From Correct Specifications to Automatically Verified Implementations

Mariano M. Moscato
National Institute of Aerospace

Joint work with: Laura Titolo (NIA), Aaron Dutle (NASA), and César Muñoz (NASA*).

June 2022

* At time of contribution

Context

- NASA LaRC: Large collection of proven-correct algorithms (NASALib)
  - 53 libraries, ~30K theorems
  - Prototype Verification System (PVS)
    - Interactive theorem prover for classical higher-order logic
- Actual implementations in modern languages needed
  - DAIDALUS: RTCA DO-365 reference implementation of a detect-and-avoid concept
  - ICAROUS: customizable software architecture with highly assured core modules
- Gap between specification and implementation
  - No automatic method available PVS -> real-world implementation
  - Regular situation: Implementation inspired in proven-correct specification

Goal

- Question: How to inherit/transfer properties in a systematic/automated way?
  - Adherence to specification
  - Full/modulo implementation idiosyncrasy

- Traversed several projects over the past years
- Leverage existing PVS specifications to improve correctness assurance
- Safety-critical implementations require a quality step over testing
- Systematic || Automatic

Results

- Success measured by
  - reviewed publications of each iteration
  - CPR: reference implementation included in the ADS-B standard
- Technique applied on four different settings
  - ADS-B's CPR (2)
  - Geofencing suite
  - Well-Clear algorithm
- Adapted the technique to incorporate different analysis
  - Floating-point round-off errors
  - Floating-point stability
  - Partial correctness of integer-based algorithms

Content

1) Application #1: Compact Position Reporting Algorithm
2) Application #2: Floating-Point Programs
3) Conclusion

ADS-B CPR

Compact Position Reporting Algorithm

The ADS-B Surveillance Technology

- Automatic Dependent Surveillance - Broadcast
  - Foundation of NextGen (Next generation of air traffic management systems)
  - Moving from ground radar and navigational aids to precise tracking
- Aircraft periodically broadcasts accurate surveillance information
  - to ground stations and near aircraft
  - position and velocity
- Automatic — no pilot intervention needed
- Dependent — on navigation system
- Mandatory in US airspace since January 2020

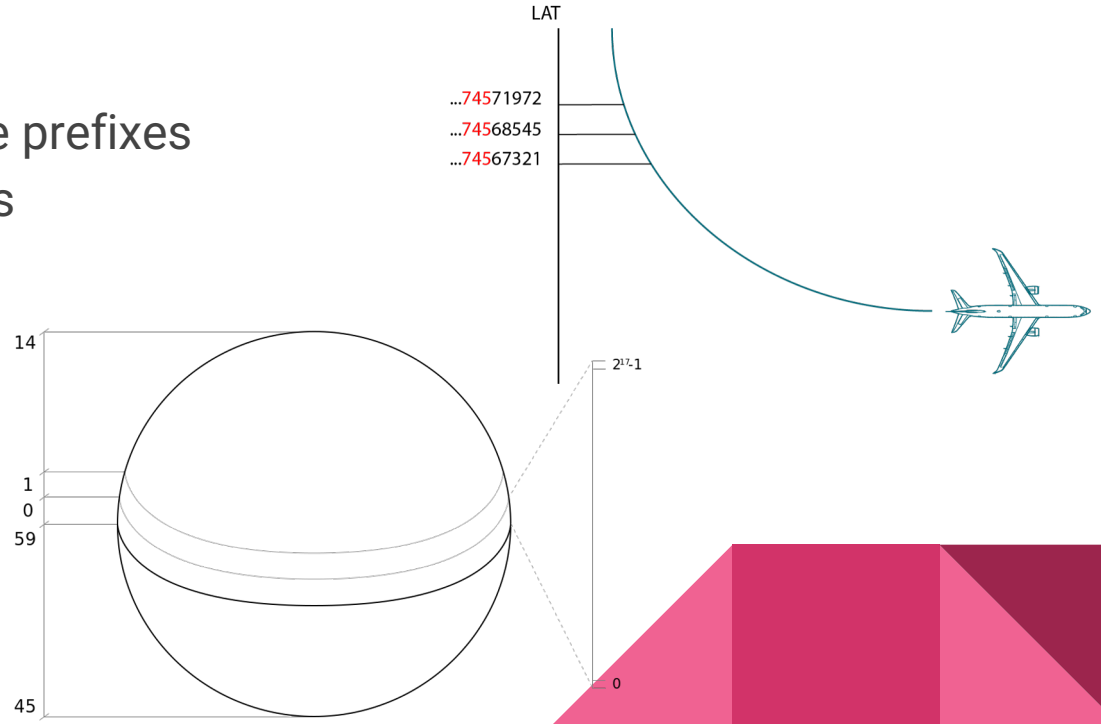
ADS−B: Highs and Lows

- Pros: broadcast vs. radar-based approaches
  - More precise
    - NextGen requirement: position granularity of ~5.1 meters
  - More coverage
- Cons: Make use of existent hardware
  - TCAS transponders
  - 35 bits for position data in the broadcast message
  - Too coarse granularity (~300 meters)
    - if raw positions are transmitted

