# 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



.
```

# Motivation

- Round-off error accumulation:

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

# Motivation

- Round-off error accumulation:

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



.
```

# Motivation

- Round-off error accumulation:

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



.
```

# Motivation

- Round-off error accumulation:

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




.
```

# Motivation

- Round-off error accumulation:

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

.
```

# Motivation

- Round-off error accumulation:

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

.
```

Divergence between ideal control flow and floating-point

# Motivation

- Round-off error accumulation:

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

.
```

Unstable guard

Divergence between ideal control flow and floating-point

# Motivation

- Round-off error accumulation:

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

.

# Motivation

- Round-off error accumulation:

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


.
```

# Motivation

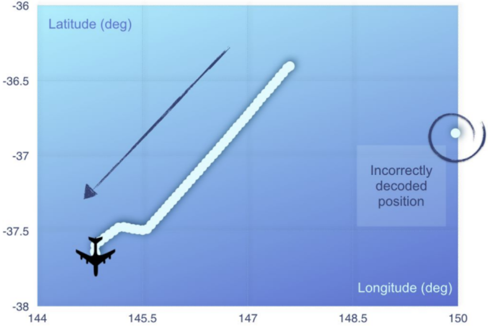
- Round-off error accumulation:

```
>>> (4/3 - 1) * 3 - 1
-2.220446049250313e-16
>>> floor((4/3 - 1) * 3 - 1)
-1
>>> 100 if floor((4/3 - 1) * 3 - 1) < 0 else 1
100
>>> (4/3 * 3 - 1 * 3) - 1
0.0
>>>
```

# 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

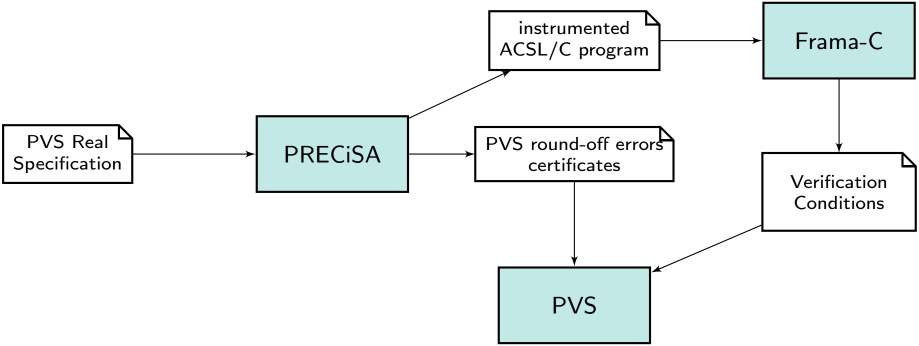
- Detect and Avoid (DAA) concept emerged as an effort to support the integration of UAVs into civil airspace
- **D**etect and **A**vo**ID** **A**lerting **L**ogic for **U**nmanned **S**ystems*
  - NASA Langley Research Center
  - Correctness and safety properties formally assured
    - PVS theorem prover
  - 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

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

# Our Approach

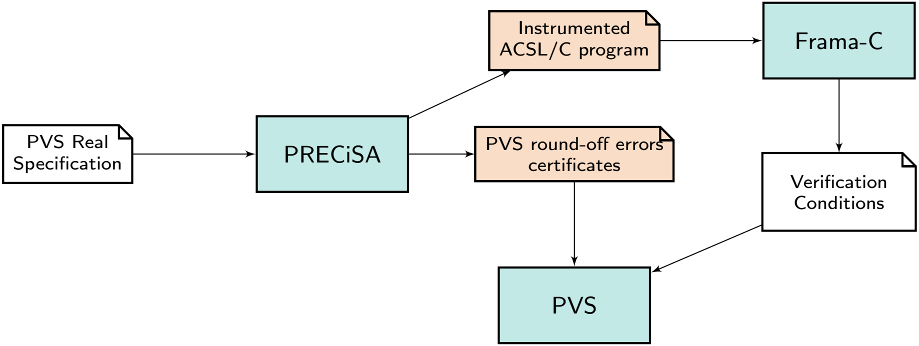
- Automatic extraction of C code from the ideal real number specification
- Features of the extracted code:
  - Instrumented to detect divergences w.r.t. the *ideal* control-flow
  - Annotated with contracts on the rounding error
  - Externally verifiable
- We use a combination of different formal-methods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

