ReFlow: from real number specifications to floating-point implementations

Laura Titolo
NIA/NASA LaRC

Joint work with Mariano Moscato (NIA), Marco Feliu (NIA), and Aaron Dutle (NASA)

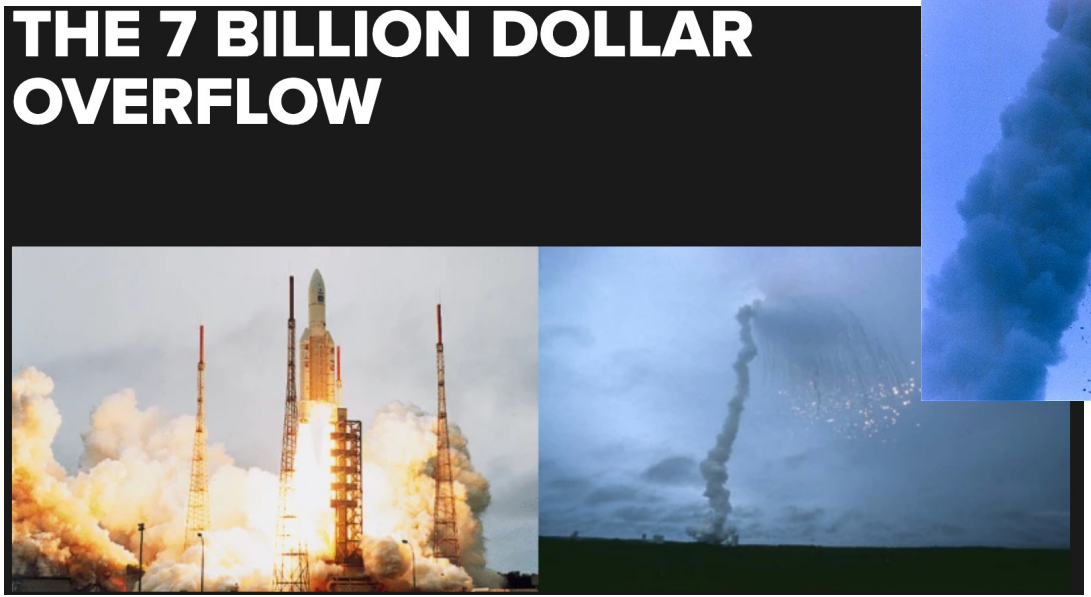
May 2023

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

Picture: Bernd vdB, Public domain, via Wikimedia Commons

Patriot missile failure (1991)

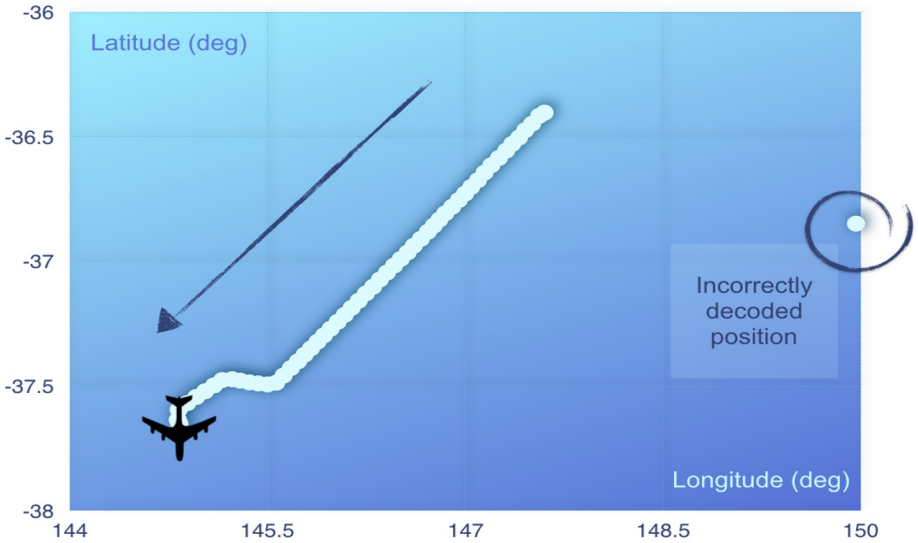
Floating-point errors may be dangerous in safety-critical applications



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)