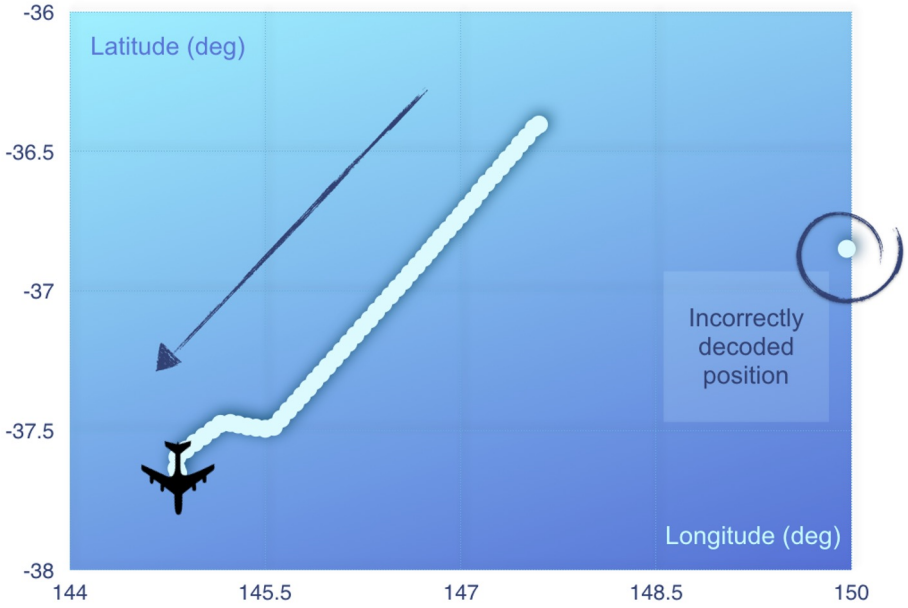
CPR: Compact Position Reporting Algorithm

Idea (for latitude)

- Consecutive positions share prefixes
- Transmit only 17-bit suffixes
- Divide the globe in zones
- Divide the zones in 2^{17} bins
- To recover the zone:
  - use 2 different zone grids



Known Issues



Reported by Airservices Australia (2007)