# Integrated Toolchain[†]




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

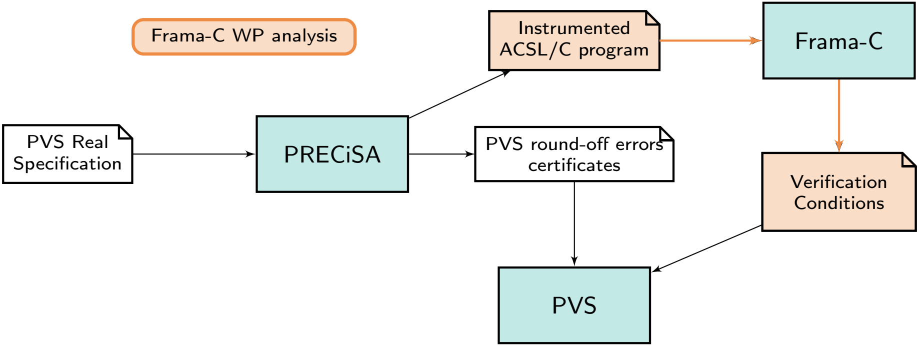
# Integrated Toolchain†




- Formal guarantees on the rounding-error
  $|\ \mathbb{R}$ specification - FP implementation $| \leq \epsilon$ computed by PRECiSA
- Control flow divergence $\Rightarrow$ warning

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

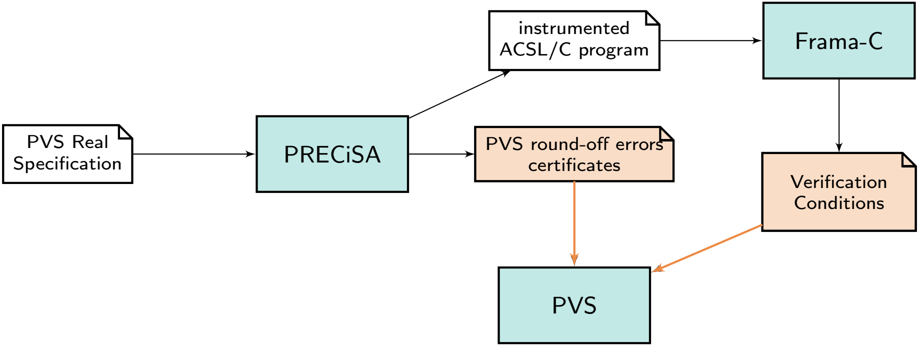
# Integrated Toolchain†




- Formal guarantees on the rounding-error
  $|\ \mathbb{R}$ specification - FP implementation $| \leq \epsilon$ computed by PRECiSA
- Control flow divergence $\Rightarrow$ warning

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

# Integrated Toolchain†




- Formal guarantees on the rounding-error
  $|\ \mathbb{R}$ 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).

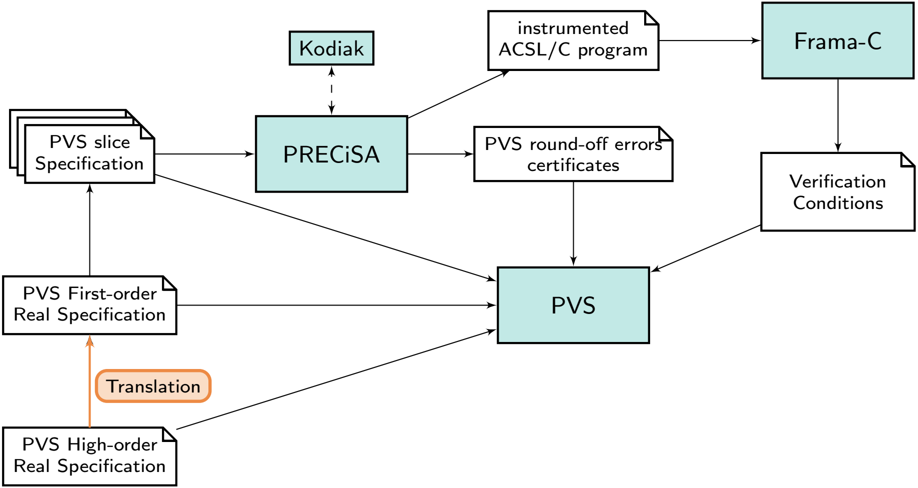
# Can This Toolchain be Applied to DAIDALUS?

