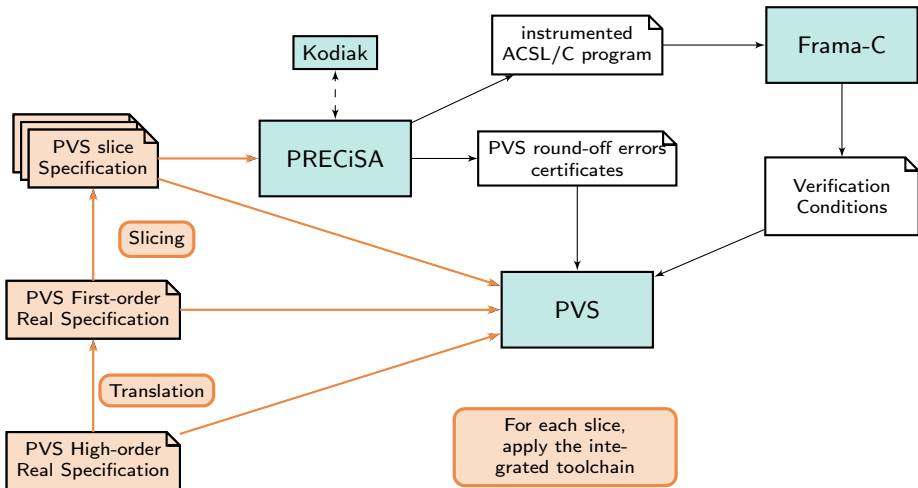
Program Slicing

```
double f1(double x){  
    return (x > 0)? f2(x / 3) : f3(x);  
}
```

```
//@ requires: x > 0  
double f1_gt_0(double x){  
    return f2(x / 3);  
}
```

```
//@ requires: x <= 0  
double f1_lte_0(double x){  
    return f3(x);  
}
```

Integrated Toolchain + Slicing



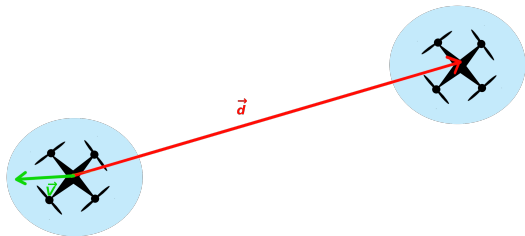
Case Study: DAIDALUS

- Mathematical definition of *Well Clear Violation Volume* of an aircraft with respect to surrounding traffic



Case Study: DAIDALUS

- The Well Clear condition guarantees a collision-free situation
- Correctness proven assuming real-valued arithmetic
- Relies on the relative distance, time variables, and respective thresholds, that is, a minimum space-time distance

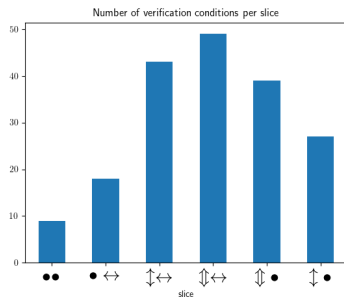
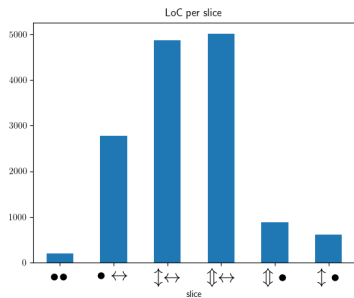


Case Study: DAIDALUS

The Well-Clear module was manually split into six slices pivoting on the relative velocity of the ownship and the intruder

- • aircraft horizontal and vertical separation unmodified;
- ↕ • aircraft vertical separation increases, horizontal separation unmodified;
- ↑ • aircraft vertical separation decreases, horizontal separation unmodified;
- ↔ aircraft alter only horizontal separation;
- ↕ ↔ aircraft increase vertical and alter horizontal separation;
- ↑ ↔ aircraft decrease vertical and alter horizontal separation.