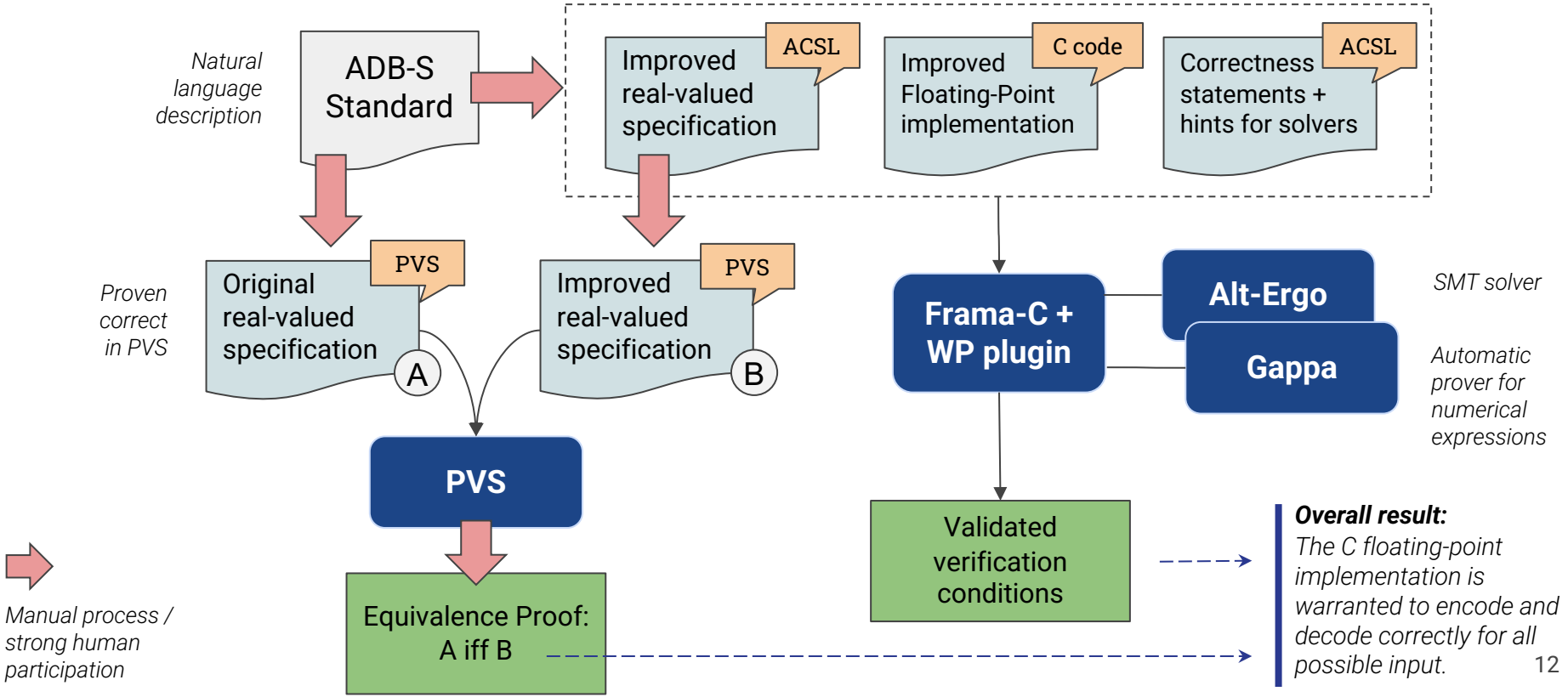
Analysis of the Algorithm

1) Found technical issues in the standard
    - Counterexamples for the real-valued model
2) Presented amended version
    - Formally proven correct in PVS
3) Proposed simpler formulation
    - Reduced numerical complexity
4) Developed formally-verified prototype implementation
    - C, PVS, Frama-C, Gappa, Alt-Ergo

Dutle A., Moscato M., Titolo L., Muñoz C. *A Formal Analysis of the Compact Position Reporting Algorithm*. VSTTE 2017.

Titolo L., Moscato M., Muñoz C., Dutle A., Bobot F. *A Formally Verified Floating-Point Implementation of the Compact Position Reporting Algorithm*. FM 2018.

First Iteration: CPR floating-point implementation



Example: Global Decoding

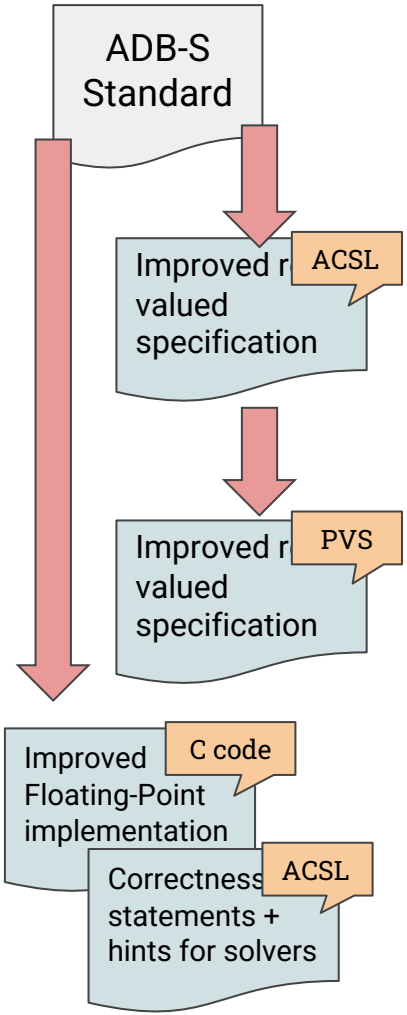
Given an *even* and an *odd* bin number YZ0 and YZ1, the recovered latitude is

$$Rlat_i(YZ_0, YZ_1) := Dlat_i \left(\mathbf{mod} \left(\lfloor ZO_0 - ZO_1 + 1/2 \rfloor, 60 - i \right) + YZ_i \frac{1}{2^{17}} \right)$$

where ZO_i (zone offset) $ZO_i := \frac{Dlat_i}{ZO} \cdot \frac{YZ_i}{2^{17}}$ where $i \in \{0, 1\}$

Note that

1) $Rlat_i$ returns the **center** of the bin where the input latitude lies,
2) It is at most at a half-bin from the input latitude



$$Rlat_0(YZ_0, YZ_1) := Dlat_0\left(\mathsf{mod}\left(\left\lfloor \frac{59YZ_0 - 60YZ_1}{2^{17}} + \frac{1}{2} \right\rfloor, 60\right) + \frac{YZ_0}{2^{17}}\right)$$

```
/*@ axiomatic real_function {
  logic real rLatr (int yz0,int yz1) =
   \let dLatr  = 360.0 / 60.0;
   \let jar    = (59.0*yz0 - 60.0*yz1 + 0x1.0p+16)*0x1.0p-17;
   \let jr     = \floor(jar);
   \let j60ir  = jr/60.0;
   dLatr*((jr-60.0*\floor(j60ir))+yz0*0x1.0p-17); } @*/
```

```
rLatr_i_0 (yz0,yz1:int): real =
 LET dLatr = 360 / 60 IN
 LET jar   = (59*yz0 - 60*yz1 + 2^16) * 2^-17 IN
 LET jr    = floor(jar) IN
 LET j60ir = jr/60 IN
 dLatr * ((jr - 60*floor(j60ir)) + yz0 * 2^-17)
```

```
/*@ requires 0 <= yz0 <= 131071; requires 0 <= yz1 <= 131071;
    requires \floor(yz0) == yz0; requires \floor(yz1) == yz1;
    ensures \abs(\result - rLatr(yz0,yz1)) <= 0.000022888; */
fp rLatf (int yz0, int yz1) {
```