- 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

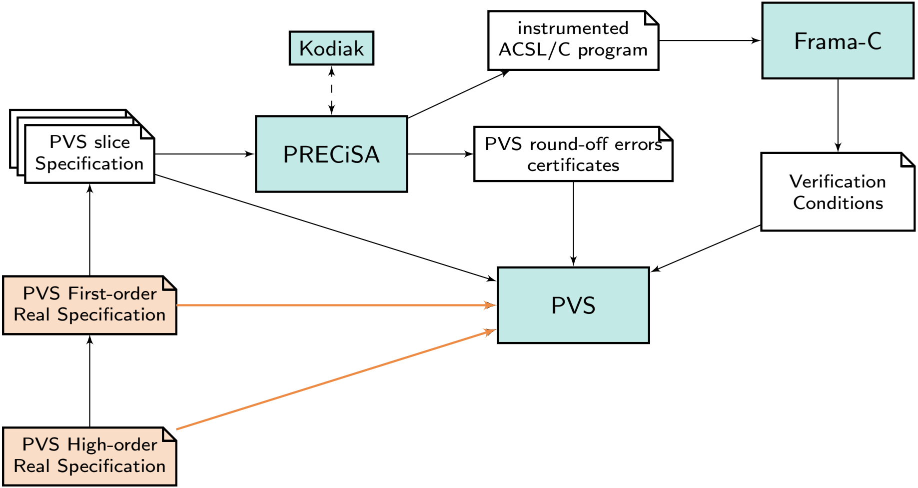
# Our Solution

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

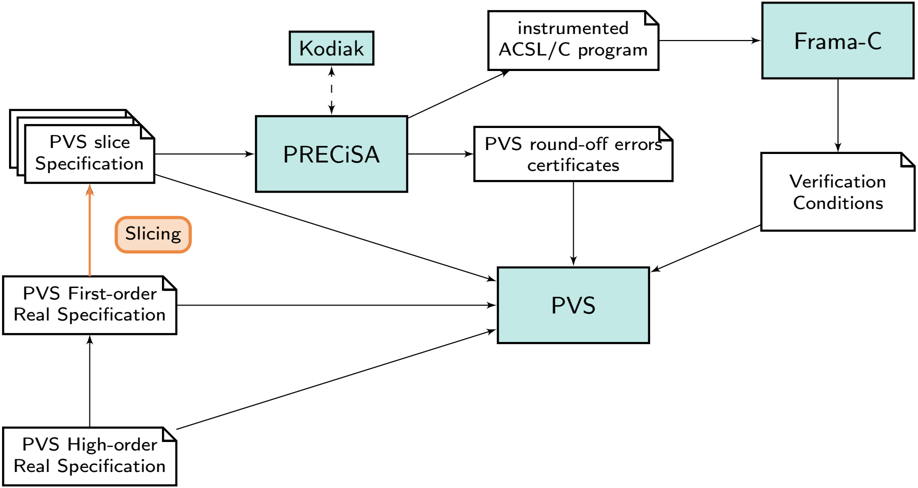
# Integrated Toolchain + Slicing



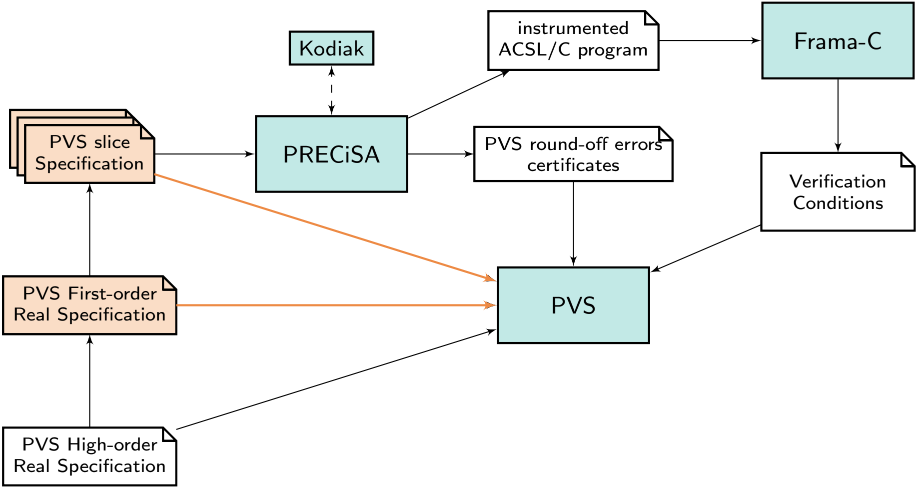
# Integrated Toolchain + Slicing



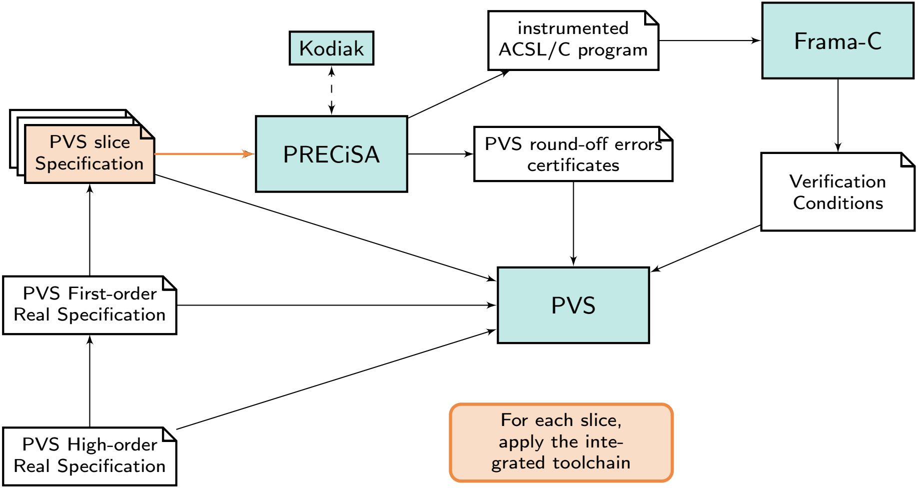
# Integrated Toolchain + Slicing



# Integrated Toolchain + Slicing

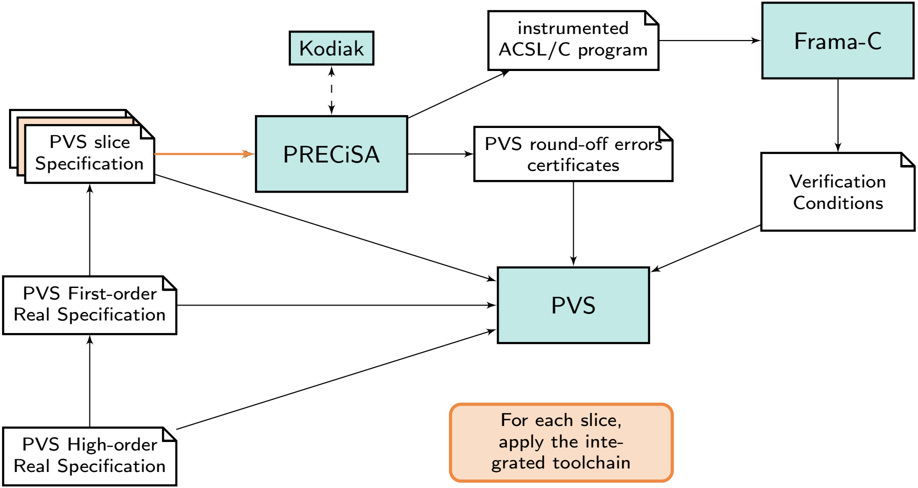


# Integrated Toolchain + Slicing



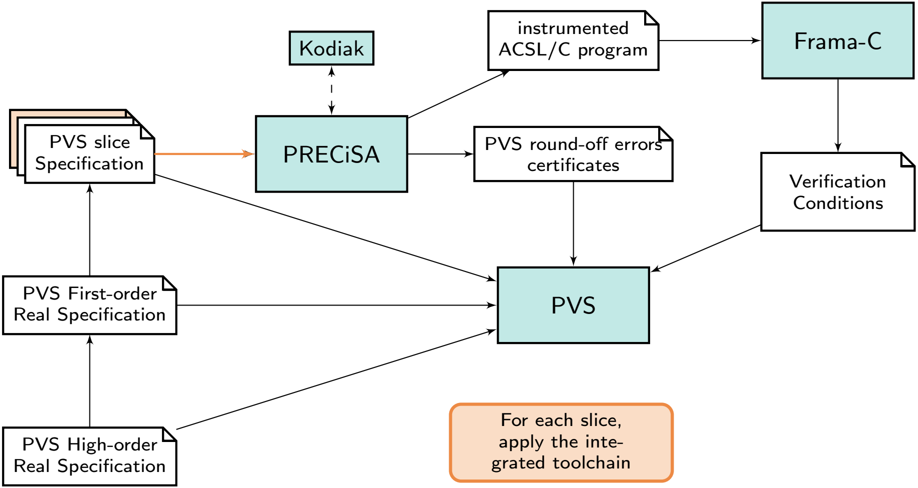

For each slice, apply the integrated toolchain

# Integrated Toolchain + Slicing




For each slice, apply the integrated toolchain

