

Taming Floating-point Errors in Avionics Applications at NASA

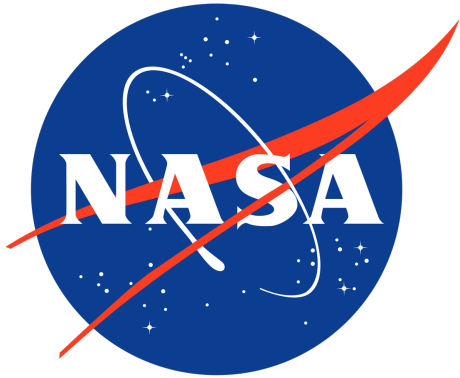
Laura Titolo

Sr. Research Scientist

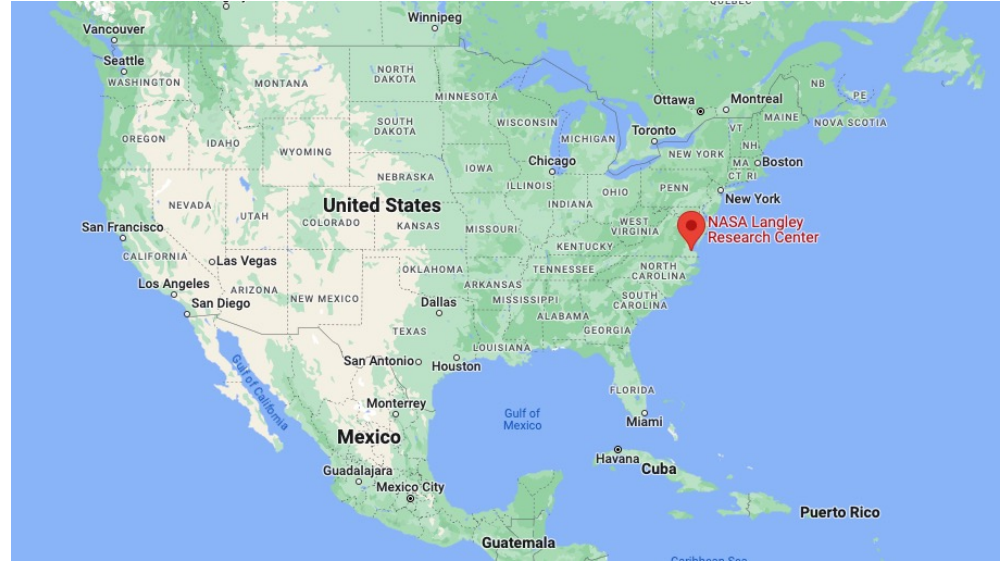
NASA Langley Research Center / AMA Inc.

September 2024

NASA Langley Research Center



National
Aeronautics and
Space
Administration



Formal Methods at NASA

- Formal specifications
- Theorem proving
- Static Analysis
- Model Checking
- Runtime monitoring
- Requirements elicitation
- Applications:
 - UAS
 - Avionics
 - Mars Rovers (PLEXIL5)

PVS

DAIDALUS



K^{paw}diak

ReFlow



PRECISA

Formal Methods at NASA

- Formal specifications
- Theorem proving
- Static Analysis
- Model Checking
- Runtime monitoring
- Requirements elicitation
- Applications:
 - UAS
 - Avionics
 - Mars Rovers (PLEXIL5)

PVS



Kadiak

ReFlow

PRECISA

DAIDALUS

RTCA/FAA Minimum
Operational Performance
Standards (MOPS) DO365
for detect and avoid of
unmanned aircraft systems
(UAS)

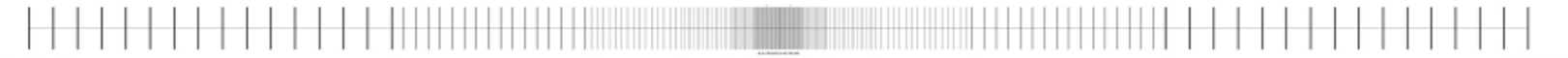


Why Formal Methods? Why PVS?

- Safety- and mission-critical systems
- Formal guarantees on the behavior of the system
- NASA PVS Library has been developed for over 3 decades and contains 46 libraries and 35K theorems
- PVS is the only theorem prover featured in a mainstream movie 😊

Floating-Point Round-Off Errors

- Floating-point (FP) = **finite representation** of real numbers
- Not every real number can be represented as a floating-point



- **Round-off error** = $|r - \text{round}(r)|$
- Propagation of rounding error $\implies$ large difference between the computed result and the expected one
- 0.1 is not a FP (rounded to 3602879701896397 / 36028797018963968)

Floating-Point arithmetic is counter-intuitive

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

Floating-Point arithmetic is counter-intuitive

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

Should be 0 if evaluated in exact
real number arithmetic

Floating-Point arithmetic is counter-intuitive

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

Should be 0 if evaluated in exact
real number arithmetic

Floating-Point arithmetic is counter-intuitive

```
Prelude> (4/3 - 1) * 3 - 1  
-2.220446049250313e-16  
Prelude> floor((4/3 - 1) * 3 - 1)  
-1  
Prelude> if (floor((4/3 - 1) * 3 - 1)) < 0 then 100 else 1  
100  
Prelude> █
```

Should be 0 if evaluated in exact real number arithmetic

The else branch should be taken

Floating-Point arithmetic is counter-intuitive

Round-off error = 99 🤔

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

```
-2.220446049250313e-16
```

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

```
-1
```

```
Prelude> if (floor((4/3 - 1) * 3 - 1)) < 0 then 100 else 1
```

```
100
```

```
Prelude> 
```

Floating-point errors may be dangerous in safety-critical applications



Picture: Bernd vdB, Public domain, via Wikimedia Commons

Patriot missile failure
(1991)

Floating-point errors may be dangerous in safety-critical applications

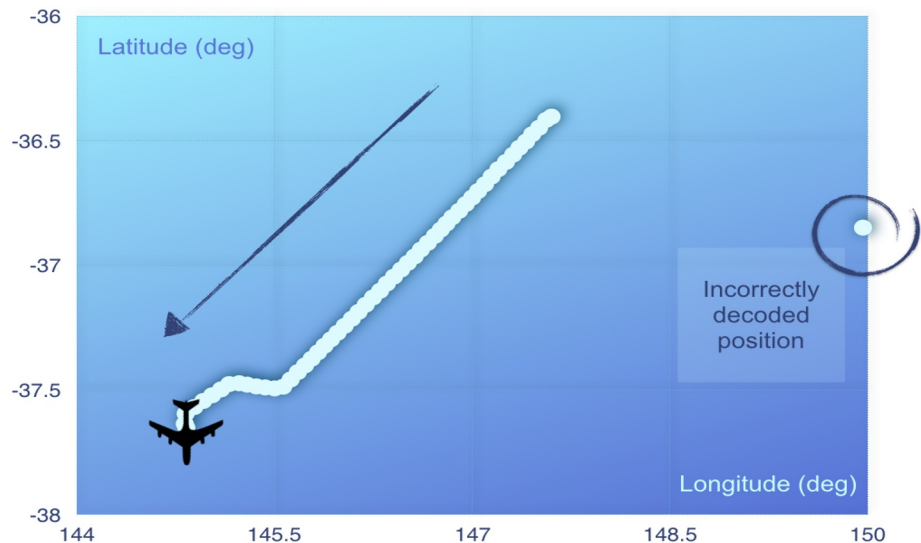
THE 7 BILLION DOLLAR OVERFLOW



Ariane V
(1996)

Photographs credits: 1996 © ESA/CNES; 1996 © ESA; 1996 © ESA.

Floating-point errors may be dangerous in safety-critical applications



Situation as reported by Airservices Australia

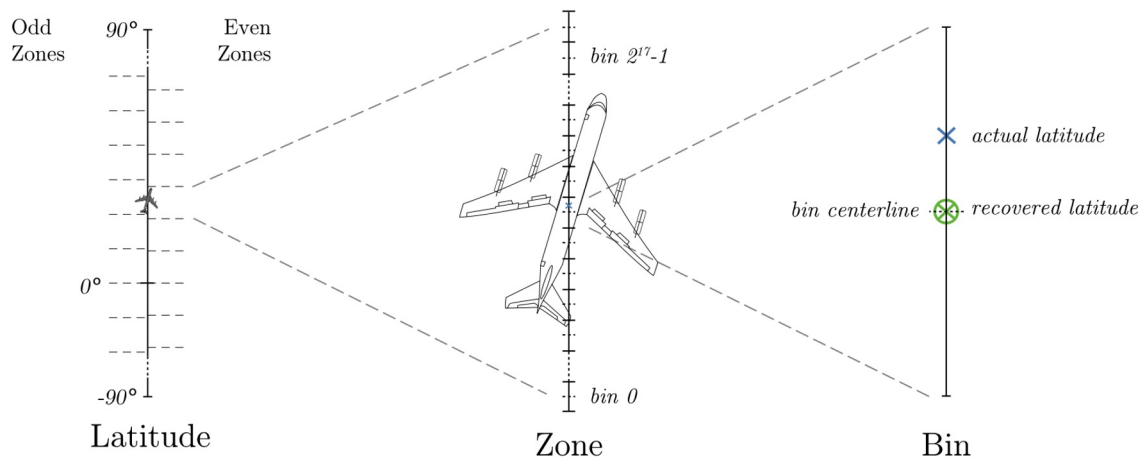
ADS-B Compact
Position Reporting
Algorithm
(2018)

ADS-B: Pro and Cons

- Pros: Broadcast vs. radar-based approaches
 - More precise
 - More Coverage
- Cons: Makes use of existing hardware
 - TCAS transponders
 - 35 bits for position in the broadcast message
 - Too coarse granularity (~300 mt) for raw positions

CPR: Compact Position Reporting Algorithm

- Divide the globe into 59 (odd) or 60 (even) equally sized zones
- Divide each zone in 2^{17} bins
- Zone number + Bin number = Position $\pm \approx 5.1\text{mt}$



CPR Encoding

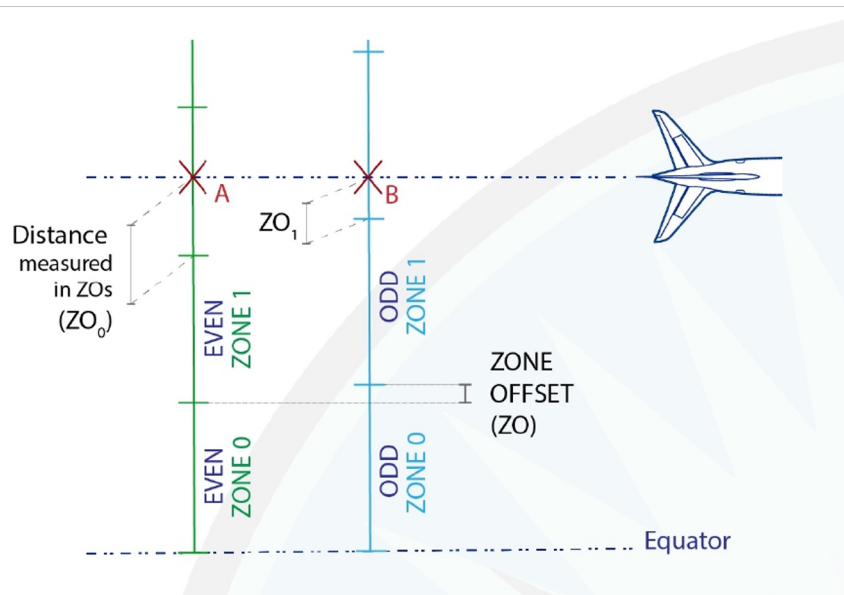
$$lat_enc(i, lat) = \text{mod} \left(\left\lfloor 2^{17} \frac{\text{mod}(lat, dlat_i)}{dlat_i} + \frac{1}{2} \right\rfloor, 2^{17} \right)$$

- The encoding computes the bin number
- Only the bin numbers and encoding type are transmitted
- Not the zone!