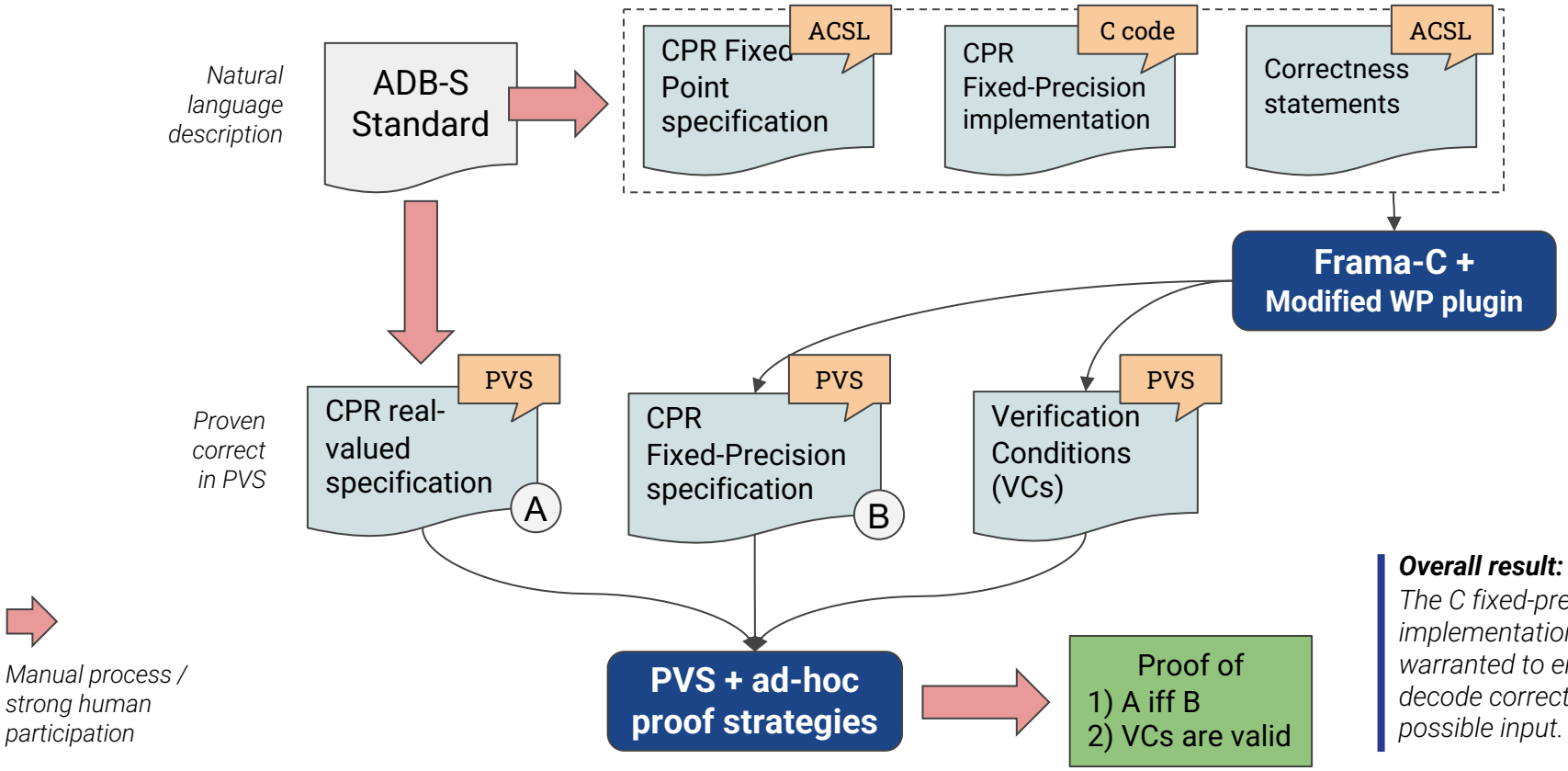
"the floating-point result is at most half bin apart from the real-valued result"

CPR: Avoiding Floating-Points

- Latitude & Longitude inputs are usually expressed in a 32-bit format
  - AWB: Angular Weighted Binary
- New CPR specification using fixed precision (32 bits) arithmetic
  - Proven equivalent to the original
- New formally-verified C implementation
  - Stronger correctness condition
- Modified a Frama-C plugin to express verification conditions directly in PVS
  - Goal: Improvement in the automation of the verification

Dutle A., Moscato M., Titolo L., Muñoz C., Anderson G., Bobot F.
Formal Analysis of the Compact Position Reporting Algorithm.
Formal Aspects of Computing 2020.