# Integrated Toolchain + Slicing



# Conditional Slicing

```
f1(x: double): double =
  if (x > 0) then
    f2(x/3)
  else
    f3(x)
  endif
```

# Conditional Slicing

```
f1(x: double): double =
  if (x > 0) then
    f2(x/3)
  else
    f3(x)
  endif
```

```
f1_gt_0(x: double | x > 0): double = f2(x/3)
```

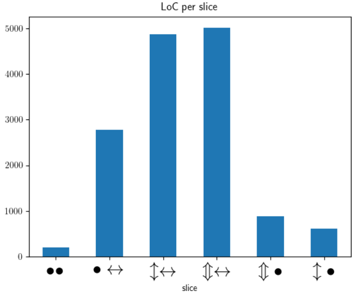
```
f1_lte_0(x: double | x <= 0): double = f3(x)
```

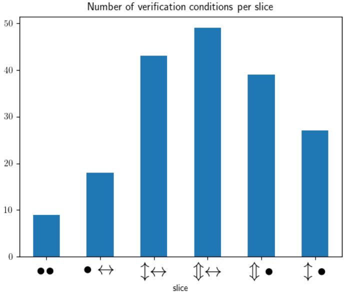
# Slicing the Well-Clear Specification

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

- $\bullet$ $\bullet$ aircraft horizontal and vertical separation unmodified;
- $\Updownarrow$ $\bullet$ aircraft vertical separation increases, horizontal separation unmodified;
- $\updownarrow$ $\bullet$ aircraft vertical separation decreases, horizontal separation unmodified;
- $\bullet$ $\leftrightarrow$ aircraft alter only horizontal separation;
- $\Updownarrow$ $\leftrightarrow$ aircraft increase vertical and alter horizontal separation;
- $\updownarrow$ $\leftrightarrow$ aircraft decrease vertical and alter horizontal separation.

# Slicing the Well-Clear Specification



# Case Study: DAIDALUS

- **T**ime of **c**losest **p**oint of **a**pproach*

```
tcpa(sx, vx, sy, vy) =
    if (vx^2 + vy^2 ≠ 0) then -(sx*vx + sy*vy)/(vx^2 + vy^2)
    else 0
```