Writing Floating-Point Code is hard

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

Writing Floating-Point Code is hard

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

Should be 0 if evaluated in exact real number arithmetic

Writing Floating-Point Code is hard

```
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

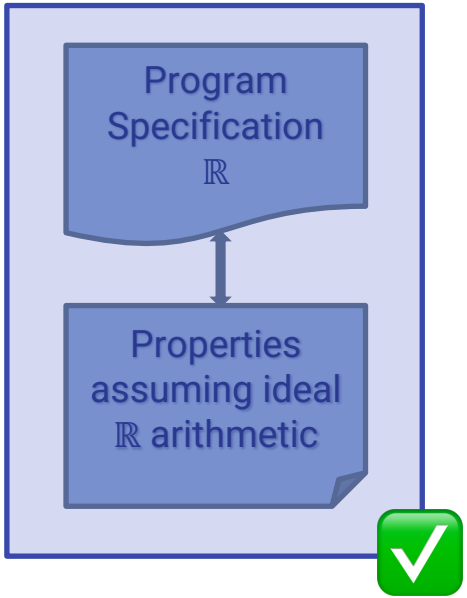
Writing Floating-Point Code is hard

```
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>
```

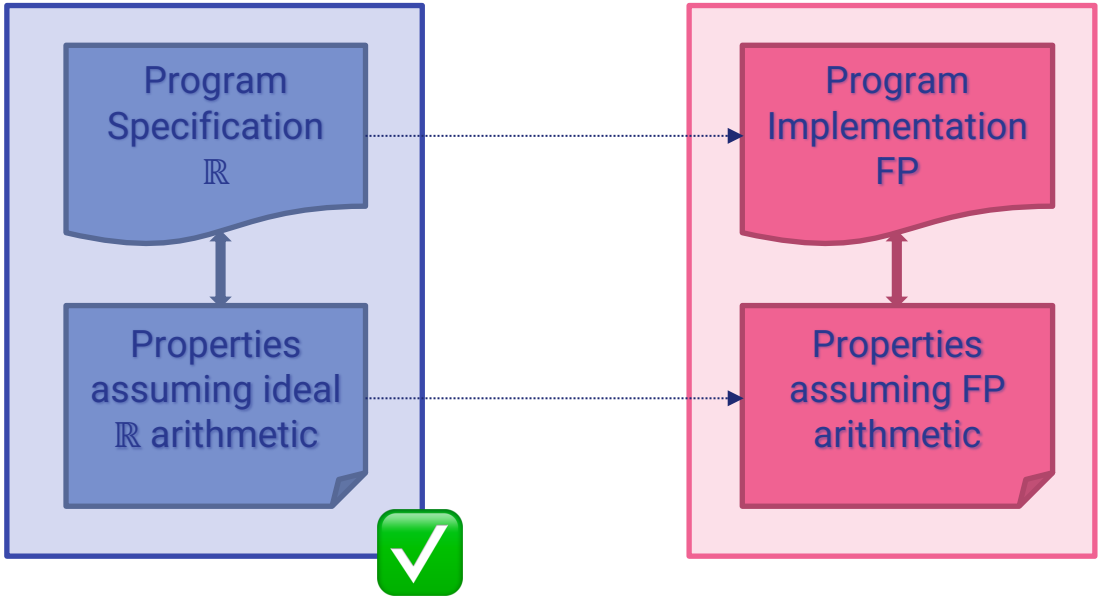
Should be 0 if evaluated in exact real number arithmetic

The else branch should be taken

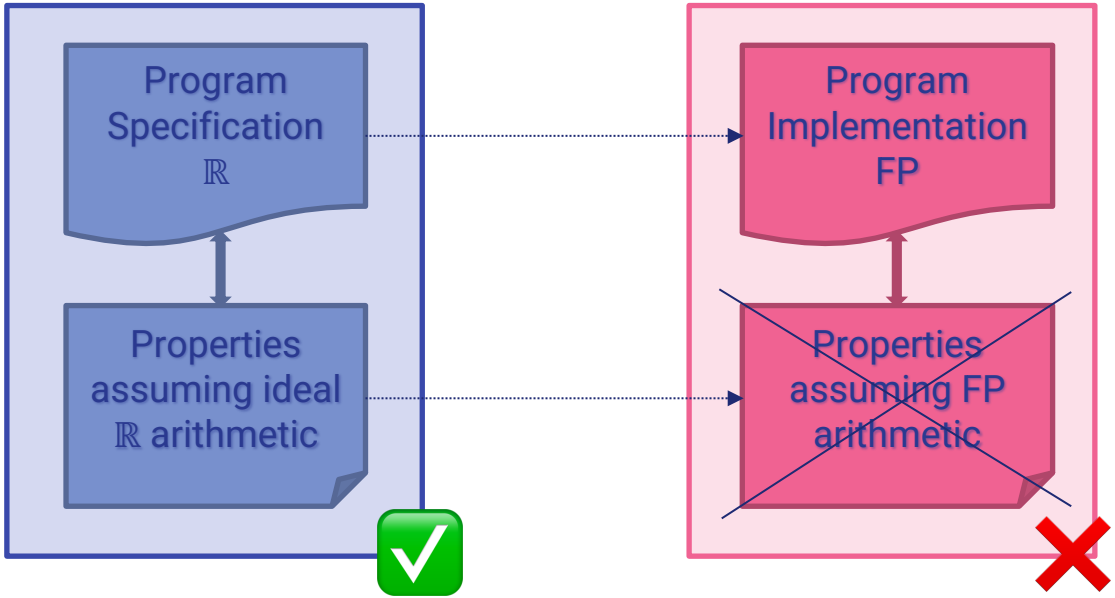
Real number ≠ Floating-point arithmetic



Real number ≠ Floating-point arithmetic



Real number ≠ Floating-point arithmetic



Rounding errors

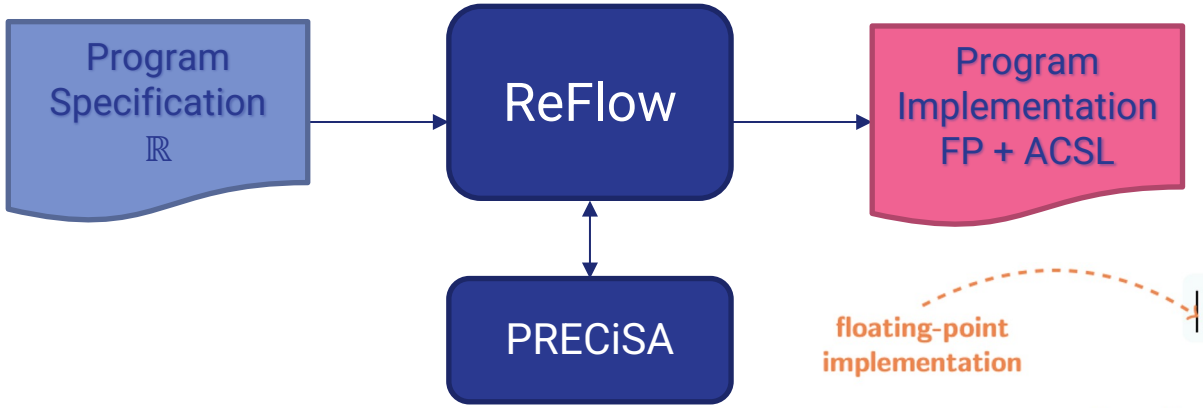
Overflows

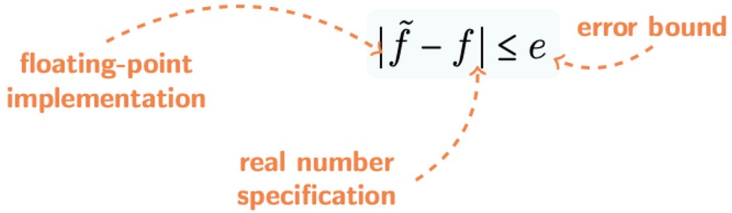
Runtime exceptions

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

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