Second Iteration: CPR fixed-point implementation



Overall result:
The C fixed-precision implementation is warranted to encode and decode correctly for all possible input.

Impact of the Formal Analysis of CPR

External

- Corrections added to the standard
- Prototype included as reference implementation
  - https://github.com/nasa/cpr

Internal

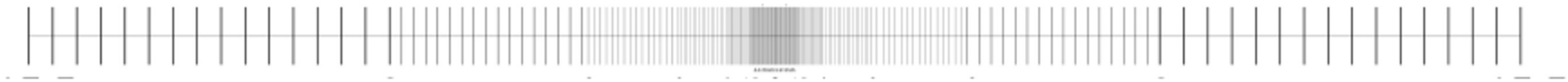
- NIA Best Paper Award 2018
- NASA LaRC Group Achievement Award 2022

Floating-Point Implementations

A real nightmare

Floating-point Issues

- Representation errors (values)
  - “Floating-point numbers look like the real ones” but they’re not



- Round-off errors (expressions)
  - $a_{\mathbb{F}} +_{\mathbb{F}} b_{\mathbb{F}} = R2F(F2R(a_{\mathbb{F}}) +_{\mathbb{R}} F2R(b_{\mathbb{F}}))$
  - $a_{\mathbb{F}} +_{\mathbb{F}} b_{\mathbb{F}}$ versus $a_{\mathbb{R}} +_{\mathbb{R}} b_{\mathbb{R}}$
  - $| f_{\mathbb{F}}(x) - f_{\mathbb{R}}(x) | \leq \varepsilon_f$

PRECiSA

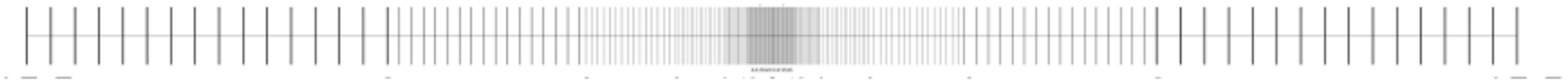
- Automatic analyzer for floating-point programs
  - Static Analysis
- Input: Floating-point program
  - $| f_{\mathbb{F}}(x) - f_{\mathbb{R}}(x) | \leq \boldsymbol{\varepsilon_f}$
- Output: Estimation for the round-off error
  - $| f_{\mathbb{F}}(x) - f_{\mathbb{R}}(x) | \leq \boldsymbol{\varepsilon_f}$
- Distinguish characteristic: formal certificates
  - PVS theorems
  - Dependent on NASALib's floating-point formalization

Moscato M., Titolo L., Dutle A., Muñoz C. Automatic Estimation of Verified Floating-Point Round-Off Errors via Static Analysis. SAFECOMP 2017.

Titolo L., Feliú M., Moscato M., Muñoz C. *An Abstract Interpretation Framework for the Round-Off Error Analysis of Floating-Point Programs*. VMCAI 2018.

Floating-point Issues -- Even More…

- Representation errors (values)
  - “Floating-point numbers look like the real ones” but they’re not



- Round-off errors (expressions)
  - $a_{\mathbb{F}} +_{\mathbb{F}} b_{\mathbb{F}} = R2F(F2R(a_{\mathbb{F}}) +_{\mathbb{R}} F2R(b_{\mathbb{F}}))$
  - $a_{\mathbb{F}} +_{\mathbb{F}} b_{\mathbb{F}}$ versus $a_{\mathbb{R}} +_{\mathbb{R}} b_{\mathbb{R}}$
  - $| f_{\mathbb{F}}(x) - f_{\mathbb{R}}(x) | \leq \varepsilon_f$
- Control-flow errors (programs)
  - Anomalies provoked by evaluation of FP expressions

Example: implicit coordination

eps_line((v_x, v_y), (s_x, s_y)) =

```
if (sx*vx + sy*vy) * (sx*vx - sy*vy) > 0 then
        1 // right turn
else
        -1 // left turn
```

Example: implicit coordination

eps_line'((v_x, v_y), (s_x, s_y), ε_f) =

let check = (sx*vx + sy*vy) * (sx*vx - sy*vy) in

if check > ε_f then

1 // right turn

elsif check ≤ -ε_f then

-1 // left turn

else ω // inconclusive

Transformation

- Based on two abstractions: $\beta^+, \beta^- : \widetilde{\mathbb{B}} \rightarrow \widetilde{\mathbb{B}}$