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

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

# Case Study: DAIDALUS

- PRECiSA output:
  *symbolic* function

$$
\begin{array}{l}
\texttt{/*@}\\
\texttt{real tcpa(real } s_x\texttt{, real } v_x\texttt{, real } s_y\texttt{, real } v_y\texttt{) =}\\
{v_x}^2 \texttt{ + } {v_y}^2 \texttt{ > 0 ? -(}s_x \texttt{ * } v_x \texttt{ + } s_y \texttt{ * } v_y\texttt{) / (}{v_x}^2 \texttt{ + } {v_y}^2\texttt{) : 0;}\\
\\
\texttt{double } \boxed{\texttt{tcpa}}\texttt{(double } \boxed{s_x}\texttt{, double } \boxed{v_x}\texttt{, double } \boxed{s_y}\texttt{, double } \boxed{v_y}\texttt{) =}\\
\boxed{v_x^2 + v_y^2 > 0}\texttt{?}\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}\texttt{:}\boxed{0}\texttt{;}\\
\\
\texttt{requires: } \boxed{0 <= \epsilon} \wedge \texttt{ finite?(}\boxed{\epsilon}\texttt{) ;}\\
\texttt{ensures: result } \neq \omega \wedge \texttt{ (|(}\boxed{v_x^2 + v_y^2}\texttt{)-(}{v_x}^2 \texttt{ + } {v_y}^2\texttt{)| } \leq \epsilon \implies \texttt{ result = } \boxed{\texttt{tcpa}}\texttt{)}\\
\texttt{*/}\\
\texttt{double' tcpa\_fp(double } \boxed{s_x}\texttt{, double } \boxed{v_x}\texttt{, double } \boxed{s_y}\texttt{, double } \boxed{v_y}\texttt{, double } \epsilon\texttt{)\{}\\
\quad \texttt{if(}\boxed{v_x^2 + v_y^2 > \epsilon}\texttt{)\{}\\
\qquad \texttt{return } \boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}\texttt{;}\\
\quad \texttt{\}}\\
\quad \texttt{else\{}\\
\qquad \texttt{if(}\boxed{v_x^2 + v_y^2 \leq -\epsilon}\texttt{)}\\
\qquad\quad \texttt{return } \boxed{0}\texttt{;}\\
\qquad \texttt{else}\\
\qquad\quad \texttt{return } \omega\texttt{;}\\
\quad \texttt{\}}\\
\texttt{\}}
\end{array}
$$

# Case Study: DAIDALUS

- PRECiSA output: *symbolic* function

```
/*@
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 $\boxed{\mathtt{tcpa}}$(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$) =
$\boxed{v_x^2 + v_y^2 > 0}$ ? $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$ : $\boxed{0}$;

requires: $\boxed{0 <= \epsilon} \wedge$ finite?($\boxed{\epsilon}$) ;
ensures: result $\neq \omega \wedge (|(\boxed{v_x^2 + v_y^2}) - (v_x{}^2 + v_y{}^2)| \leq \epsilon \implies$ result = $\boxed{\mathtt{tcpa}}$)
*/

double' tcpa_fp(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$, double $\epsilon$){
    if($\boxed{v_x^2 + v_y^2 > \epsilon}$){
        return $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$;
    }
    else{
        if($\boxed{v_x^2 + v_y^2 \leq -\epsilon}$)
            return $\boxed{0}$;
        else
            return $\omega$;
    }
}

# Case Study: DAIDALUS

- PRECiSA output:
  *symbolic* function

```
/*@
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];

requires: [0 <= ε] ∧ finite?([ε]) ;
ensures: result ≠ ω ∧ (|([v_x^2 + v_y^2])-(v_x^2 + v_y^2)| ≤ ε  ⟹  result = [tcpa])
*/
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;
    }
}
```

# Case Study: DAIDALUS

- PRECiSA output: *symbolic* function

```
/*@
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 $\boxed{\text{tcpa}}$(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$) =
$\boxed{v_x^2 + v_y^2 > 0}$ ? $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$ : $\boxed{0}$;

requires: $\boxed{0 <= \epsilon}$ $\wedge$ finite?($\boxed{\epsilon}$) ;
ensures: result $\neq \omega$ $\wedge$ ($|(\boxed{v_x^2 + v_y^2}) - (v_x^2 + v_y^2)| \leq \epsilon \implies$ result = $\boxed{\text{tcpa}}$)
*/
double' tcpa_fp(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$, double $\epsilon$){
    if($\boxed{v_x^2 + v_y^2 > \epsilon}$){
        return $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$;
    }
    else{
        if($\boxed{v_x^2 + v_y^2 \leq -\epsilon}$)
            return $\boxed{0}$;
        else
            return $\omega$;
    }
}