Case Study: DAIDALUS



Case Study: DAIDALUS

- Time of closest point of approach*

```
tcpa( $s_x$ ,  $v_x$ ,  $s_y$ ,  $v_y$ ) =  
  if ( $v_x^2 + v_y^2 \neq 0$ ) then  $-(s_x*v_x + s_y*v_y)/(v_x^2 + v_y^2)$   
  else 0
```

*Muñoz, Narkawicz, Hagen, Upchurch, Dutle, Consiglio, and Chamberlain, *DAIDALUS: Detect and Avoid Alerting Logic for Unmanned Systems* (DASC 2015)

Case Study: DAIDALUS

```

/*@
real tcpa(real s_x, real v_x, real s_y, real v_y) =
  v_x^2 + v_y^2 > 0 ? -(s_x * v_x + s_y * v_y) / (v_x^2 + v_y^2) : 0;

double tcpa(double s_x, double v_x, double s_y, double v_y) =
  v_x^2 + v_y^2 > 0 ? -(s_x * v_x + s_y * v_y) / (v_x^2 + v_y^2) : 0;

predicate tcpa_stable_paths(real s_x, real v_x, real s_y, real v_y,
  double s_x, double v_x, double s_y, double v_y) =
  v_x^2 + v_y^2 > 0 ∧ v_x^2 + v_y^2 > 0
  ∨ ¬(v_x^2 + v_y^2 > 0) ∧ ¬(v_x^2 + v_y^2 > 0);

requires: 0 ≤ ε ∧ finite?(ε) ;
ensures: result ≠ ω ⇒ result = tcpa
ensures: result ≠ ω ⇒ (|(v_x^2 + v_y^2) - (v_x^2 + v_y^2)| ≤ ε ⇒ tcpa_stable_paths)
*/
double tcpa_fp(double s_x, double v_x, double s_y, double v_y, double ε){
  if(v_x^2 + v_y^2 > ε){
    return -(s_x * v_x + s_y * v_y) / (v_x^2 + v_y^2);
  }
  else{
    if(v_x^2 + v_y^2 ≤ -ε)
      return 0;
    else
      return ω;
  }
}

```

- PRECiSA output:
stable paths contract in
ACSL (symbolic)

Case Study: DAIDALUS

- PRECiSA output: *stable paths* contract in ACSL (numeric)

```

/*@
ensures (∀ real sx, sy, vx, vy;
  1 < vx < 1000 ∧ 1 < vy < 1000 ∧ 1 < sx < 1000 ∧ 1 < sy < 1000 ∧
  |⌊sx⌋ - sx| ≤ ulp(sx) / 2 ∧ |⌊sy⌋ - sy| ≤ ulp(sy) / 2 ∧
  |⌊vx⌋ - vx| ≤ ulp(sx) / 2 ∧ |⌊vy⌋ - vy| ≤ ulp(vy) / 2
  ⇒ |(result - tcpa(sx, vx, sy, vy)| ≤ 9.106570e - 07)| ;
*/
double' tcpa_num(double ⌊sx⌋, double ⌊vx⌋, double ⌊sy⌋, double ⌊vy⌋) {
  return tcpa_fp (sx, vx, sy, vy, 4.602043e-10);
}

```

Case Study: DAIDALUS

- PRECiSA output: Predicate

```

/*@
predicate WCV_interval↓.(real b, t, sx, sy, sz, vx, vy, vz) = ....
predicate WCV_interval↓.-fp(double b, t, sx, sy, sz, vx, vy, vz) = ....;

ensures: ∀ real b, t, sx, sy, sz, vx, vy, vz;
  \result ≠ ω ∧ \result
  => WCV_interval↓.(b, t, sx, sy, sz, vx, vy, vz) ∧
      WCV_interval↓.-fp(b, t, sx, sy, sz, vx, vy, vz)
*/
bool' WCV_interval↓.-plus(double b, t, sx, sy, sz, vx, vy, vz, e1, e2, e3) {...}

/*@

ensures: ∀ real b, t, sx, sy, sz, vx, vy, vz;
  \result ≠ ω ∧ \result
  => ¬ WCV_interval↓.(b, t, sx, sy, sz, vx, vy, vz) ∧
      ¬ WCV_interval↓.-fp(b, t, sx, sy, sz, vx, vy, vz)
*/
bool' WCV_interval↓.-mns(double b, t, sx, sy, sz, vx, vy, vz, e1, e2, e3) {...}