$$\beta^+(\widetilde{expr} \leq 0) = \widetilde{expr} \leq -\epsilon \qquad \beta^-(\widetilde{expr} \leq 0) = \widetilde{expr} > \epsilon$$
$$\beta^+(\widetilde{expr} \geq 0) = \widetilde{expr} \geq \epsilon \qquad \beta^-(\widetilde{expr} \geq 0) = \widetilde{expr} < -\epsilon$$
$$\beta^+(\widetilde{expr} < 0) = \widetilde{expr} < -\epsilon \qquad \beta^-(\widetilde{expr} < 0) = \widetilde{expr} \geq \epsilon$$
$$\beta^+(\widetilde{expr} > 0) = \widetilde{expr} > \epsilon \qquad \beta^-(\widetilde{expr} > 0) = \widetilde{expr} \leq -\epsilon$$
$$\beta^+(\tilde{\phi}_1 \wedge \tilde{\phi}_2) = \beta^+(\tilde{\phi}_1) \wedge \beta^+(\tilde{\phi}_2) \qquad \beta^-(\tilde{\phi}_1 \wedge \tilde{\phi}_2) = \beta^-(\tilde{\phi}_1) \vee \beta^-(\tilde{\phi}_2)$$
$$\beta^+(\tilde{\phi}_1 \vee \tilde{\phi}_2) = \beta^+(\tilde{\phi}_1) \vee \beta^+(\tilde{\phi}_2) \qquad \beta^-(\tilde{\phi}_1 \vee \tilde{\phi}_2) = \beta^-(\tilde{\phi}_1) \wedge \beta^-(\tilde{\phi}_2)$$
$$\beta^+(\neg\tilde{\phi}) = \beta^-(\tilde{\phi}) \qquad \beta^-(\neg\tilde{\phi}) = \beta^+(\tilde{\phi})$$

$$\widetilde{eval}_{\widetilde{\mathbb{B}}}(\tilde{\sigma}, \beta^+(\tilde{\phi})) \Rightarrow \widetilde{eval}_{\widetilde{\mathbb{B}}}(\tilde{\sigma}, \tilde{\phi}) \wedge eval_{\mathbb{B}}(\sigma, R_{\widetilde{\mathbb{B}}}(\tilde{\phi})).$$
$$\widetilde{eval}_{\widetilde{\mathbb{B}}}(\tilde{\sigma}, \beta^-(\tilde{\phi})) \Rightarrow \widetilde{eval}_{\widetilde{\mathbb{B}}}(\tilde{\sigma}, \neg\tilde{\phi}) \wedge eval_{\mathbb{B}}(\sigma, \neg\, R_{\widetilde{\mathbb{B}}}(\tilde{\phi})).$$

Transformation

- Conditionals

$$\tau(\mathit{if}\ \tilde{\phi}\ \mathit{then}\ S_1\ \mathit{else}\ S_2) = \mathit{if}\ \beta^+(\tilde{\phi})\ \mathit{then}\ \tau(S_1)\ \mathit{elseif}\ \beta^-(\tilde{\phi})\ \mathit{then}\ \tau(S_2)\ \mathit{else}\ \omega;$$

- Support for simple iterations

$$\tau(for(i_0, i_n, acc_0, \lambda(i, acc).S)) = for(i_0, i_n, acc_0, \lambda(i, acc).\tau(S)).$$

- Non-recursive function applications

$$\begin{aligned}\tau(g(A_1, \ldots, A_n)) = \mathit{if}\ & A_1 = \omega\ \mathit{then}\ \omega \\ & \vdots \\ & \mathit{elsif}\ A_n = \omega\ \mathit{then}\ \omega \\ & \mathit{else}\ g^\tau(A_1, \ldots, A_n, e'_1, \ldots, e'_m)\end{aligned}$$