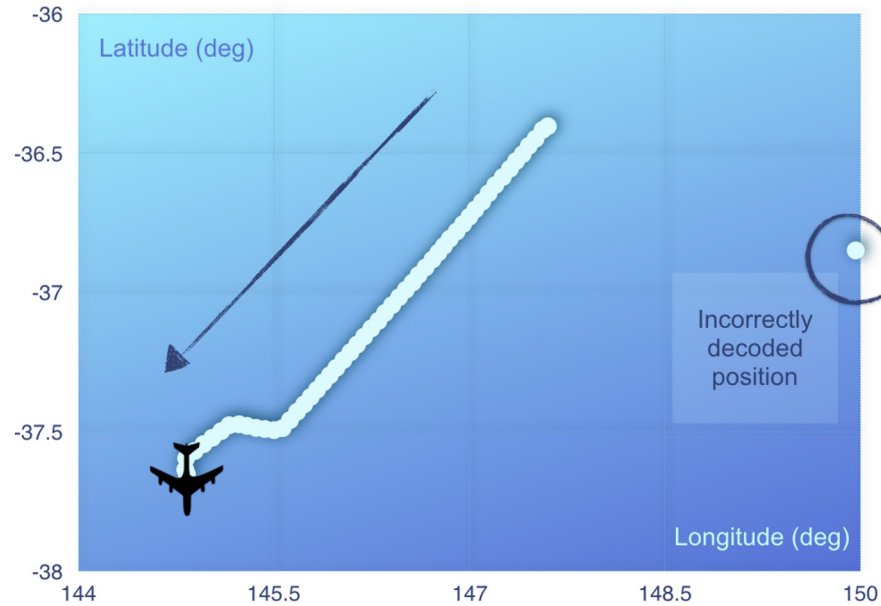


CPR Decoding

- Both **zone** and **bin number** are necessary to recover the position
- CPR needs an odd and an even message sent **sufficiently close** to each other to compute the decoded zone



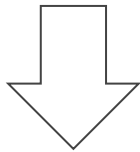
CPR Issues



Situation as reported by Airservices Australia

What was the issue?

Requirements were not enough to guarantee the intended precision even assuming exact real-number arithmetic.



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

We proposed a slightly **tightened set of requirements** and formally proved the algorithm correct in PVS **assuming exact arithmetic**

$| \text{original position} - \text{recovered position} | \leq 5.1\text{mt}$ (position accuracy)

Implementing CPR in finite precision arithmetic

- Heavy use of modulus and floor $\Rightarrow$ huge accumulated round-off error
- For instance, using single-precision floating-point, the recovered position of lat = -77.368 and lon = 180 was off by approx **1500 nautical miles!**
- Large rounding errors also with double precision floating-point and fixed-point arithmetic

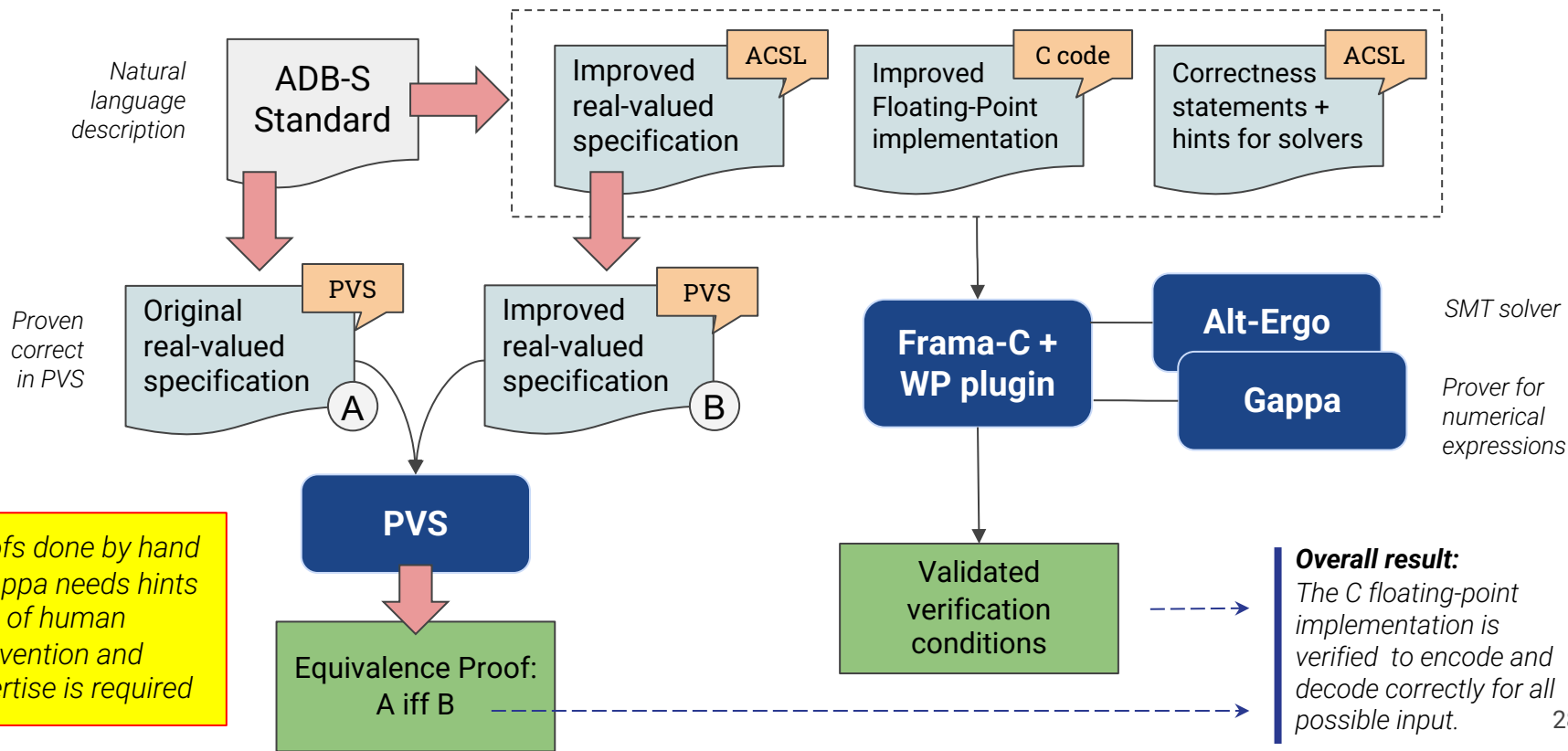
Our solution

- Propose simpler formulation reducing numerical complexity (**CPR***)
- Use a suite of existing **formal methods** tools to provide a verified and correct implementation of CPR with finite precision arithmetic:
 - **PVS**: interactive theorem prover
 - **Gappa**: framework for the analysis of floating-point programs
 - **Frama-C**: static analysis suite for C



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

CPR floating-point implementation verification



CPR*

- CPR* = **Formally verified** C implementation of CPR
 - double-precision floating-point
 - 32 bits unsigned integers
- **Reference implementation** in the revised version of the ADS-B **standard** document (RTCA DO-260B/Eurocae ED-102A)
- Available at <https://github.com/nasa/cpr>
- NASA group achievement award 2020

RTCA



CPR* Verification approach

- Successful combination of different formal methods tools
- The CPR verification was not completely automatic!
 - Manual annotation of the C code
 - Gappa needs hints to verify a tight rounding error
 - A lot of interactive theorem proving in PVS
- What do we want?
 - **Automatically** obtain sound and tight **rounding error estimations**
 - **Automatically generate and verify** a C finite-precision implementation from a real-number specification

PRECiSA

<https://github.com/nasa/PRECiSA>

Static analysis tool for
automatically estimating the round-
off error of floating-point programs

PRECiSA

- PRECiSA is a framework for the analysis of floating-point programs
- Computes round-off error bounds + formal proof PVS

The diagram illustrates the relationship between a floating-point implementation and a real number specification. A central light blue box contains the equation $|\tilde{f} - f| \leq e$. An orange dashed arrow points from the text "floating-point implementation" to the box. Another orange dashed arrow points from the text "real number specification" to the box. A third orange dashed arrow points from the text "error bound" to the box.

floating-point implementation

$|\tilde{f} - f| \leq e$

real number specification

error bound



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

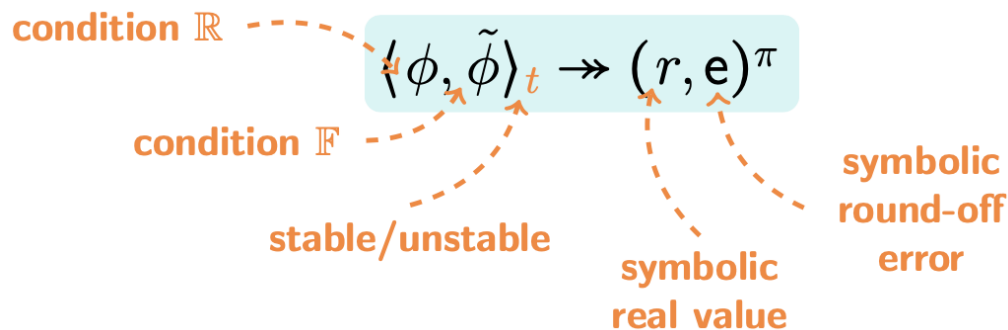


Titolo L., Feliú M., Moscato M., Muñoz C. *An Abstract Interpretation Framework for the Round-Off Error Analysis of Floating-Point Programs.* VMCAI 2018.

PRECiSA

PRECiSA Step I - Static Analysis

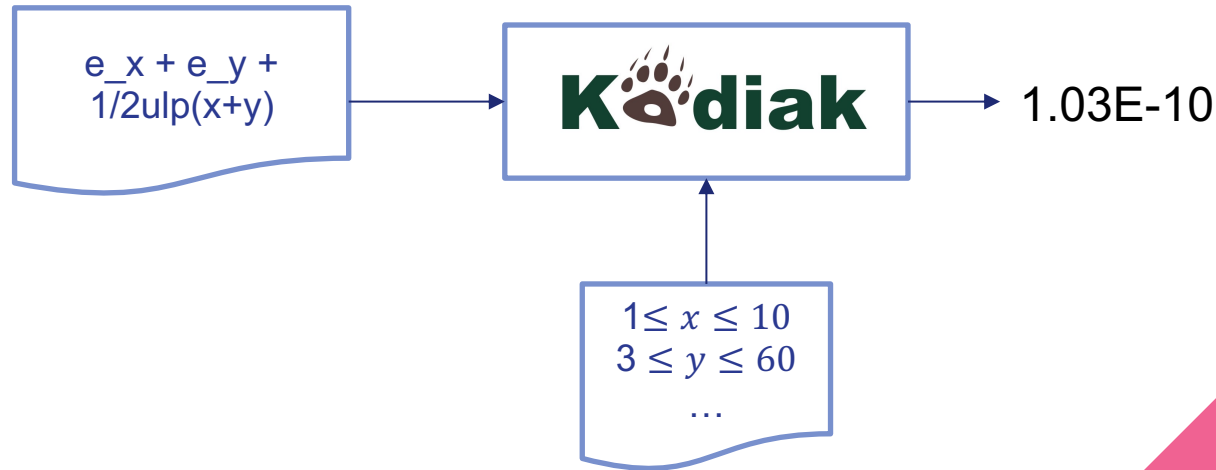
- Input: program = set of function declarations f
- Output: for each f PRECiSA computes a conditional error bound



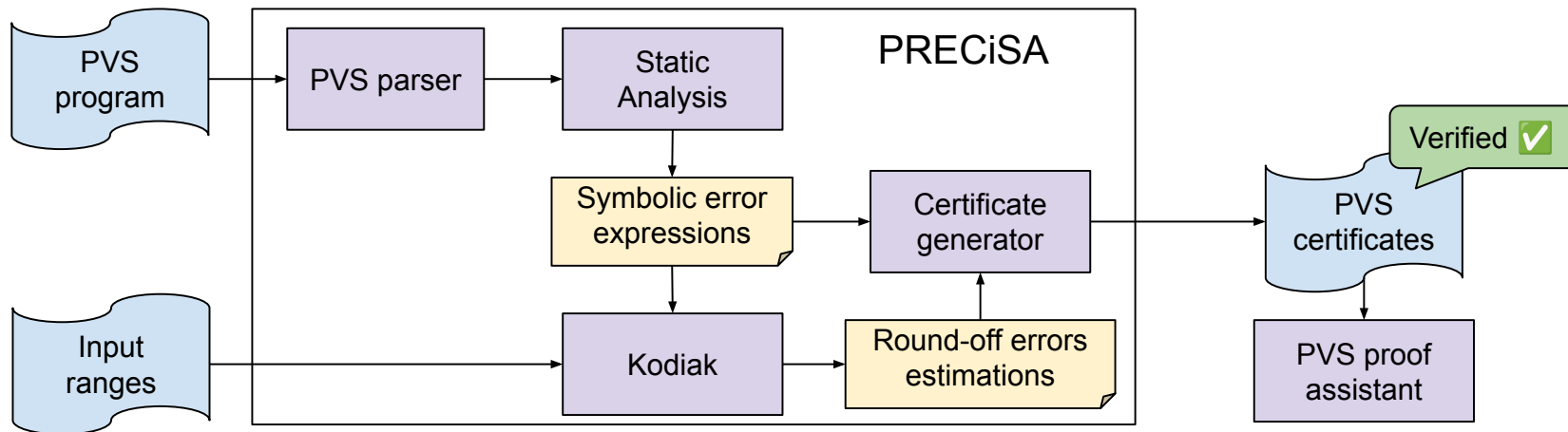
- Everything is symbolic!  Compositional analysis