# Case Study: DAIDALUS

- PRECiSA output: *symbolic* function

```
/*@
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 $\boxed{\texttt{tcpa}}$(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$) =
$\boxed{v_x^2 + v_y^2 > 0}$ ? $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$ : $\boxed{0}$;

requires: $\boxed{0 <= \epsilon}$ $\wedge$ finite?($\boxed{\epsilon}$) ;

ensures: result $\neq \omega$ $\wedge$ (|($\boxed{v_x^2 + v_y^2}$)-($v_x^2$ + $v_y^2$)| $\leq \epsilon$ $\Longrightarrow$ result = $\boxed{\texttt{tcpa}}$)

*/

double' tcpa_fp(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$, double $\epsilon$){
    if($\boxed{v_x^2 + v_y^2 > \epsilon}$){
        return $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$;
    }
    else{
        if($\boxed{v_x^2 + v_y^2 \leq -\epsilon}$)
            return $\boxed{0}$;
        else
            return $\omega$;
    }
}

# Case Study: DAIDALUS

- PRECiSA output: *symbolic* function

```
/*@
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 $\boxed{\texttt{tcpa}}$(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$) =
$\boxed{v_x^2 + v_y^2 > 0}$ ? $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$ : $\boxed{0}$;

requires: $\boxed{0 <= \epsilon} \wedge$ finite?($\boxed{\epsilon}$) ;
ensures: result $\neq \omega \wedge (|(\boxed{v_x^2 + v_y^2}) - (v_x^2 + v_y^2)| \leq \epsilon \implies$ result = $\boxed{\texttt{tcpa}}$)
*/

double' tcpa_fp(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$, double $\epsilon$){
    if($\boxed{v_x^2 + v_y^2 > \epsilon}$){
        return $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$;
    }
    else{
        if($\boxed{v_x^2 + v_y^2 \leq -\epsilon}$)
            return $\boxed{0}$;
        else
            return $\omega$;
    }
}

# Case Study: DAIDALUS

- PRECiSA output: *symbolic* function

```
/*@
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 $\boxed{\mathtt{tcpa}}$(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$) =
$\boxed{v_x^2 + v_y^2 > 0}$?$\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$:$\boxed{0}$;

requires: $\boxed{0 <= \epsilon} \wedge$ finite?($\boxed{\epsilon}$) ;
ensures: result $\neq \omega \wedge$ (|($\boxed{v_x^2 + v_y^2}$)-($v_x^2$ + $v_y^2$)| $\leq \epsilon \implies$ result = $\boxed{\mathtt{tcpa}}$)
*/

double' tcpa_fp(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$, double $\epsilon$){
    if($\boxed{v_x^2 + v_y^2 > \epsilon}$){
        return $\boxed{-(s_x * v_x + s_y * v_y)/(v_x^2 + v_y^2)}$;
    }
    else{
        if($\boxed{v_x^2 + v_y^2 \leq -\epsilon}$)
            return $\boxed{0}$;
        else
            return $\omega$;
    }
}

# Case Study: DAIDALUS

- PRECiSA output:
  *symbolic* function

```
/*@
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];

requires: [0 <= ε] ∧ finite?([ε]) ;
ensures: result ≠ ω ∧ (|([v_x^2 + v_y^2])-(v_x^2 + v_y^2)| ≤ ε  ⟹  result = [tcpa])
*/
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;
    }
}
```

# Case Study: DAIDALUS

- PRECiSA output: *symbolic* function

```
/*@
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];

requires: [0 <= ε] ∧ finite?([ε]) ;
ensures: result ≠ ω ∧ (|([v_x^2 + v_y^2])-(v_x^2 + v_y^2)| ≤ ε ⟹ result = [tcpa])
*/
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;
    }
}
```

# Case Study: DAIDALUS

- PRECiSA output:
  *symbolic* function