```

Case Study: DAIDALUS

- Top-Layer

```

/*@
predicate wcv_in_range(real b, t, s_x, s_y, s_z, v_x, v_y, v_z) =
  // WCV?((b, t), (v_x, v_y, v_z), (s_x, s_y, s_z)) previously defined

requires: \is_finite( $\boxed{e_0}$ )  $\wedge$   $\boxed{e_0} > 0$   $\wedge$  ...  $\wedge$  \is_finite( $\boxed{e_n}$ )  $\wedge$   $\boxed{e_n} > 0$ ;
ensures:  $\forall$  real  $b, t, s_x, s_y, s_z, v_x, v_y, v_z$ ;
   $\boxed{\delta - v_z * \delta_{tcoa}} - \delta - v_z * \delta_{tcoa} < \boxed{e_0} \wedge$ 
   $\boxed{(t - coalt\_t\_asc\_fp(s_z, v_z)) - (t - coalt\_t\_asc\_fp(s_z, v_z))} < \boxed{e_1} \wedge$ 
  ...
  \result  $\neq \omega$ 
   $\implies$  (\result  $\iff$  wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double  $\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3, \dots}$ ) {
  bool' res;
  if ( $\boxed{v_z > 0}$ ) { // increasing vertical separation
    if ( $\boxed{v_x == 0.0} \wedge \boxed{v_y == 0.0}$ ) { // maintaining horizontal separation
      res = WCV_int $\boxed{, \_plus}(b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3)$ ;
      if (res ==  $\omega$   $\vee$  res) return res;
      res = WCV_int $\boxed{, \_minus}(b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3)$ ;
      if (res ==  $\omega$ ) return  $\omega$ ;
      if (res) return false;
      return  $\omega$ ;
    } else { // altering horizontal separation
      ...
    }
  } else if ( $\boxed{v_z < 0}$ ) { // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
    ...
  }
}

```

Case Study: DAIDALUS

- Frama-C/WP output: *stable-paths* verification condition in PVS

tcpa_fp_stable_paths_ensures: THEOREM

FORALL (v_x, v_y : real, $\epsilon, s_x, s_y, v_x, v_y$: double, result : double'):

($0 \leq \epsilon$) => result $\neq \omega$ =>

$|v_x * v_x + v_y * v_y - (v_x^2 + v_y^2)| \leq \epsilon$ =>

(finite?(ϵ)) =>

(IF $\epsilon < v_x^2 + v_y^2$

THEN result = $-(s_x * v_x + s_y * v_y) / (v_x^2 + v_y^2)$

ELSE (result = 0 AND $v_x^2 + v_y^2 \leq -\epsilon$ ENDIF) =>

p_tcpa_stable_paths(v_x, v_y, v_x, v_y)

```
requires:  $0 \leq \epsilon \wedge \text{finite?}(\epsilon)$  ;
ensures: result  $\neq \omega \implies (|v_x^2 + v_y^2 - (v_x^2 + v_y^2)| \leq \epsilon$ 
=> tcpa_stable_paths)
*/
```

Case Study: DAIDALUS

- Frama-C/WP output: *structure* verification condition in PVS

```

tcpa_fp_structural_ensures: THEOREM
FORALL ( $\epsilon, s_x, s_y, v_x, v_y$  : double, result : double'):
( $0 \leq \epsilon$ ) => result  $\neq \omega$  =>
(finite?( $\epsilon$ ) =>
(IF  $\epsilon < v_x^2 + v_y^2$ 
THEN result =  $-(s_x * v_x + s_y * v_y) / (v_x^2 + v_y^2)$ 
ELSE (result = 0 AND  $v_x^2 + v_y^2 \leq -\epsilon$  ENDIF) =>
result = 1_tcpa_fp( $s_x, s_y, v_x, v_y$ )))

```

```

requires:  $0 \leq \epsilon \wedge \text{finite?}(\epsilon)$  ;
ensures: result  $\neq \omega \implies \text{result} = \text{tcpa}$ 