PRECiSA Step II: Global Optimization

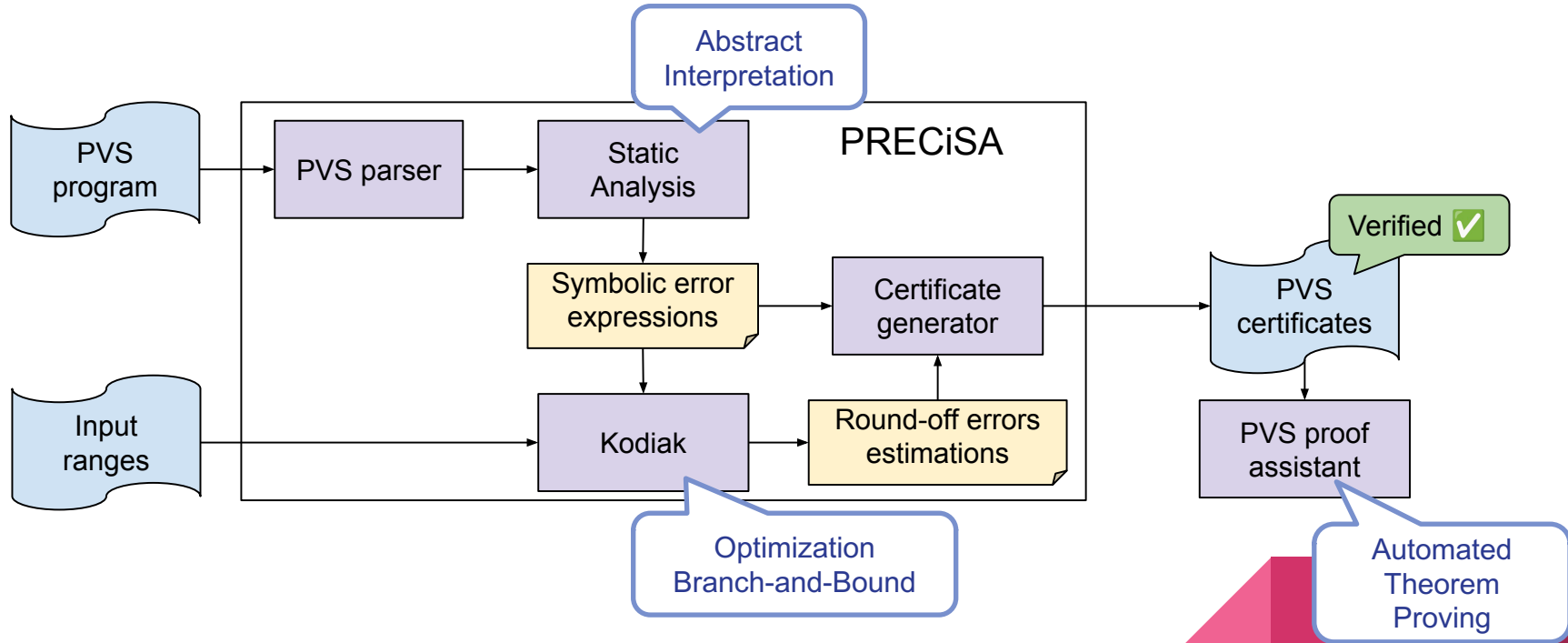
- Given input ranges for the variables
- the global optimizer Kodiak is used to maximize the error expression



PRECiSA workflow



Different Formal Methods Techniques in PRECiSA



Sound analysis of conditionals

if $\tilde{\phi}$ then A else B

Stable :

$$\tilde{\phi} \wedge \mathbf{R}(\tilde{\phi})$$

$$\neg \tilde{\phi} \wedge \neg \mathbf{R}(\tilde{\phi})$$

$$\mathcal{E}[[A]]^{\pi \cdot 1} \Downarrow_{(\mathbf{R}(\tilde{\phi}), \tilde{\phi})}$$

$$\mathcal{E}[[B]]^{\pi \cdot 0} \Downarrow_{(\mathbf{R}(\neg \tilde{\phi}), \neg \tilde{\phi})}$$

Unstable :

$$\tilde{\phi} \wedge \neg \mathbf{R}(\tilde{\phi})$$

$$\neg \tilde{\phi} \wedge \mathbf{R}(\tilde{\phi})$$

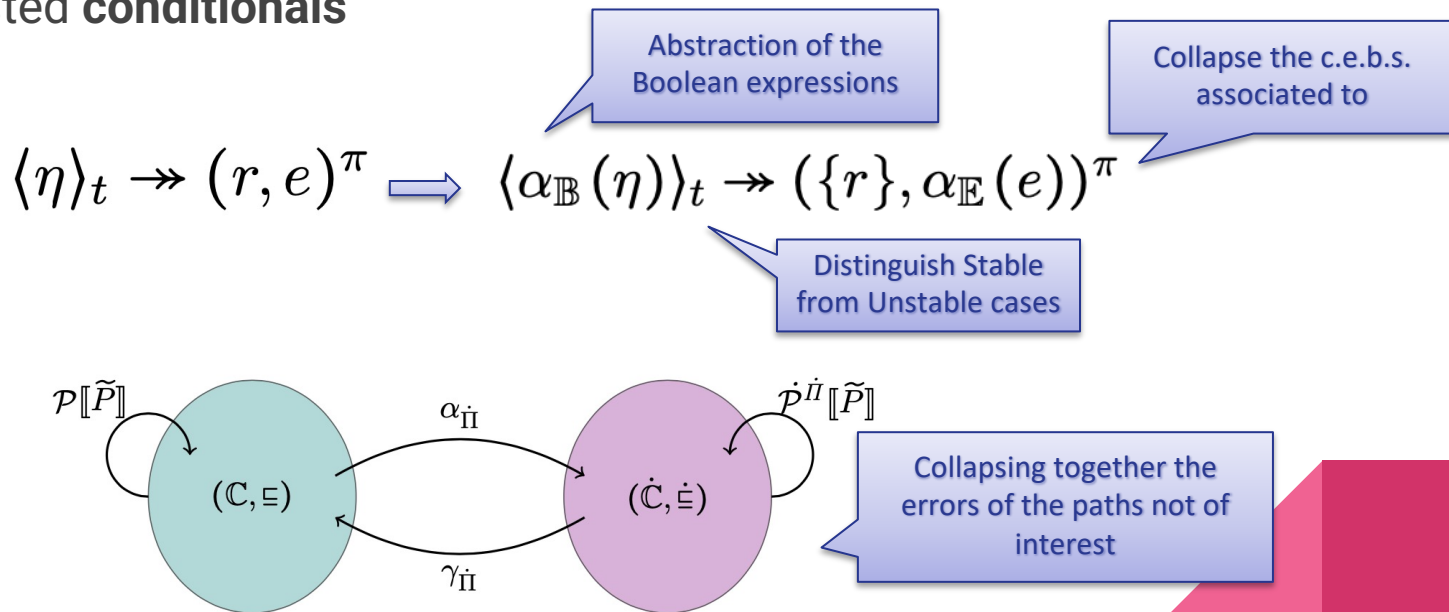
mark as unstable

$$\langle \eta \wedge \neg \mathbf{R}(\tilde{\phi}), \tilde{\eta} \wedge \tilde{\phi} \rangle_{\text{u}} \rightarrow (r_B, e_A + |r_A - r_B|)^{\pi \cdot 1}$$

$$\langle \eta \wedge \mathbf{R}(\tilde{\phi}), \tilde{\eta} \wedge \neg \tilde{\phi} \rangle_{\text{u}} \rightarrow (r_A, e_B + |r_A - r_B|)^{\pi \cdot 0}$$

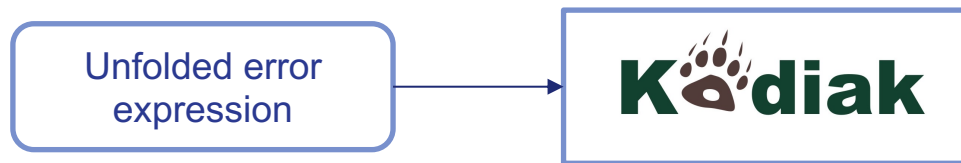
PRECiSA uses abstract interpretation to improve scalability

Problem: Explosion of the number of elements in our semantics in the case of nested **conditionals**

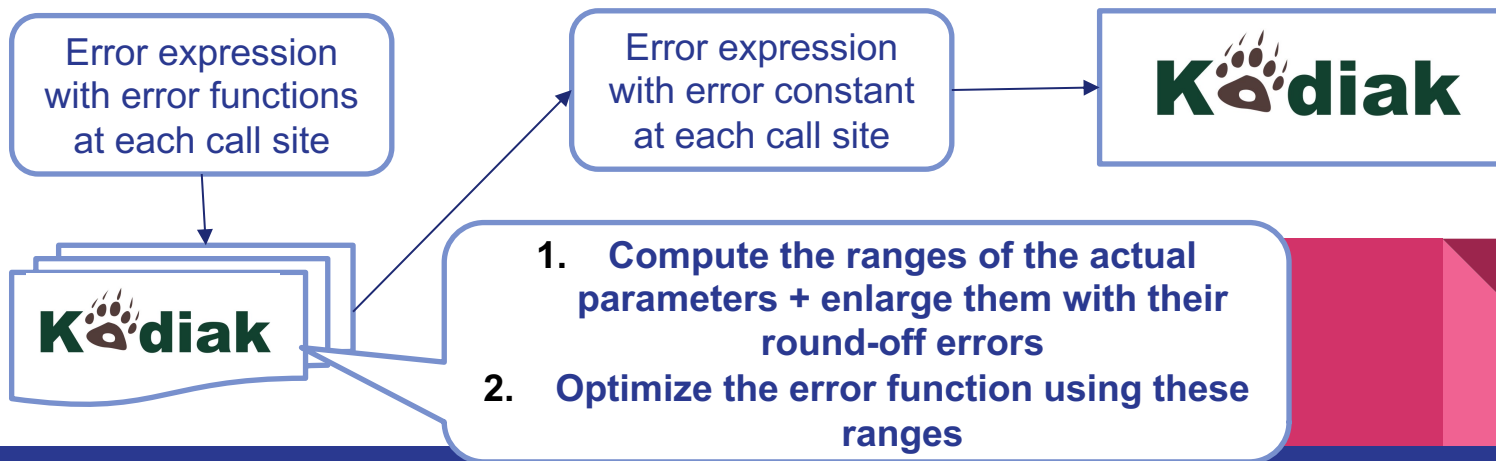


Function calls abstract analysis

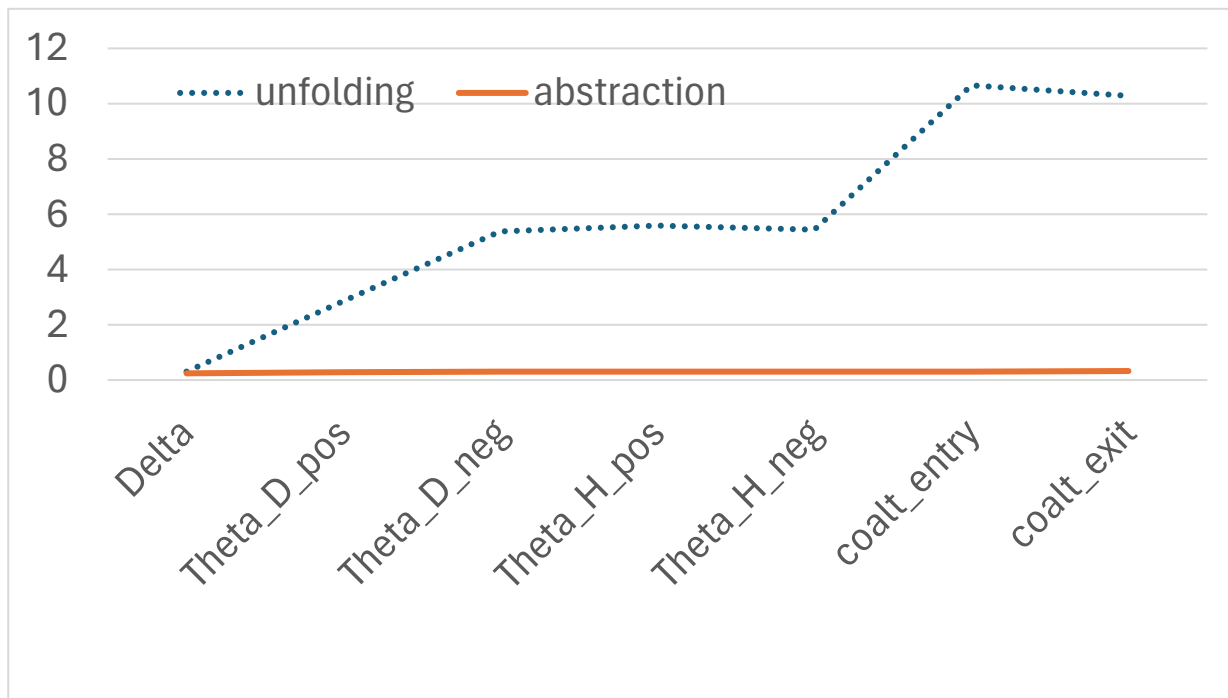
Before:



Now:



Function calls abstract analysis: experiments

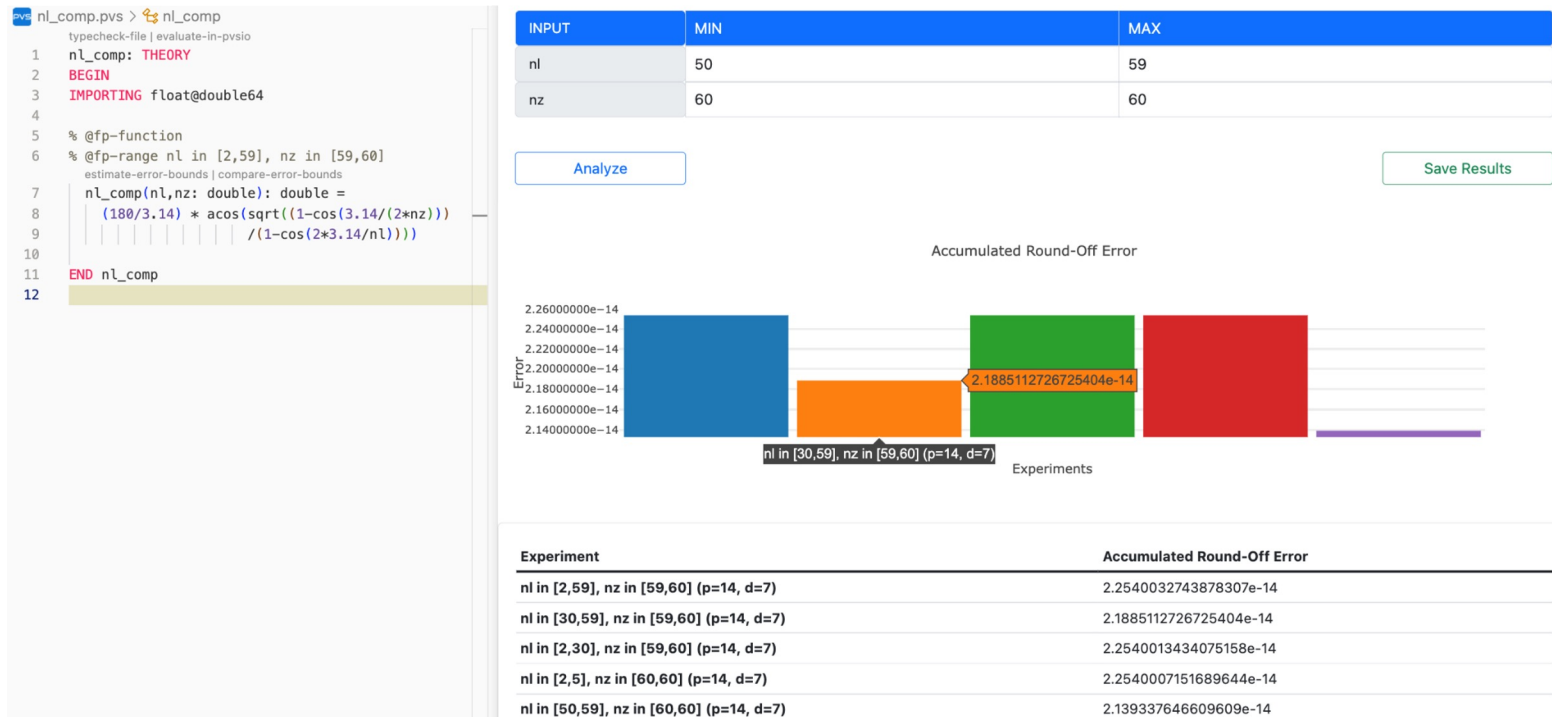


VSCoDe-PRECiSA

<https://github.com/nasa/PRECiSA/tree/master/vscode-precisa>

Integration of PRECiSA into VSCoDe

VSCoDe-PRECiSA: Round-off Error Analysis GUI



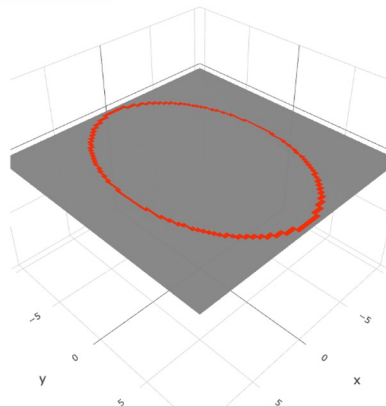
VSCoDe-PRECiSA: Instability Analysis GUI

- Problem: rounding error in boolean guards of if-then-else
- Detect the values that may provoke divergences in the control flow

```
pvs ellipse.pvs > ...
typecheck-file | evaluate-in-pvsio
1 ellipse: THEORY
2 BEGIN
3 IMPORTING float@aerr754dp
4
5 % @fp-function
6 % @fp-range x in [1,300], y in [1,300]
7 estimate-error-bounds | compare-error-bounds
8 pointInEllipse(x,y: double): double =
9     IF x*x/4 + y*y/9 <= 10 THEN 1 ELSE -1 ENDIF
10 END ellipse
```

INPUT	MIN	MAX
x	-10	10
y	-10	10

Analyze



INPUT	MIN	MAX
x	0	5
y	0	5

Analyze

Paving results indicate all Ok!

— This means that the function does not contain unstable guards

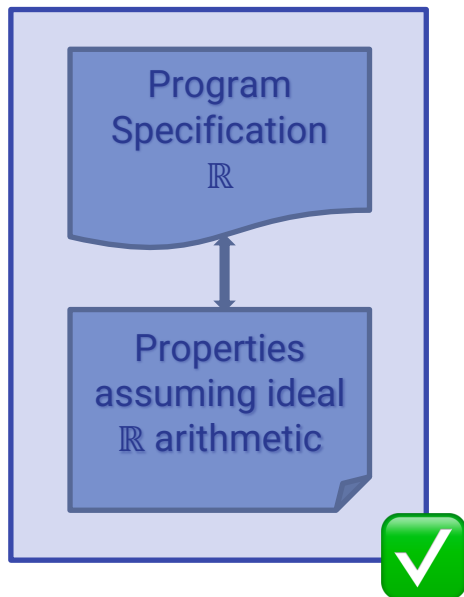


ReFlow

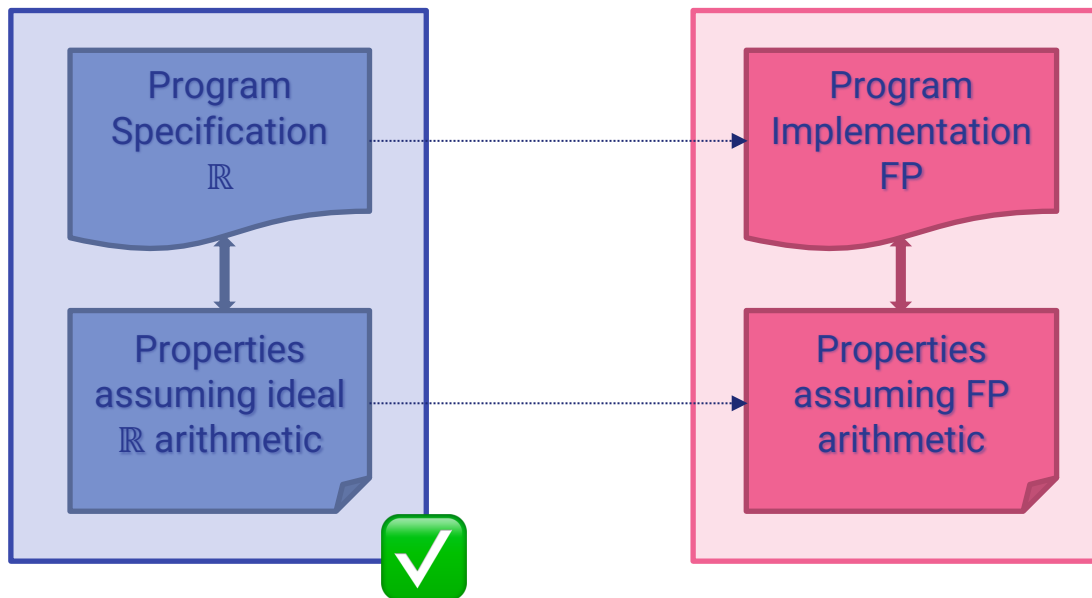
<https://github.com/nasa/reflow>

C floating-point code extraction
from PVS specification

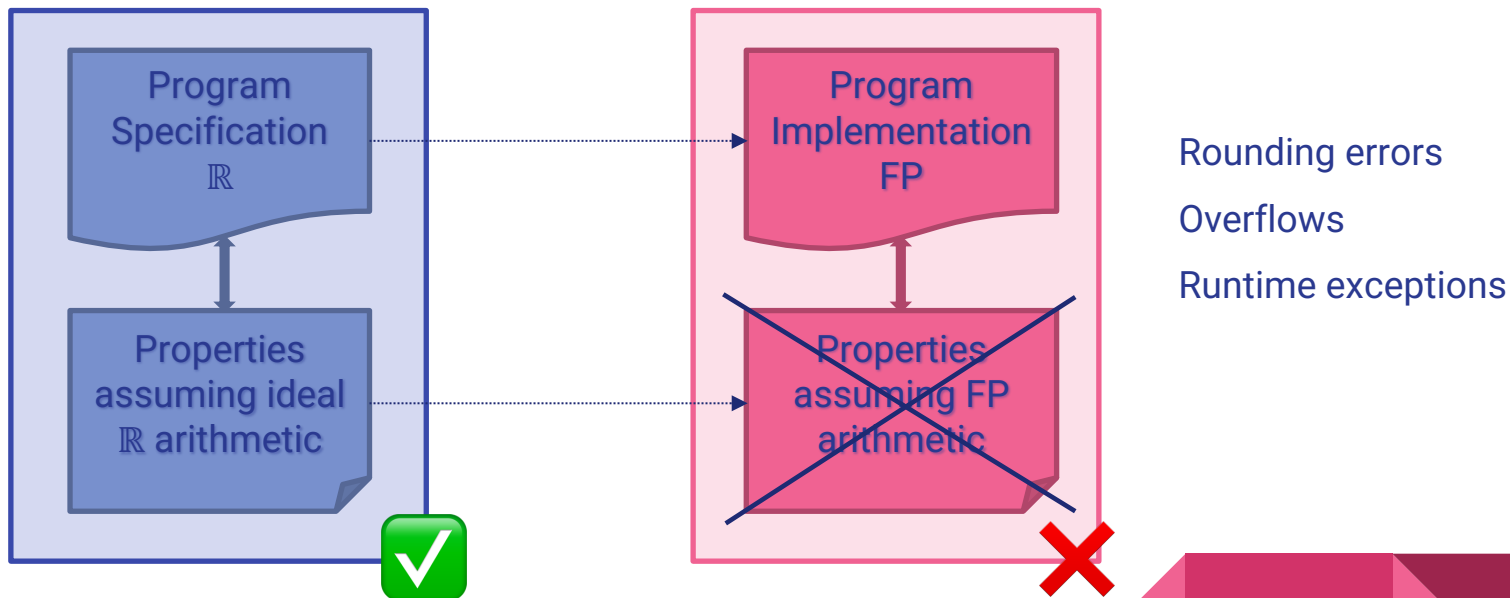
Real number $\neq$ Floating-point arithmetic