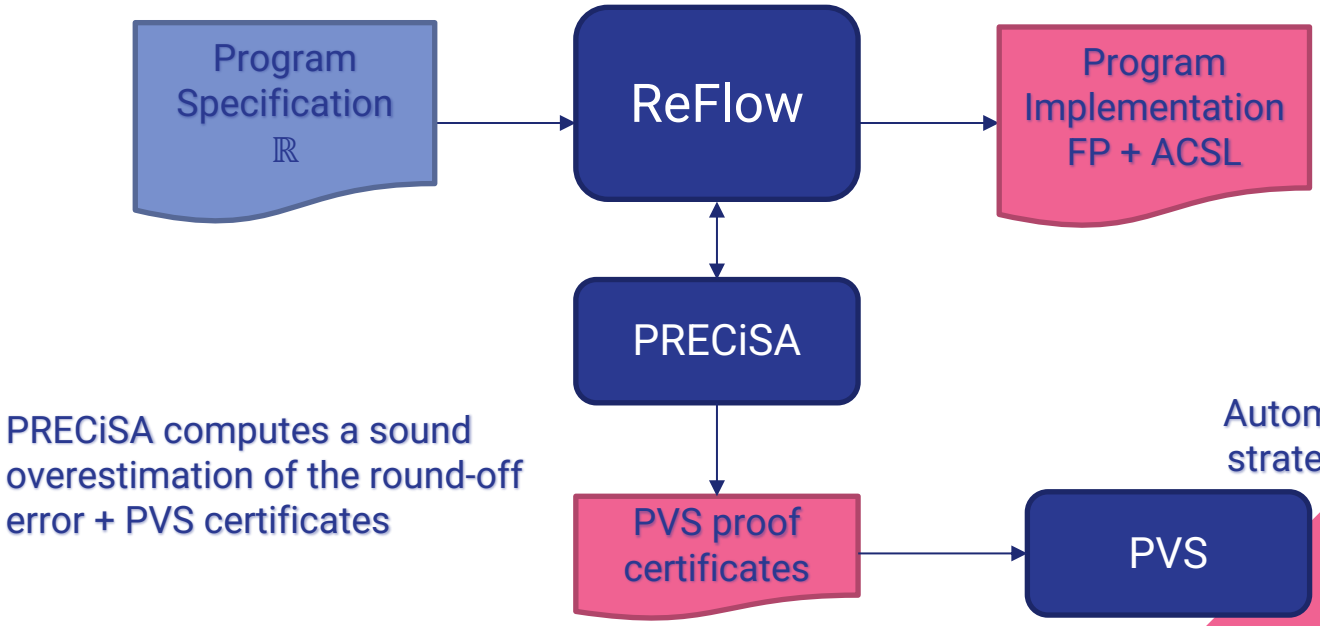
ACSL annotations model the discrepancies between real specification and floating-point implementation





PRECiSA computes a sound overestimation of the round-off error

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



Example: Code extraction eps_line

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

Example: Code extraction 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: Code extraction 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: Code extraction 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

Floating-point implementation

Example: Code extraction 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: Code extraction 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: Code extraction 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

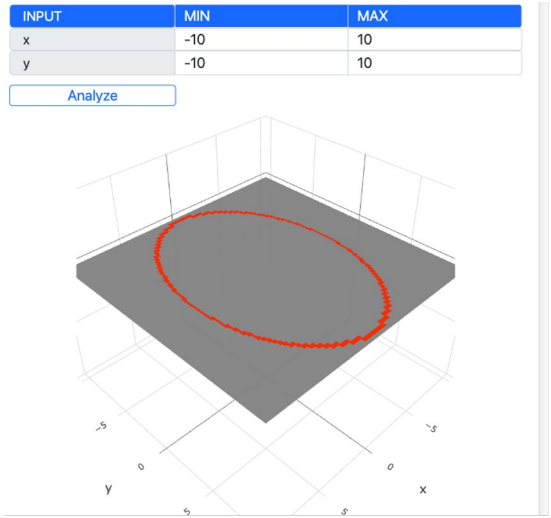
ellipse.pvs > ...

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]
    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
```