Transformation

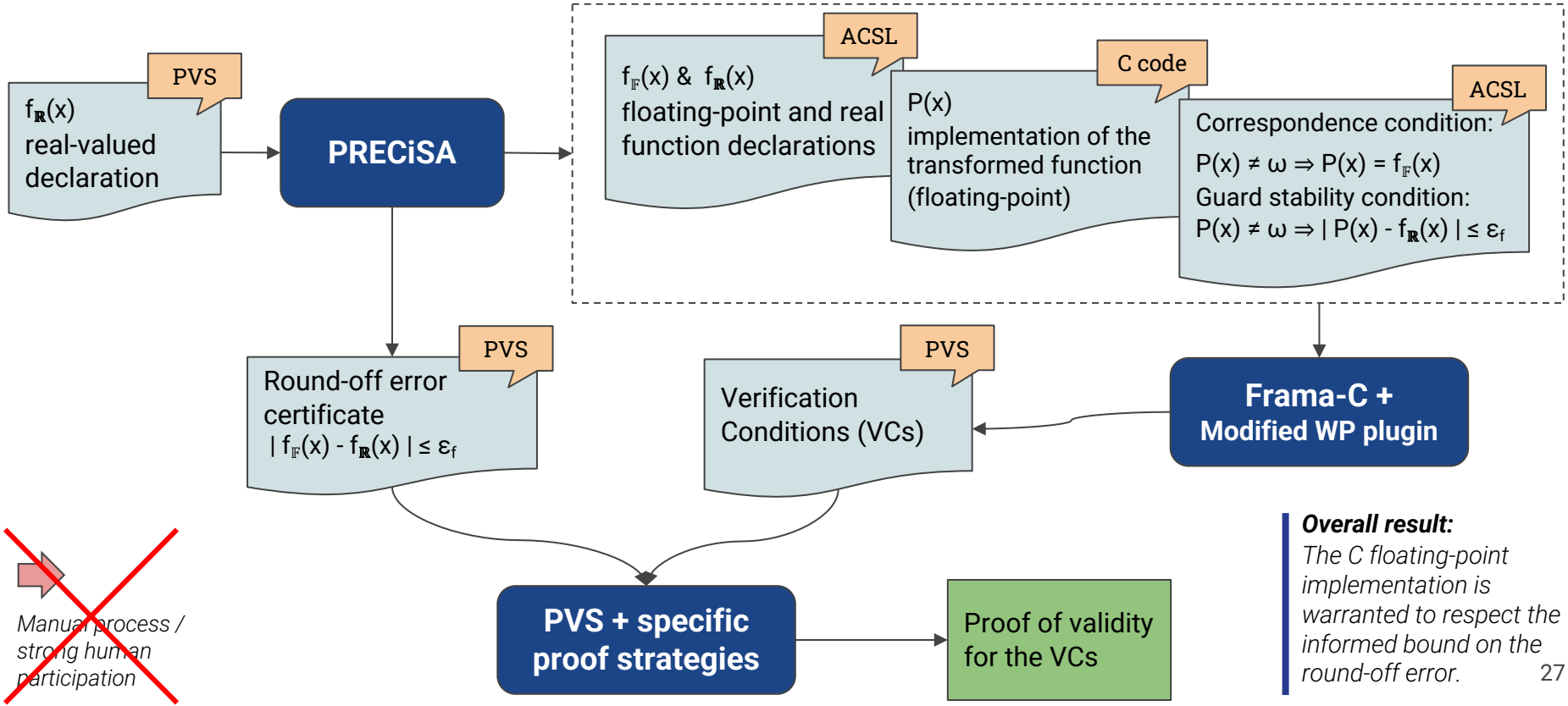
- Main theorem:

Theorem 2 (Program Transformation Correctness). *Given $P \in \mathbb{P}$, for all $f(x_1, \ldots, x_n) = S \in P$, let $\tilde{f}^{\tau}(\tilde{x}_1, \ldots, \tilde{x}_n, e_1, \ldots, e_m) \in \bar{\tau}(P)$ be its transformed floating-point version. Let $\sigma : \{x_1 \ldots x_n\} \to \mathbb{R}$, and $\tilde{\sigma} : \{\tilde{x}_1 \ldots \tilde{x}_n\} \to \mathbb{F}$, such that for all $i \in \{1, \ldots, n\}$, $R(\tilde{\sigma}(\tilde{x}_i)) = \sigma(x_i)$, it holds that*

$$\tilde{f}^{\tau}(\tilde{x}_1, \ldots, \tilde{x}_n, e_1, \ldots, e_m) \neq \omega \iff |f(x_1, \ldots, x_n) - \tilde{f}^{\tau}(\tilde{x}_1, \ldots, \tilde{x}_n, e_1, \ldots, e_m)| \leq e_{\tilde{f}}$$

- Practical corollary: “If the result of the transformed program is numeric (not warning) then there would have been no flow-instability in the original FP program with the same input”

Third Iteration: Floating-Point Guard Stability



Overall result:
The C floating-point implementation is warranted to respect the informed bound on the round-off error.

$$tcoa(s_z, v_z) = if\ s_z v_z < 0\ then\ -(s_z/v_z)\ else\ 0$$

Time to co-altitude

$$/*@\ real\ tcoa(real\ s_z, real\ v_z) = s_z * v_z < 0\ ?\ -(s_z/v_z) : 0$$

$$double\ fp\_tcoa(double\ \tilde{s}_z, double\ \tilde{v}_z) = \tilde{s}_z \tilde{*} \tilde{v}_z < 0\ ?\ \tilde{-}(\tilde{s}_z \tilde{/} \tilde{v}_z) : 0$$

$$predicate\ tcoa\_stable\_paths(real\ s_z, real v_z, double\ \tilde{s}_z, double\ \tilde{v}_z) =$$

$$(v_z \neq 0 \wedge s_z * v_z < 0 \wedge \tilde{v}_z \neq 0 \wedge \tilde{s}_z \tilde{*} \tilde{v}_z < 0) \vee (s_z * v_z \geq 0 \wedge \tilde{s}_z \tilde{*} \tilde{v}_z \geq 0)$$

$$\textbf{requires} : 0 \leq e$$

$$\textbf{ensures} : result \neq \omega \Rightarrow (result = fp\_tcoa(\tilde{s}_z, \tilde{v}_z)$$