```

Case Study: DAIDALUS

- Frama-C/WP output: *numerical* verification condition in PVS

tcpa_num_ensures: **THEOREM**

FORALL (s_x, v_x, s_y, v_y :**real**, $\boxed{\epsilon, s_x, s_y, v_x, v_y}$:**double**, **result**:**double**):

result $\neq \omega \Rightarrow$

$1 < v_x < 1000 \Rightarrow 1 < v_y < 1000 \Rightarrow 1 < s_x < 1000 \Rightarrow 1 < s_y < 1000 \Rightarrow$

$\boxed{s_x} - s_x \leq \text{ulp}(s_x) / 2 \Rightarrow \boxed{s_y} - s_y \leq \text{ulp}(s_y) / 2 \Rightarrow$

$\boxed{v_x} - v_x \leq \text{ulp}(s_x) / 2 \Rightarrow \boxed{v_y} - v_y \leq \text{ulp}(v_y) / 2$

$\boxed{\text{result} = \text{l_tcpa_fp}(s_x, s_y, v_x, v_y)} \Rightarrow$

(**FORALL** (v_x, v_y :**real**):

$|\boxed{v_x * v_x + v_y * v_y} - (v_x^2 + v_y^2)| \leq$

$(8901646138474497 / 19342813113834066795298816)) \Rightarrow$

$\text{p_tcpa_stable_paths}(v_x, v_y, \boxed{v_x, v_y})) \Rightarrow$

$|\text{result} - \text{tcpa}(s_x, v_x, s_y, v_y)| \leq$

$(4300455909721841 / 4722366482869645213696$

return tcpa_fp (sx, vx, sy, vy, 4.602043e-10);

$\Rightarrow |(\text{result} - \text{tcpa}(s_x, v_x, s_y, v_y))| \leq 9.106570e-07) | ;$

Conclusions and Future Work

- ✔ Analysis of the whole DAIDALUS' *Well Clear* module
- ✔ Slicing made the process manageable by the toolchain and simplified the annotations on the code, producing simpler verification conditions
- ✔ We discovered several issues and opportunities for improvement
- ⚙ We used a new floating-point formalization
 - Improved the efficiency of the analysis greatly
 - Added support for special values
 - But impacted the existing proof strategies
- ▶▶ Improve automation degree:
 - 👍 Automatic slicing and equivalence lemmas definition
 - 👍 Develop more and maintain PVS strategies

Thanks for your attention!

Download PRECiSA

- FP Analyzer: <https://github.com/nasa/PRECiSA>
- Code generator (ReFlow): <https://github.com/nasa/reflow>

- Quantitative details

Step	Specification lines	Proof lines	Proof commands
First-order version	649	2529	880
Slicing	1083	7976	2778
Overflow tests	24087	7100	2568

- PRECiSA execution time

File	time (mm:ss)
First-order version	Time-Out (> 1d)
• •	00:00.00
↕ •	00:00.14
↕ •	00:00.18
• ↔	00:08.26
↕ ↔	05:20.46
↕ ↔	07:10.11

Related Work

- There are several tools available to analyse floating-point representation errors in C programs
- Most of them distinguish from this work in at least one of the following aspects:
 - Do not handle unstable guards
 - Do not instrument the final code to provide a warning when an unstable test is detected
 - Do not provide a proof certificate that can be verified using an external proof assistant as PVS
 - Need hints from the user
 - Need qualified specialist

Related Work

- Several tools have been proposed in the last few years to improve the quality of floating-point software
- Two main groups:
 - Precision allocation:
 - Rosa, Precimonius, and FPTuner
 - Optimization:
 - CoHD, Herbie, AutoRNP, and Salsa

Related Work

- The current work uses PRECiSA with Frama-C
- Frama-C was used to analyse numerical properties of C source
 - Usually using Gappa as back-end
 - Applicable just to straight-line code
 - Verifying more complex program requires additional annotations and expert user hints
- Coq has been used to prove verification conditions together with Gappa
 - Requires user intervention in the specification and verification processes

Related Work

- Fluctuat correctly estimates the rounding error of a program, and it detects possible unstable guards but does not provide any warning in this situation
- Astree is a tool that detects the presence of run-time exceptions such as overflows, not-a-number, and division by zero