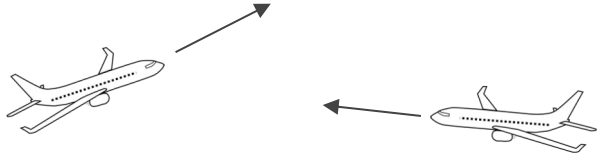
INPUT	MIN	MAX
x	-10	10
y	-10	10

Analyze



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

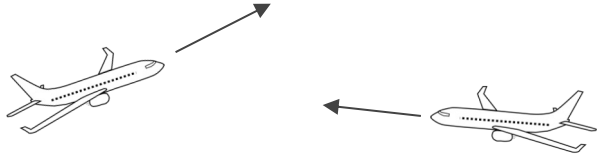
```
tcoa(sz,vz) =
        if (sz*vz) < 0 then
                -(sz/vz)
        else
                -1
```



ReFlow instruments the code 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)
        elsif (sz*vz) ≥ ε then
                -1
        else instability_warning
```

ε is a sound overestimation of the error of the expression

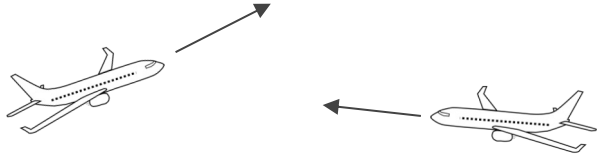


ReFlow instruments the code to detect when the floating-point control flow diverges w.r.t. the ideal one

```
tcoa(sz,vz) =
        if (sz*vz) < - ε then
                −(sz/vz)
        elsif (sz*vz) ≥ ε 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



Time to co-altitude code extraction

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

Time to co-altitude code extraction

```
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);
}
```

Time to co-altitude code extraction

```
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

Time to co-altitude code extraction

```
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

Time to co-altitude code extraction

```
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

Time to co-altitude code extraction

```
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

Time to co-altitude code extraction

```
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

ACSL contract

function call to tcoa_fp

Time to co-altitude code extraction