Real number $\neq$ Floating-point arithmetic



Real number $\neq$ Floating-point arithmetic



Idea: automatically extracting a FP implementation with formal guarantees on the rounding error

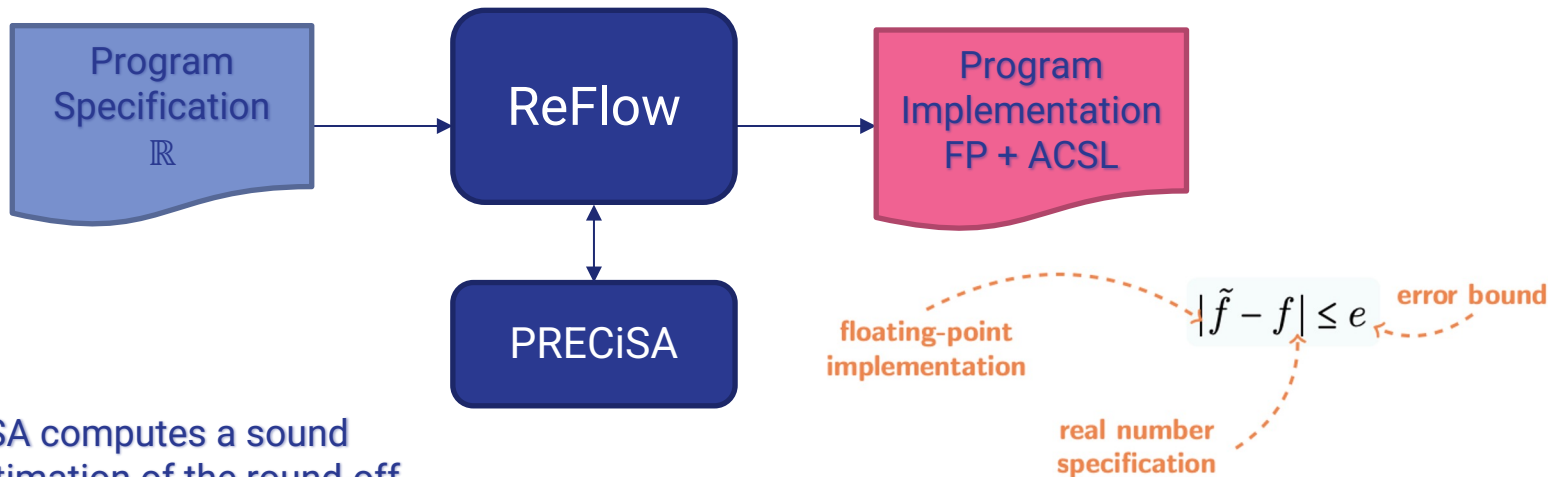


ReFlow automatically generates a C floating-point implementation from a PVS real number specification



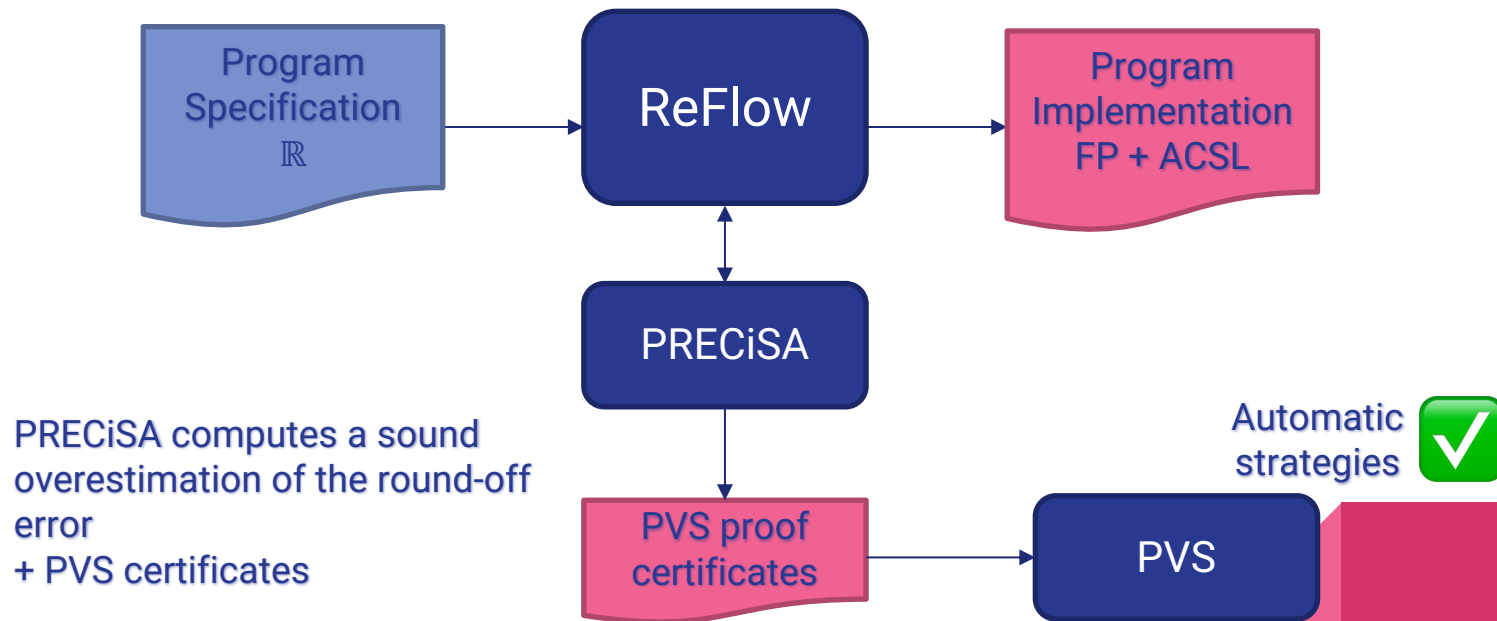
ReFlow

ACSL annotations model the discrepancies between real specification and floating-point implementation



PRECiSA computes a sound overestimation of the round-off error

ACSL annotations model the discrepancies between real specification and floating-point implementation



Example: eps_line

```
eps_line(sx,vx,sy,vy: real): real = ((sx*vx) + (sy*vy)) * ((sx*vx) - (sy*vy))
```

Example: eps_line

```
eps_line(sx,vx,sy,vy: real): real = ((sx*vx) + (sy*vy)) * ((sx*vx) - (sy*vy))
```



```
/*@
logic real eps_line (real sx, real vx, real sy, real vy) =
  (((sx * vx) + (sy * vy)) * ((sx * vx) - (sy * vy)));

ensures \forall real sx, real vx, real sy, real vy;
  -60000 <= sx && sx <= 60000 && -60000 <= vx && vx <= 60000 &&
  -60000 <= sy && sy <= 60000 && -60000 <= vy && vy <= 60000 &&
  \abs(sx_d - sx) <= ulp_dp(sx)/2 && \abs(vx_d - vx) <= ulp_dp(vx)/2 &&
  \abs(sy_d - sy) <= ulp_dp(sy)/2 && \abs(vy_d - vy) <= ulp_dp(vy)/2
  ==> \abs((\result - eps_line(sx, vx, sy, vy)) <= 0x1.db070f4580002p14 ;
*/
double eps_line_fp (double sx_d, double vx_d, double sy_d, double vy_d) {
  | return ((sx_d * vx_d) + (sy_d * vy_d)) * ((sx_d * vx_d) - (sy_d * vy_d));
}
```

Example: eps_line

```
eps_line(sx,vx,sy,vy: real): real = ((sx*vx) + (sy*vy)) * ((sx*vx) - (sy*vy))
```



```
/*@  
logic real eps_line (real sx, real vx, real sy, real vy) =  
  (((sx * vx) + (sy * vy)) * ((sx * vx) - (sy * vy)));  
  
ensures \forall real sx, real vx, real sy, real vy;  
-60000 <= sx && sx <= 60000 && -60000 <= vx && vx <= 60000 &&  
-60000 <= sy && sy <= 60000 && -60000 <= vy && vy <= 60000 &&  
\abs(sx_d - sx) <= ulp_dp(sx)/2 && \abs(vx_d - vx) <= ulp_dp(vx)/2 &&  
\abs(sy_d - sy) <= ulp_dp(sy)/2 && \abs(vy_d - vy) <= ulp_dp(vy)/2  
==> \abs((\result - eps_line(sx, vx, sy, vy)) <= 0x1.db070f4580002p14 ;  
*/  
double eps_line_fp (double sx_d, double vx_d, double sy_d, double vy_d) {  
  return ((sx_d * vx_d) + (sy_d * vy_d)) * ((sx_d * vx_d) - (sy_d * vy_d));  
}
```

Axiomatic real-valued program

Example: eps_line

```
eps_line(sx,vx,sy,vy: real): real = ((sx*vx) + (sy*vy)) * ((sx*vx) - (sy*vy))
```



```
/*@
logic real eps_line (real sx, real vx, real sy, real vy) =
  (((sx * vx) + (sy * vy)) * ((sx * vx) - (sy * vy)));

ensures \forall real sx, real vx, real sy, real vy;
-60000 <= sx && sx <= 60000 && -60000 <= vx && vx <= 60000 &&
-60000 <= sy && sy <= 60000 && -60000 <= vy && vy <= 60000 &&
\abs(sx_d - sx) <= ulp_dp(sx)/2 && \abs(vx_d - vx) <= ulp_dp(vx)/2 &&
\abs(sy_d - sy) <= ulp_dp(sy)/2 && \abs(vy_d - vy) <= ulp_dp(vy)/2
==> \abs(\result - eps_line(sx, vx, sy, vy)) <= 0x1.db070f4580002p14 ;
*/
```

Axiomatic real-valued program

```
double eps_line_fp (double sx_d, double vx_d, double sy_d, double vy_d) {
  return ((sx_d * vx_d) + (sy_d * vy_d)) * ((sx_d * vx_d) - (sy_d * vy_d));
}
```

Floating-point implementation

Example: eps_line

```
eps_line(sx,vx,sy,vy: real): real = ((sx*vx) + (sy*vy)) * ((sx*vx) - (sy*vy))
```



```
/*@
logic real eps_line (real sx, real vx, real sy, real vy) =
  (((sx * vx) + (sy * vy)) * ((sx * vx) - (sy * vy)));

ensures \forall real sx, real vx, real sy, real vy;
-60000 <= sx && sx <= 60000 && -60000 <= vx && vx <= 60000 &&
-60000 <= sy && sy <= 60000 && -60000 <= vy && vy <= 60000 &&
\abs(sx_d - sx) <= ulp_dp(sx)/2 && \abs(vx_d - vx) <= ulp_dp(vx)/2 &&
\abs(sy_d - sy) <= ulp_dp(sy)/2 && \abs(vy_d - vy) <= ulp_dp(vy)/2
==> \abs(\result - eps_line(sx, vx, sy, vy)) <= 0x1.db070f4580002p14 ;
*/
double eps_line_fp (double sx_d, double vx_d, double sy_d, double vy_d) {
  return ((sx_d * vx_d) + (sy_d * vy_d)) * ((sx_d * vx_d) - (sy_d * vy_d));
}
```

Axiomatic real-valued program

Initial ranges

Floating-point implementation

Example: eps_line

```
eps_line(sx,vx,sy,vy: real): real = ((sx*vx) + (sy*vy)) * ((sx*vx) - (sy*vy))
```



```
/*@
logic real eps_line (real sx, real vx, real sy, real vy) =
  (((sx * vx) + (sy * vy)) * ((sx * vx) - (sy * vy)));

ensures \forall real sx, real vx, real sy, real vy;
  -60000 <= sx && sx <= 60000 && -60000 <= vx && vx <= 60000 &&
  -60000 <= sy && sy <= 60000 && -60000 <= vy && vy <= 60000 &&
  \abs(sx_d - sx) <= ulp_dp(sx)/2 && \abs(vx_d - vx) <= ulp_dp(vx)/2 &&
  \abs(sy_d - sy) <= ulp_dp(sy)/2 && \abs(vy_d - vy) <= ulp_dp(vy)/2
  ==> \abs(\result - eps_line(sx, vx, sy, vy)) <= 0x1.db070f4580002p14 ;
*/
double eps_line_fp (double sx_d, double vx_d, double sy_d, double vy_d) {
  return ((sx_d * vx_d) + (sy_d * vy_d)) * ((sx_d * vx_d) - (sy_d * vy_d));
}
```