$$\wedge \forall s_z, v_z(|(\tilde{s}_z \tilde{*} \tilde{v}_z) - (s_z * v_z)| \leq e \Rightarrow tcoa\_stable\_paths(s_z, v_z, \tilde{s}_z, \tilde{v}_z))$$

$$*/$$

```
double tau_tcoa (double s̃_z, double ṽ_z, double e){
    if (s̃_z ~* ṽ_z < −e){
    return ~−(s̃_z ~/ ṽ_z);
    } else { if (s̃_z ~* ṽ_z ≥ −e)
            {return 0;
            } else {return ω;}}}
```

transformed program

Time to co-altitude

Numeric (concrete) code

$$
\begin{aligned}
/*@\mathbf{ensures}: & \forall s_z, v_z (1 \le s_z \le 1000 \wedge 1 \le v_z \le 1000 \wedge \\
& |\tilde{s}_z - s_z| \le ulp(s_z)/2 \wedge |\tilde{v}_z - v_z| \le ulp(v_z)/2) \wedge \\
& result \neq \omega \\
& \Rightarrow |result - tcoa(s_z, v_z)| \le 2.78e-12 \\
& */
\end{aligned}
$$

$$
\begin{aligned}
& double\ tau\_tcoa\_num(double\ \tilde{s}_z, double\ \tilde{v}_z)\{ \\
& \quad return\ \ tau\_tcoa\ (\tilde{s}_z, \tilde{v}_z, 1.72e-10)\}
\end{aligned}
$$

call to symbolic function

round-off error e computed by Kodiak

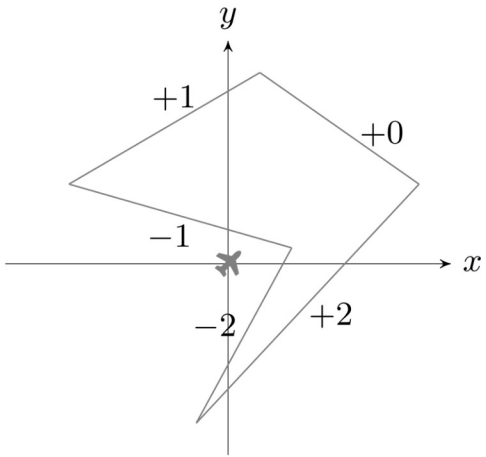
Pros and Cons of this iteration

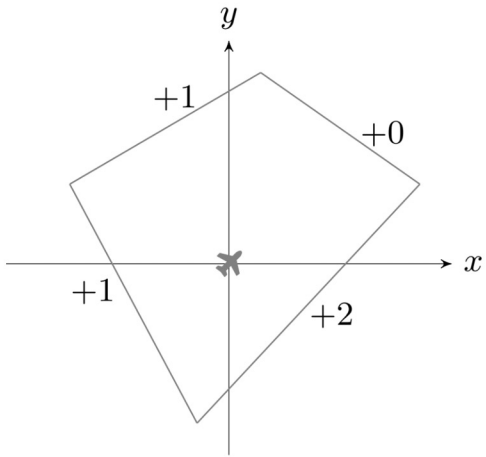
- Automatic generation and analysis of floating-point C implementations
- Proof strategies automatically discharge obligations
  - Proofs are strongly dependent on the program structure
- No required expertise on floating-point nor Formal Methods tools

- New-arguments number growth
  - Caller functions inherit new error arguments from callees
  - Human inspection of generated ACSL can be cumbersome

Applications

- Automatic implementations of geofencing and detect-and-avoid algorithms





Moscato M., Titolo L., Feliú M., Muñoz C. Provably Correct Floating-Point Implementation of a Point-in-Polygon Algorithm. FM 2019.

Titolo L., Moscato M., Feliú M., Muñoz C. Automatic Generation of Guard-Stable Floating-Point Code. iFM 2020.

Conclusion

Previous efforts

- MINERVA approach
  - Narkawicz, Muñoz, Dutle (2017) *The MINERVA Software Development Process*
  - Testing-based validation of human-generated implementations
  - Use PVS specifications as testing Oracle

- Spirit of previous attempt
  - Lensink, Muñoz, Goodloe (2009). *From Verified Models to Verifiable Code*
  - Ad-hoc automatic code generator for PVS
  - Intermediate language
  - Idea of analysis by off-the-shelf tools

What’s Next

- Support for more complex loops
  - Structured idioms
- Scalability
  - Factorization of error bounds
  - Automatic slicing of code
- Integration with modern IDEs
  - VSCode-PVS
- Application to bigger examples
  - NASA’s DAIDALUS

Summary

- Cross-project development of implementation analysis technique
- Different formal methods tools
  - Frama-C, Alt-Ergo, Gappa, PVS, Kodiak
- Incremental reduction of human intervention
- High impact
  - CPR: verified reference implementation included as part of the standard

From Correct Specifications to Automatically Verified Implementations

Mariano M. Moscato

Thank you for your attention