```
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

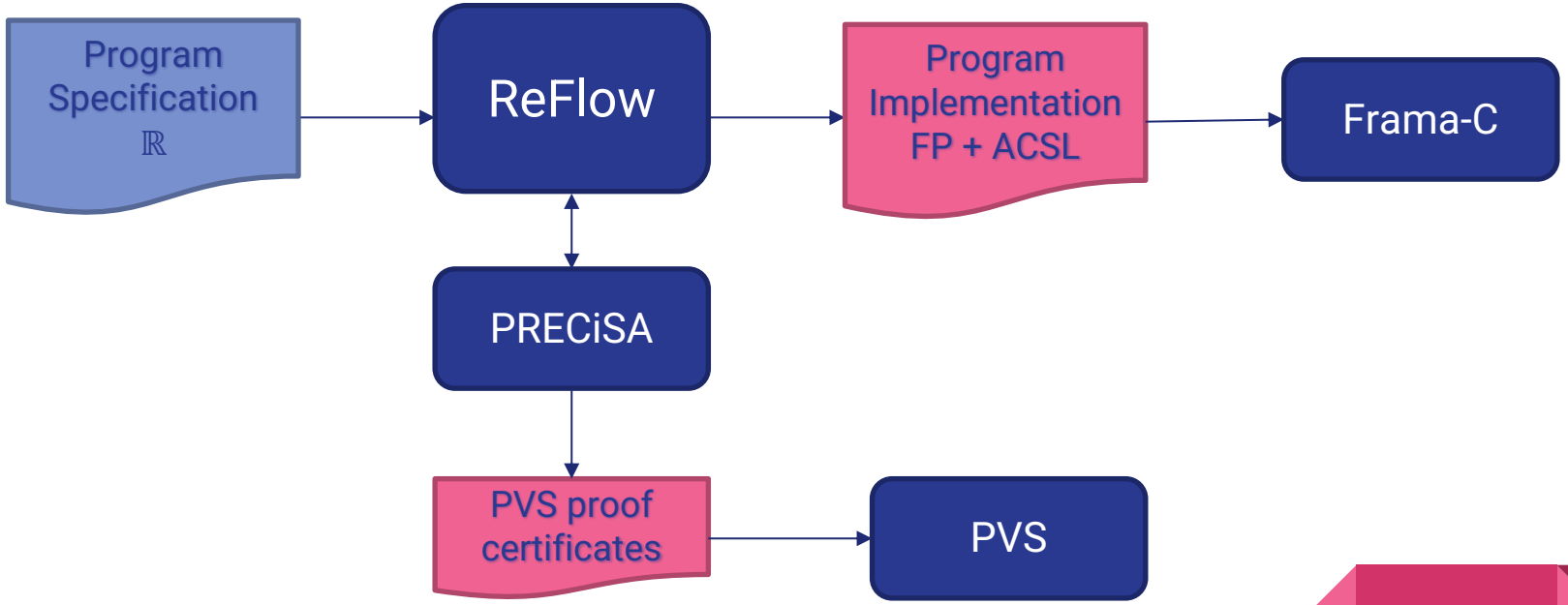
generic

ACSL contract

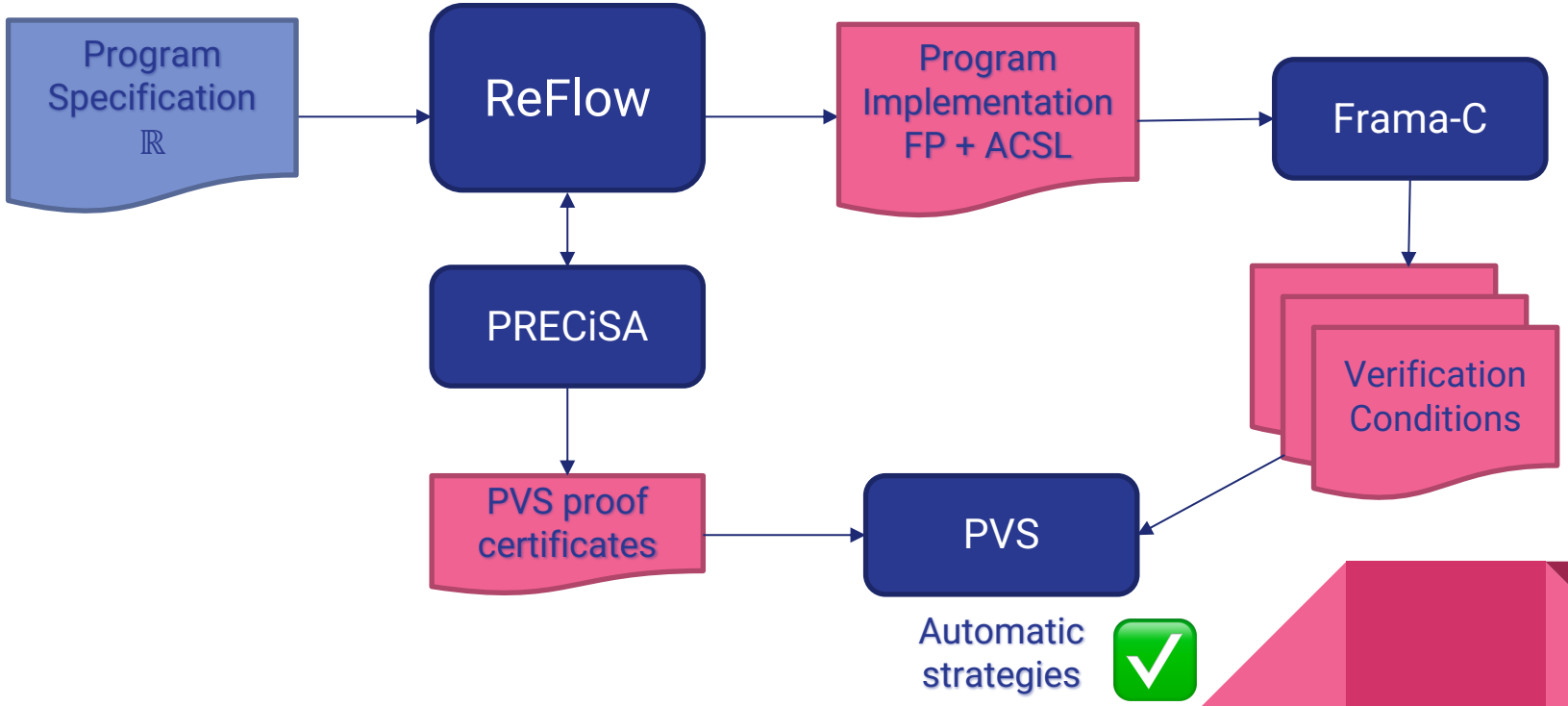
function call to tcoa_fp

concrete

The C code is automatically verified in Frama-C and PVS



The C code is automatically verified 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
    typecheck-file | evaluate-in-pvsio
5   tcoa_num_cert: THEORY
6   BEGIN
7   IMPORTING cert_tcoa, PRECiSA@bbiasp, PRECiSA@bbiadp, PRECiSA@strategies
8
9   %|- *_TCC* : PROOF
10  %|- (precisa-gen-cert-tcc)
11  %|- QED
12
13  % Floating-Point Results: 0, neg_double(div_double(sz, vz))
14  % Real Results: -((r_sz / r_vz)), 0
15  % Control Flow:  Stable
    prove | status-proofchain | show-prooflite
16  tcoa_fp_c_0 : LEMMA
17  FORALL(r_sz, r_vz: real, sz: double, vz: double):
18  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
19  AND ((((r_vz /= 0) AND ((r_sz * r_vz) < 0)) AND ((vz /= integerToDouble(0)) AND (mul_double(sz, vz) <
    integerToDouble(0)))) OR (NOT(((r_sz * r_vz) < 0)) AND NOT((mul_double(sz, vz) < integerToDouble(0)))))
20  AND r_sz ## [|1,1000|] AND  r_vz ## [|1,1000|] AND
21  finite_double?(tcoa_fp(sz, vz)) AND finite_double?(sz) AND finite_double?(vz) AND finite_double?(mul_double
    (sz, vz)) AND finite_double?(integerToDouble(0))
22  IMPLIES