Axiomatic real-valued program

Initial ranges

Vars are round-to the nearest

Floating-point implementation

Example: eps_line

```
eps_line(sx,vx,sy,vy: real): real = ((sx*vx) + (sy*vy)) * ((sx*vx) - (sy*vy))
```



```
/*@
logic real eps_line (real sx, real vx, real sy, real vy) =
  (((sx * vx) + (sy * vy)) * ((sx * vx) - (sy * vy)));

ensures \forall real sx, real vx, real sy, real vy;
  -60000 <= sx && sx <= 60000 && -60000 <= vx && vx <= 60000 &&
  -60000 <= sy && sy <= 60000 && -60000 <= vy && vy <= 60000 &&
  \abs(sx_d - sx) <= ulp_dp(sx)/2 && \abs(vx_d - vx) <= ulp_dp(vx)/2 &&
  \abs(sy_d - sy) <= ulp_dp(sy)/2 && \abs(vy_d - vy) <= ulp_dp(vy)/2
==> \abs(\result - eps_line(sx, vx, sy, vy)) <= 0x1.db070f4580002p14;
*/
double eps_line_fp (double sx_d, double vx_d, double sy_d, double vy_d) {
  return ((sx_d * vx_d) + (sy_d * vy_d)) * ((sx_d * vx_d) - (sy_d * vy_d));
}
```

Axiomatic real-valued program

Initial ranges

Vars are round-to the nearest

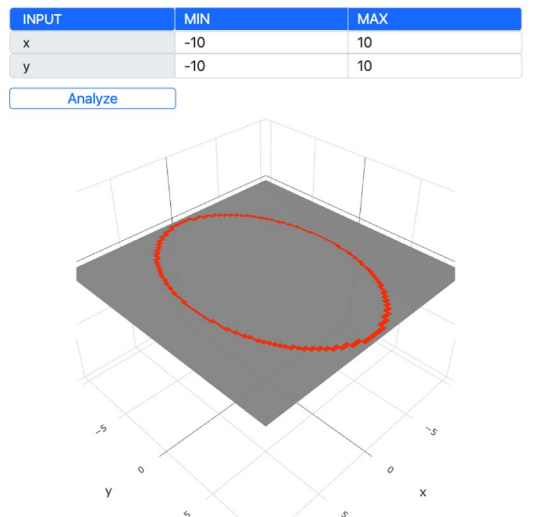
Maximum round-off error

Floating-point implementation

Problem: Unstable conditionals

- Rounding error in Boolean guards within if-then-else statements
- Floating-point control flow diverges w.r.t. the ideal real valued one

```
pvs ellipse.pvs > ...  
typecheck-file | evaluate-in-pvsio  
1 ellipse: THEORY  
2 BEGIN  
3 IMPORTING float@aeerr754dp  
4  
5 % @fp-function  
6 % @fp-range x in [1,300], y in [1,300]  
estimate-error-bounds | compare-error-bounds  
7 pointInEllipse(x,y: double): double =  
8   IF x*x/4 + y*y/9 <= 10 THEN 1 ELSE -1 ENDIF  
9  
10 END ellipse
```



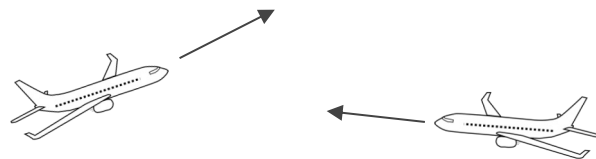
ReFlow instruments the program to detect when the floating-point control flow diverges w.r.t. the ideal one

tcoa(sz,vz) =

if $(sz * vz) < 0$ then
 $-(sz / vz)$

else

-1



ReFlow instruments the program to detect when the floating-point control flow diverges w.r.t. the ideal one

tcoa(sz,vz) =

if (sz*vz) < $-\epsilon$ then

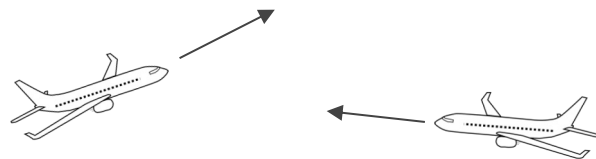
$-(sz/vz)$

elseif (sz*vz) $\geq \epsilon$ then

-1

else instability_warning

ϵ is a sound overestimation
of the error of the expression



ReFlow instruments the program to detect when the floating-point control flow diverges w.r.t. the ideal one

tcoa(sz,vz) =

if (sz*vz) < - ϵ then

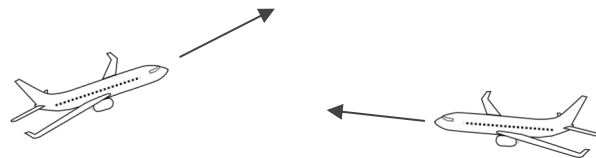
 -(sz/vz)

elseif (sz*vz) $\geq \epsilon$ then

 -1

else instability_warning

ϵ is a sound overestimation
of the error of the expression



The rounding error may affect
the evaluation of the guard

Example: Time to co-altitude

```
tcoa(sz,vz:real): real = IF (sz*vz < 0) THEN -(sz/vz) ELSE -1 ENDIF
```

Example: Time to co-altitude

```
tcoa(sz,vz:real): real = IF (sz*vz < 0) THEN -(sz/vz) ELSE -1 ENDIF
```



```
/*@
requires (0 <= E) ;
ensures \forall real sz, real vz; ((\abs(Dmul(sz_dp, vz_dp) - (sz * vz)) <= E
  ==> \result.isValid ==> (sz * vz < 0 && Dmul(sz_dp, vz_dp) < 0)
  || (sz * vz >= 0 && Dmul(sz_dp, vz_dp) >= 0) ;
*/
struct maybeDouble tcoa_fp (double sz_dp, double vz_dp, double E) {
  if (sz_dp * vz_dp < - E)
  { return someDouble(-sz_dp / vz_dp);
  } else { if (sz_dp * vz_dp >= E)
  { return someDouble(-1);
  } else { return instability_warning();}}
}

/*@
ensures \forall real sz, real vz; (0 <= sz) && (sz <= 1000) && (400 <= vz) && (vz <= 600) &&
\abs(sz_dp - sz) <= ulp_dp(sz)/2 && \abs(vz_dp - vz) <= ulp_dp(vz)/2 &&
\result.isValid ==> \abs(\result.value - tcoa(sz, vz)) <= 0x1.c7ae147ae147dp-51;
*/
struct maybeDouble tcoa_num (double sz_dp, double vz_dp) {
  return tcoa_fp (sz_dp, vz_dp, 0x1.4800000000001p-33);
}
```

Example: Time to co-altitude

```
tcoa(sz,vz:real): real = IF (sz*vz < 0) THEN -(sz/vz) ELSE -1 ENDIF
```



```
/*@
requires (0 <= E) ;
ensures \forall real sz, real vz; ((\abs(Dmul(sz_dp, vz_dp) - (sz * vz)) <= E
  ==> \result.isValid ==> (sz * vz < 0 && Dmul(sz_dp, vz_dp) < 0)
  || (sz * vz >= 0 && Dmul(sz_dp, vz_dp) >= 0) ;
*/
struct maybeDouble tcoa_fp (double sz_dp, double vz_dp, double E) {
  if (sz_dp * vz_dp < - E)
  { return someDouble(-sz_dp / vz_dp);
  } else { if (sz_dp * vz_dp >= E)
  { return someDouble(-1);
  } else { return instability_warning();}}
}

/*@
ensures \forall real sz, real vz; (0 <= sz) && (sz <= 1000) && (400 <= vz) && (vz <= 600) &&
\abs(sz_dp - sz) <= ulp_dp(sz)/2 && \abs(vz_dp - vz) <= ulp_dp(vz)/2 &&
\result.isValid ==> \abs(\result.value - tcoa(sz, vz)) <= 0x1.c7ae147ae147dp-51;
*/
struct maybeDouble tcoa_num (double sz_dp, double vz_dp) {
  return tcoa_fp (sz_dp, vz_dp, 0x1.4800000000001p-33);
}
```

Instrumented program

Example: Time to co-altitude

```
tcoa(sz,vz:real): real = IF (sz*vz < 0) THEN -(sz/vz) ELSE -1 ENDIF
```



```
/*@
requires (0 <= E) ;
ensures \forall real sz, real vz; ((\abs(Dmul(sz_dp, vz_dp) - (sz * vz)) <= E
==> \result.isValid ==> (sz * vz < 0 && Dmul(sz_dp, vz_dp) < 0)
|| (sz * vz >= 0 && Dmul(sz_dp, vz_dp) >= 0) ;
*/
struct maybeDouble tcoa_fp (double sz_dp, double vz_dp, double E) {
  if (sz_dp * vz_dp < - E)
  { return someDouble(-sz_dp / vz_dp);
  } else { if (sz_dp * vz_dp >= E)
  { return someDouble(-1);
  } else { return instability_warning();}}
}
/*@
ensures \forall real sz, real vz; (0 <= sz) && (sz <= 1000) && (400 <= vz) && (vz <= 600) &&
\abs(sz_dp - sz) <= ulp_dp(sz)/2 && \abs(vz_dp - vz) <= ulp_dp(vz)/2 &&
\result.isValid ==> \abs(\result.value - tcoa(sz, vz)) <= 0x1.c7ae147ae147dp-51;
*/
struct maybeDouble tcoa_num (double sz_dp, double vz_dp) {
  return tcoa_fp (sz_dp, vz_dp, 0x1.4800000000001p-33);
}
```

ACSL contract

Instrumented program

Example: Time to co-altitude

```
tcoa(sz,vz:real): real = IF (sz*vz < 0) THEN -(sz/vz) ELSE -1 ENDIF
```



```
/*@
requires (0 <= E) ;
ensures \forall real sz, real vz; ((\abs(Dmul(sz_dp, vz_dp) - (sz * vz)) <= E
==> \result.isValid ==> (sz * vz < 0 && Dmul(sz_dp, vz_dp) < 0)
|| (sz * vz >= 0 && Dmul(sz_dp, vz_dp) >= 0) ;
*/
struct maybeDouble tcoa_fp (double sz_dp, double vz_dp, double E) {
  if (sz_dp * vz_dp < - E)
  { return someDouble(-sz_dp / vz_dp);
  } else { if (sz_dp * vz_dp >= E)
  { return someDouble(-1);
  } else { return instability_warning();}}
}

/*@
ensures \forall real sz, real vz; (0 <= sz) && (sz <= 1000) && (400 <= vz) && (vz <= 600) &&
\abs(sz_dp - sz) <= ulp_dp(sz)/2 && \abs(vz_dp - vz) <= ulp_dp(vz)/2 &&
\result.isValid ==> \abs(\result.value - tcoa(sz, vz)) <= 0x1.c7ae147ae147dp-51;
*/
struct maybeDouble tcoa_num (double sz_dp, double vz_dp) {
  return tcoa_fp (sz_dp, vz_dp, 0x1.4800000000001p-33);
}
```

ACSL contract

Instrumented program

generic

Example: Time to co-altitude

```
tcoa(sz,vz:real): real = IF (sz*vz < 0) THEN -(sz/vz) ELSE -1 ENDIF
```



```
/*@
requires (0 <= E) ;
ensures \forall real sz, real vz; ((\abs(Dmul(sz_dp, vz_dp) - (sz * vz)) <= E
==> \result.isValid ==> (sz * vz < 0 && Dmul(sz_dp, vz_dp) < 0)
|| (sz * vz >= 0 && Dmul(sz_dp, vz_dp) >= 0) ;
*/
struct maybeDouble tcoa_fp (double sz_dp, double vz_dp, double E) {
  if (sz_dp * vz_dp < - E)
  { return someDouble(-sz_dp / vz_dp);
  } else { if (sz_dp * vz_dp >= E)
  { return someDouble(-1);
  } else { return instability_warning();}}
}

/*@
ensures \forall real sz, real vz; (0 <= sz) && (sz <= 1000) && (400 <= vz) && (vz <= 600) &&
\abs(sz_dp - sz) <= ulp_dp(sz)/2 && \abs(vz_dp - vz) <= ulp_dp(vz)/2 &&
\result.isValid ==> \abs(\result.value - tcoa(sz, vz)) <= 0x1.c7ae147ae147dp-51;
*/
struct maybeDouble tcoa_num (double sz_dp, double vz_dp) {
  return tcoa_fp (sz_dp, vz_dp, 0x1.4800000000001p-33);
}
```