23  abs(safe_prjct_double(tcoa_fp(sz, vz)) - tcoa(r_sz, r_vz))<=6876345720111303 / 247588007857076054979824844 8
24
25  %|- tcoa_fp_c_0 : PROOF
26  %|- (prove-concrete-lemma tcoa_fp_0 14 7)
27  %|- QED
28
```

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
    typecheck-file | evaluate-in-pvsio
5   tcoa_num_cert: THEORY
6   BEGIN
7   IMPORTING cert_tcoa, PRECiSA@bbiasp, PRECiSA@bbiadp, PRECiSA@strategies
8
9   %|- *_TCC* : PROOF
10  %|- (precisa-gen-cert-tcc)
11  %|- QED
12
13  % Floating-Point Results: 0, neg_double(div_double(sz, vz))
14  % Real Results: -((r_sz / r_vz)), 0
15  % Control Flow:  Stable
    prove | status-proofchain | show-prooflite
16  tcoa_fp_c_0 : LEMMA
17  FORALL(r_sz, r_vz: real, sz: double, vz: double):
18  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
19  AND ((((r_vz /= 0) AND ((r_sz * r_vz) < 0)) AND ((vz /= integerToDouble(0)) AND (mul_double(sz, vz) <
    integerToDouble(0)))) OR (NOT(((r_sz * r_vz) < 0)) AND NOT((mul_double(sz, vz) < integerToDouble(0)))))
20  AND r_sz ## [|1,1000|] AND  r_vz ## [|1,1000|] AND
21  finite_double?(tcoa_fp(sz, vz)) AND finite_double?(sz) AND finite_double?(vz) AND finite_double?(mul_double
    (sz, vz)) AND finite_double?(integerToDouble(0))
22  IMPLIES
23  abs(safe_prjct_double(tcoa_fp(sz, vz)) - tcoa(r_sz, r_vz))<=6876345720111303 / 247588007857076054979824844 8
24
25  %|- tcoa_fp_c_0 : PROOF
26  %|- (prove-concrete-lemma tcoa_fp_0 14 7)
27  %|- QED
28
```

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 > tcoa_num_cert.pvs > tcoa_num_ce

```
1   % This file is automatically generated by PRECiSA
2
3   % maxDepth: 7 , prec: 10^-14
4
    typecheck-file | evaluate-in-pvsio
5   tcoa_num_cert: THEORY
6   BEGIN
7   IMPORTING cert_tcoa, PRECiSA@bbiasp, PRECiSA@bbiadp, PRECiSA@strategies
8
9   %|- *_TCC* : PROOF
10  %|- (precisa-gen-cert-tcc)
11  %|- QED
12
13  % Floating-Point Results: 0, neg_double(div_double(sz, vz))
14  % Real Results: -((r_sz / r_vz)), 0
15  % Control Flow:  Stable
    prove | status-proofchain | show-prooflite
16  tcoa_fp_c_0 : LEMMA
17  FORALL(r_sz, r_vz: real, sz: double, vz: double):
18  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
19  AND ((((r_vz /= 0) AND ((r_sz * r_vz) < 0)) AND ((vz /= integerToDouble(0)) AND (mul_double(sz, vz) <
    integerToDouble(0)))) OR (NOT(((r_sz * r_vz) < 0)) AND NOT((mul_double(sz, vz) < integerToDouble(0)))))
20  AND r_sz ## [|1,1000|] AND  r_vz ## [|1,1000|] AND
21  finite_double?(tcoa_fp(sz, vz)) AND finite_double?(sz) AND finite_double?(vz) AND finite_double?(mul_double
    (sz, vz)) AND finite_double?(integerToDouble(0))
22  IMPLIES
23  abs(safe_prjct_double(tcoa_fp(sz, vz)) - tcoa(r_sz, r_vz))<=6876345720111303 / 247588007857076054979824844 8
24
25  %|- tcoa_fp_c_0 : PROOF
26  %|- (prove-concrete-lemma tcoa_fp_0 14 7)
27  %|- QED
28
```