ACSL contract

instrumented program

function call to tcoa_fp

generic

Example: Time to co-altitude

```
tcoa(sz,vz:real): real = IF (sz*vz < 0) THEN -(sz/vz) ELSE -1 ENDIF
```



```
/*@  
requires (0 <= E) ;  
ensures \forall real sz, real vz; ((\abs(Dmul(sz_dp, vz_dp) - (sz * vz)) <= E  
==> \result.isValid ==> (sz * vz < 0 && Dmul(sz_dp, vz_dp) < 0)  
|| (sz * vz >= 0 && Dmul(sz_dp, vz_dp) >= 0) ;  
*/
```

```
struct maybeDouble tcoa_fp (double sz_dp, double vz_dp, double E) {  
  if (sz_dp * vz_dp < - E)  
  { return someDouble(-sz_dp / vz_dp);  
  } else { if (sz_dp * vz_dp >= E)  
  { return someDouble(-1);  
  } else { return instability_warning();}}  
}
```

```
/*@  
ensures \forall real sz, real vz; (0 <= sz) && (sz <= 1000) && (400 <= vz) && (vz <= 600) &&  
\abs(sz_dp - sz) <= ulp_dp(sz)/2 && \abs(vz_dp - vz) <= ulp_dp(vz)/2 &&  
\result.isValid ==> \abs(\result.value - tcoa(sz, vz)) <= 0x1.c7ae147ae147dp-51;  
*/
```

```
struct maybeDouble tcoa_num (double sz_dp, double vz_dp) {  
  return tcoa_fp (sz_dp, vz_dp, 0x1.4800000000001p-33);  
}
```

ACSL contract

Instrumented program

ACSL contract

function call to tcoa_fp

generic

Example: Time to co-altitude

```
tcoa(sz,vz:real): real = IF (sz*vz < 0) THEN -(sz/vz) ELSE -1 ENDIF
```



```
/*@  
requires (0 <= E) ;  
ensures \forall real sz, real vz; ((\abs(Dmul(sz_dp, vz_dp) - (sz * vz)) <= E  
==> \result.isValid ==> (sz * vz < 0 && Dmul(sz_dp, vz_dp) < 0)  
|| (sz * vz >= 0 && Dmul(sz_dp, vz_dp) >= 0) ;  
*/
```

```
struct maybeDouble tcoa_fp (double sz_dp, double vz_dp, double E) {  
  if (sz_dp * vz_dp < - E)  
  { return someDouble(-sz_dp / vz_dp);  
  } else { if (sz_dp * vz_dp >= E)  
  { return someDouble(-1);  
  } else { return instability_warning();}}  
}
```

```
/*@  
ensures \forall real sz, real vz; (0 <= sz) && (sz <= 1000) && (400 <= vz) && (vz <= 600) &&  
\abs(sz_dp - sz) <= ulp_dp(sz)/2 && \abs(vz_dp - vz) <= ulp_dp(vz)/2 &&  
\result.isValid ==> \abs(\result.value - tcoa(sz, vz)) <= 0x1.c7ae147ae147dp-51;  
*/
```

```
struct maybeDouble tcoa_num (double sz_dp, double vz_dp) {  
  return tcoa_fp (sz_dp, vz_dp, 0x1.4800000000001p-33);  
}
```

ACSL contract

instrumented program

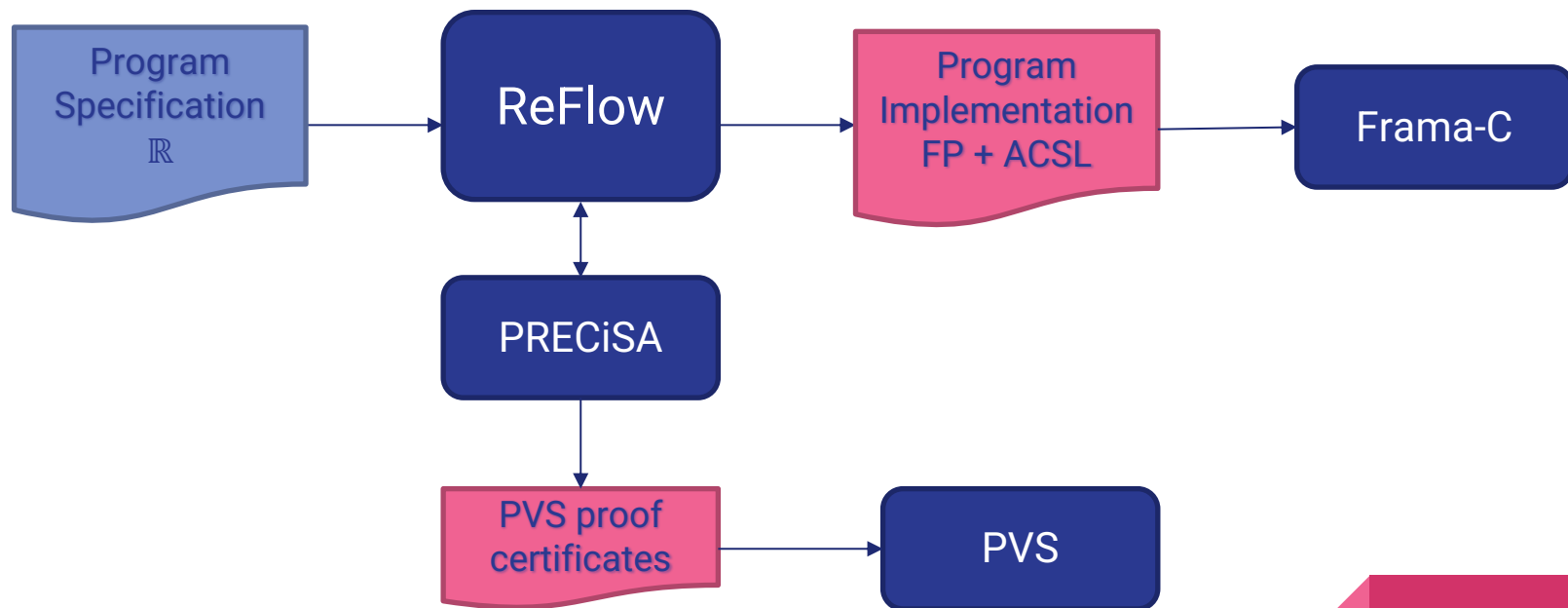
ACSL contract

function call to tcoa_fp

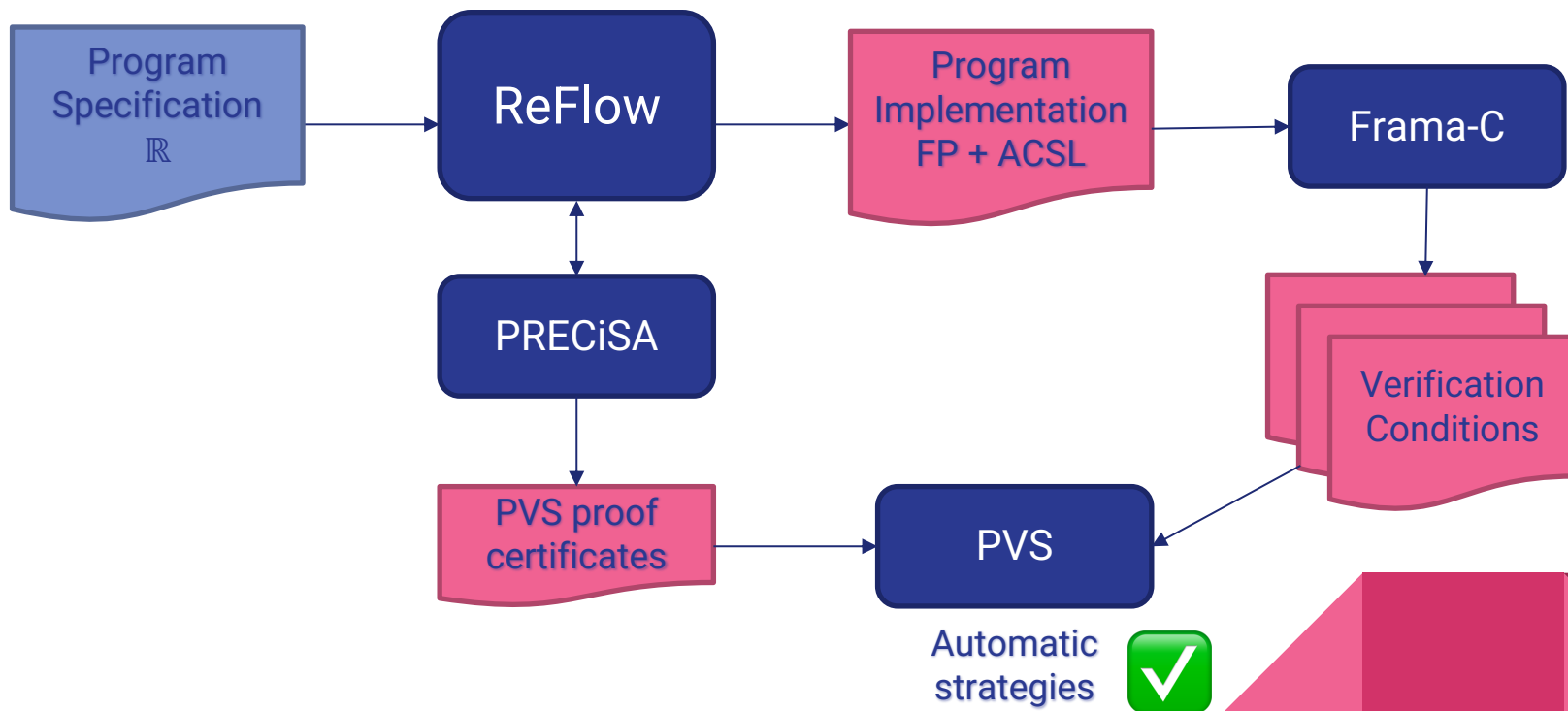
generic

concrete

Automatic verification in Frama-C and PVS



Automatic verification in Frama-C and PVS



Time to co-altitude proof certificate

Users > ltitolo > Desktop > precisa > public > benchmarks > code-generation > daidalus > tcoa >  tcoa_num_cert.pvs >  tcoa_num_ce

```
1  % This file is automatically generated by PRECISA
2
3  % maxDepth: 7 , prec: 10-14
4
5  typecheck-file | evaluate-in-pvsio
6  tcoa_num_cert: THEORY
7  BEGIN
8  IMPORTING cert_tcoa, PRECISA@bbiasp, PRECISA@bbiadp, PRECISA@strategies
9
10 %|- *_TCC* : PROOF
11 %|- (precisa-gen-cert-tcc)
12 %|- QED
13
14 % Floating-Point Results: 0, neg_double(div_double(sz, vz))
15 % Real Results: -((r_sz / r_vz)), 0
16 % Control Flow: Stable
17 prove | status-proofchain | show-prooflite
18 tcoa_fp_c_0 : LEMMA
19 FORALL(r_sz, r_vz: real, sz: double, vz: double):
20   abs(safe_prjct_double(sz) - r_sz) <= ulp_dp(r_sz)/2 AND abs(safe_prjct_double(vz) - r_vz) <= ulp_dp(r_vz)/2
21   AND (((r_vz /= 0) AND ((r_sz * r_vz) < 0)) AND ((vz /= integerToDouble(0)) AND (mul_double(sz, vz) <
22     integerToDouble(0)))) OR (NOT(((r_sz * r_vz) < 0)) AND NOT((mul_double(sz, vz) < integerToDouble(0))))
23   AND r_sz ## [1,1000] AND r_vz ## [1,1000] AND
24   finite_double?(tcoa_fp(sz, vz)) AND finite_double?(sz) AND finite_double?(vz) AND finite_double?(mul_double
25     (sz, vz)) AND finite_double?(integerToDouble(0))
26 IMPLIES
27   abs(safe_prjct_double(tcoa_fp(sz, vz)) - tcoa(r_sz, r_vz)) <= 6876345720111303 / 2475880078570760549798248448
28
29 %|- tcoa_fp_c_0 : PROOF
30 %|- (prove-concrete-lemma tcoa_fp_0 14 7)
31 %|- QED
32
```