Overflow detection

Rounding error

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
    typecheck-file | evaluate-in-pvsio
5   tcoa_num_cert: THEORY
6   BEGIN
7   IMPORTING cert_tcoa, PRECiSA@bbiasp, PRECiSA@bbiadp, PRECiSA@strategies
8
9   %|- *_TCC* : PROOF
10  %|- (precisa-gen-cert-tcc)
11  %|- QED
12
13  % Floating-Point Results: 0, neg_double(div_double(sz, vz))
14  % Real Results: -((r_sz / r_vz)), 0
15  % Control Flow:  Stable
    prove | status-proofchain | show-prooflite
16  tcoa_fp_c_0 : LEMMA
17  FORALL(r_sz, r_vz: real, sz: double, vz: double):
18  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
19  AND ((((r_vz /= 0) AND ((r_sz * r_vz) < 0)) AND ((vz /= integerToDouble(0)) AND (mul_double(sz, vz) <
    integerToDouble(0)))) OR (NOT(((r_sz * r_vz) < 0)) AND NOT((mul_double(sz, vz) < integerToDouble(0)))))
20  AND r_sz ## [|1,1000|] AND  r_vz ## [|1,1000|] AND
21  finite_double?(tcoa_fp(sz, vz)) AND finite_double?(sz) AND finite_double?(vz) AND finite_double?(mul_double
    (sz, vz)) AND finite_double?(integerToDouble(0))
22  IMPLIES
23  abs(safe_prjct_double(tcoa_fp(sz, vz)) - tcoa(r_sz, r_vz))<=6876345720111303 / 247588007857076054979824844 8
24
25  %|- tcoa_fp_c_0 : PROOF
26  %|- (prove-concrete-lemma tcoa_fp_0 14 7)
27  %|- QED
28
```

Overflow detection

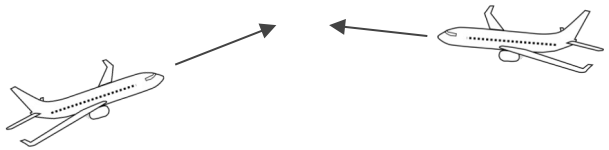
Rounding error

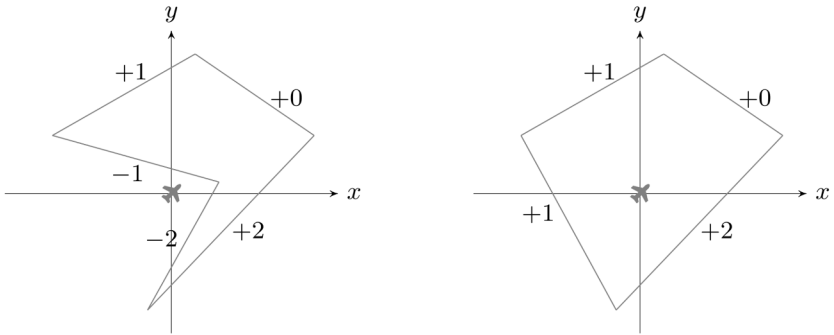
Automatic strategy

a PVS formally verified optimization algorithm is used

Applications

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





What’s next?

- ReFlow is currently under revision for NASA open-source release
- Currently limited support for a class of for loops
  ⇒ Add support for more complex loops
- Improve the precision in the instability analysis for conditional and loops
- Reduce the complexity of the ACSL annotation
- Integration with precision optimization tools (Herbie)

Thanks for your attention!

Laura Titolo

https://lauratitolo.github.io/