Time to co-altitude proof certificate

```
Users > ltitolo > Desktop > precisa > public > benchmarks > code-generation > daidalus > tcoa > pvs tcoa_num_cert.pvs > tcoa_num_ce

1  % This file is automatically generated by PRECISA
2
3  % maxDepth: 7 , prec: 10^-14
4
5  typecheck-file | evaluate-in-pvsio
6  tcoa_num_cert: THEORY
7  BEGIN
8  IMPORTING cert_tcoa, PRECISA@bbiasp, PRECISA@bbiadp, PRECISA@strategies
9
10 %|- *_TCC* : PROOF
11 %|- (precisa-gen-cert-tcc)
12 %|- QED
13
14 % Floating-Point Results: 0, neg_double(div_double(sz, vz))
15 % Real Results: -((r_sz / r_vz)), 0
16 % Control Flow: Stable
17 prove | status-proofchain | show-proofite
18 tcoa_fp_c_0 : LEMMA
19 FORALL(r_sz, r_vz: real, sz: double, vz: double):
20   abs(safe_prjct_double(sz) - r_sz) <= ulp_dp(r_sz)/2 AND abs(safe_prjct_double(vz) - r_vz) <= ulp_dp(r_vz)/2
21   AND (((r_vz /= 0) AND ((r_sz * r_vz) < 0)) AND ((vz /= integerToDouble(0)) AND (mul_double(sz, vz) <
22     integerToDouble(0)))) OR (NOT(((r_sz * r_vz) < 0)) AND NOT((mul_double(sz, vz) < integerToDouble(0))))
23   AND r_sz ## [1,1000] AND r_vz ## [1,1000] AND
24   finite_double?(tcoa_fp(sz, vz)) AND finite_double?(sz) AND finite_double?(vz) AND finite_double?(mul_double
25     (sz, vz)) AND finite_double?(integerToDouble(0))
26 IMPLIES
27   abs(safe_prjct_double(tcoa_fp(sz, vz)) - tcoa(r_sz, r_vz)) <= 6876345720111303 / 2475880078570760549798248448
28
29 %|- tcoa_fp_c_0 : PROOF
30 %|- (prove-concrete-lemma tcoa_fp_0 14 7)
31 %|- QED
32
```

Overflow detection

Rounding error

Automatic strategy
a PVS formally verified
optimization algorithm is used

Time to co-altitude proof certificate

Users > ltitolo > Desktop > precisa > public > benchmarks > code-generation > daidalus > tcoa > pvs tcoa_num_cert.pvs > tcoa_num_ce

```
1  % This file is automatically generated by PRECISA
2
3  % maxDepth: 7 , prec: 10^-14
4
5  typecheck-file | evaluate-in-pvsio
6  tcoa_num_cert: THEORY
7  BEGIN
8  IMPORTING cert_tcoa, PRECISA@bbiasp, PRECISA@bbiadp, PRECISA@strategies
9
10 %|- *_TCC* : PROOF
11 %|- (precisa-gen-cert-tcc)
12 %|- QED
13
14 % Floating-Point Results: 0, neg_double(div_double(sz, vz))
15 % Real Results: -((r_sz / r_vz)), 0
16 % Control Flow: Stable
17 prove | status-proofchain | show-proofite
18 tcoa_fp_c_0 : LEMMA
19 FORALL(r_sz, r_vz: real, sz: double, vz: double):
20   abs(safe_prjct_double(sz) - r_sz) <= ulp_dp(r_sz)/2 AND abs(safe_prjct_double(vz) - r_vz) <= ulp_dp(r_vz)/2
21   AND (((r_vz /= 0) AND ((r_sz * r_vz) < 0)) AND ((vz /= integerToDouble(0)) AND (mul_double(sz, vz) <
22     integerToDouble(0)))) OR (NOT(((r_sz * r_vz) < 0)) AND NOT((mul_double(sz, vz) < integerToDouble(0))))
23   AND r_sz ## [1,1000] AND r_vz ## [1,1000] AND
24   finite_double?(tcoa_fp(sz, vz)) AND finite_double?(sz) AND finite_double?(vz) AND finite_double?(mul_double
25     (sz, vz)) AND finite_double?(integerToDouble(0))
26 IMPLIES
27   abs(safe_prjct_double(tcoa_fp(sz, vz)) - tcoa(r_sz, r_vz)) <= 6876345720111303 / 2475880078570760549798248448
28
29 %|- tcoa_fp_c_0 : PROOF
30 %|- (prove-concrete-lemma tcoa_fp_0 14 7)
31 %|- QED
```

Overflow detection

Rounding error

Time to co-altitude proof certificate

Users > ltitolo > Desktop > precisa > public > benchmarks > code-generation > daidalus > tcoa > pvs tcoa_num_cert.pvs > tcoa_num_ce

```
1  % This file is automatically generated by PRECISA
2
3  % maxDepth: 7 , prec: 10^-14
4
5  typecheck-file | evaluate-in-pvsio
6  tcoa_num_cert: THEORY
7  BEGIN
8  IMPORTING cert_tcoa, PRECISA@bbiasp, PRECISA@bbiadp, PRECISA@strategies
9
10 %|- *_TCC* : PROOF
11 %|- (precisa-gen-cert-tcc)
12 %|- QED
13
14 % Floating-Point Results: 0, neg_double(div_double(sz, vz))
15 % Real Results: -((r_sz / r_vz)), 0
16 % Control Flow: Stable
17 prove | status-proofchain | show-proofite
18 tcoa_fp_c_0 : LEMMA
19 FORALL(r_sz, r_vz: real, sz: double, vz: double):
20   abs(safe_prjct_double(sz) - r_sz) <= ulp_dp(r_sz)/2 AND abs(safe_prjct_double(vz) - r_vz) <= ulp_dp(r_vz)/2
21   AND (((r_vz /= 0) AND ((r_sz * r_vz) < 0)) AND ((vz /= integerToDouble(0)) AND (mul_double(sz, vz) <
22     integerToDouble(0)))) OR (NOT(((r_sz * r_vz) < 0)) AND NOT((mul_double(sz, vz) < integerToDouble(0))))
23   AND r_sz ## [1,1000] AND r_vz ## [1,1000] AND
24   finite_double?(tcoa_fp(sz, vz)) AND finite_double?(sz) AND finite_double?(vz) AND finite_double?(mul_double
25     (sz, vz)) AND finite_double?(integerToDouble(0))
26 IMPLIES
27   abs(safe_prjct_double(tcoa_fp(sz, vz)) - tcoa(r_sz, r_vz)) <= 6876345720111303 / 2475880078570760549798248448
28
29 %|- tcoa_fp_c_0 : PROOF
30 %|- (prove-concrete-lemma tcoa_fp_0 14 7)
31 %|- QED
```

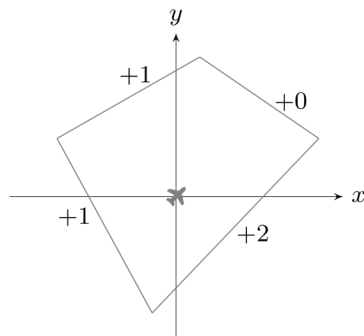
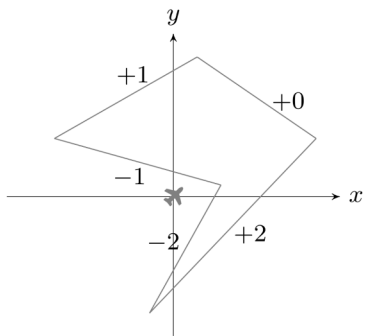
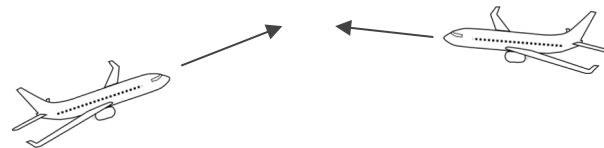
Overflow detection

Rounding error

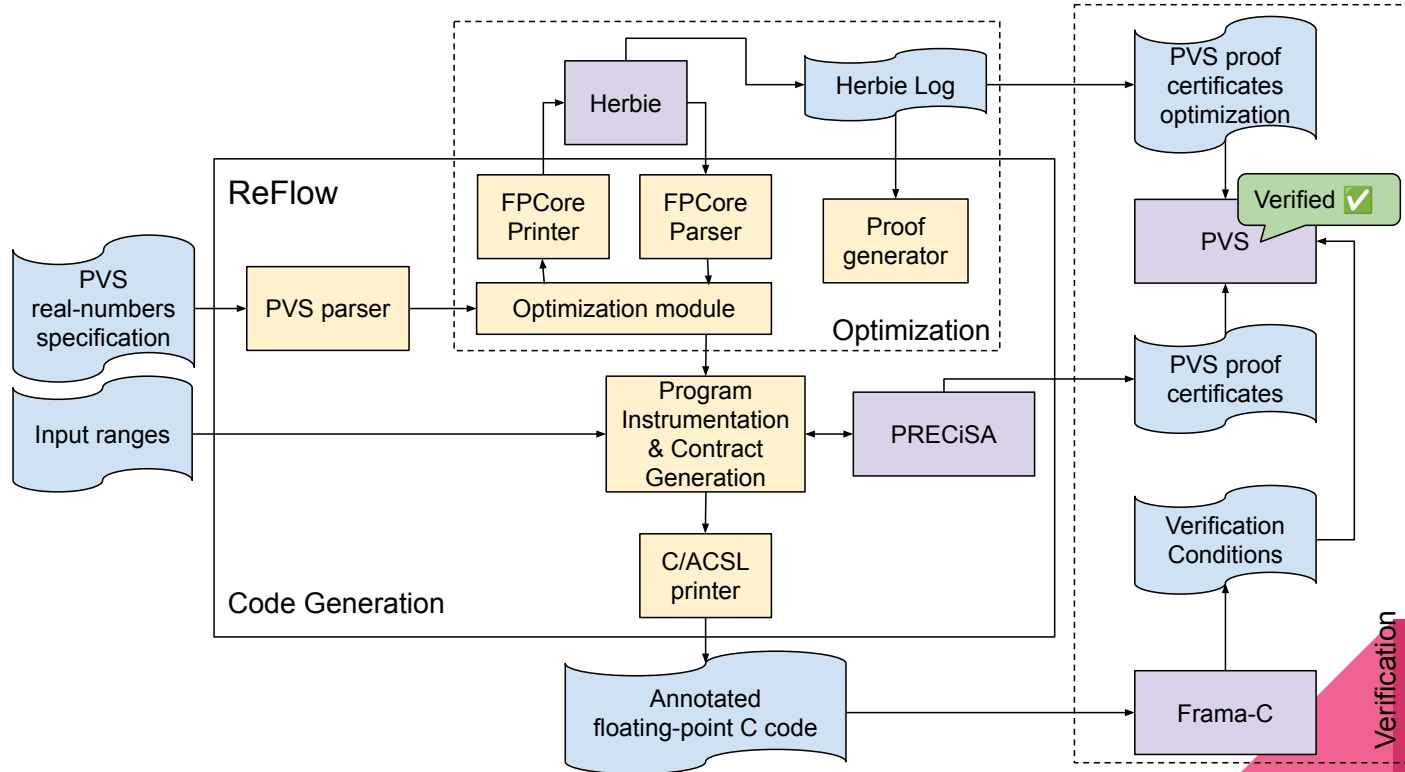
Automatic strategy
a PVS formally verified
optimization algorithm is used

Applications

- POLYCARP - Geofencing
- DAIDALUS - Detect-and-avoid
- ADS-B CPR - Compact Position Reporting Algorithm



Ongoing Integration with Herbie



Conclusion

- Automatic round-off error estimations + certificates
- Automatic generation and analysis of floating-point C implementations
- Proof strategies automatically discharge obligations in PVS
- No expertise required on floating-point nor theorem proving
- What's next?
 - Add support for more complex **loops**
 - Reduce the **complexity** of the ACSL annotations
 - More applications (NASA's DAIDALUS, Machine learning)

Thanks for your attention!

Laura Titolo

laura.titolo@nasa.gov

<https://lauratitolo.github.io/>

<https://github.com/nasa/cpr>

<https://github.com/nasa/PRECiSA>