```
/*@
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];

requires: [0 <= ε] ∧ finite?([ε]) ;
ensures: result ≠ ω ∧ (|([v_x^2 + v_y^2])-(v_x^2 + v_y^2)| ≤ ε ⟹ result = [tcpa])
*/
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;
    }
}
```

# Case Study: DAIDALUS

- PRECiSA output: *numeric* function

/*@

ensures $\forall$ **real** $s_x, s_y, v_x, v_y$; $\forall$ **double** $\boxed{s_x}, \boxed{s_y}, \boxed{v_x}, \boxed{v_y}$;

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

$|\boxed{s_x} - s_x| \leq ulp(s_x)/2 \wedge |\boxed{s_y} - s_y| \leq ulp(s_y)/2 \wedge$

$|\boxed{v_x} - v_x| \leq ulp(s_x)/2 \wedge |\boxed{v_y} - v_y| \leq ulp(v_y)/2$

$\Longrightarrow$ |(result - tcpa($s_x$, $v_x$, $s_y$, $v_y$)| $\leq 9.106570e-07$)| ;

*/

double' tcpa_num(double $\boxed{s_x}$, double $\boxed{v_x}$, double $\boxed{s_y}$, double $\boxed{v_y}$) {

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

}

# Case Study: DAIDALUS

- PRECiSA output: *numeric* function

$$
\begin{array}{l}
\texttt{/*@} \\
\texttt{ensures}\ \forall\ \textbf{real}\ s_x, s_y, v_x, v_y;\ \forall\ \textbf{double}\ \boxed{s_x}, \boxed{s_y}, \boxed{v_x}, \boxed{v_y}; \\
\quad 1 < v_x < 1000 \wedge 1 < v_y < 1000 \wedge 1 < s_x < 1000 \wedge 1 < s_y < 1000 \wedge \\
\quad |\boxed{s_x} - s_x| \leq ulp(s_x)/2 \wedge |\boxed{s_y} - s_y| \leq ulp(s_y)/2 \wedge \\
\quad |\boxed{v_x} - v_x| \leq ulp(s_x)/2 \wedge |\boxed{v_y} - v_y| \leq ulp(v_y)/2 \\
\quad \Longrightarrow\ \texttt{|(result - tcpa(}s_x\texttt{, }v_x\texttt{, }s_y\texttt{, }v_y\texttt{)|} \leq 9.106570e-07\texttt{)| ;} \\
\texttt{*/} \\
\texttt{double' tcpa\_num(double } \boxed{s_x}\texttt{, double } \boxed{v_x}\texttt{, double } \boxed{s_y}\texttt{, double } \boxed{v_y}\texttt{) \{} \\
\quad \texttt{return tcpa\_fp (}sx\texttt{, }vx\texttt{, }sy\texttt{, }vy\texttt{, 4.602043e-10);} \\
\texttt{\}}
\end{array}
$$

# Case Study: DAIDALUS

- PRECiSA output: *numeric* function

$$
\begin{array}{l}
\texttt{/*@} \\
\texttt{ensures } \forall \textbf{ real } s_x, s_y, v_x, v_y; \forall \textbf{ double } \boxed{s_x}, \boxed{s_y}, \boxed{v_x}, \boxed{v_y}; \\
\quad 1 < v_x < 1000 \land 1 < v_y < 1000 \land 1 < s_x < 1000 \land 1 < s_y < 1000 \land \\
\quad \left|\boxed{s_x} - s_x\right| \leq ulp(s_x)/2 \land \left|\boxed{s_y} - s_y\right| \leq ulp(s_y)/2 \land \\
\quad \left|\boxed{v_x} - v_x\right| \leq ulp(s_x)/2 \land \left|\boxed{v_y} - v_y\right| \leq ulp(v_y)/2 \\
\quad \Longrightarrow \texttt{|(result - tcpa(}s_x\texttt{, } v_x\texttt{, } s_y\texttt{, } v_y\texttt{)|} \leq 9.106570e-07\texttt{)| ;} \\
\texttt{*/} \\
\texttt{double' tcpa\_num(double } \boxed{s_x}\texttt{, double } \boxed{v_x}\texttt{, double } \boxed{s_y}\texttt{, double } \boxed{v_y}\texttt{) \{} \\
\quad \texttt{return tcpa\_fp (}sx\texttt{, } vx\texttt{, } sy\texttt{, } vy\texttt{, 4.602043e-10);} \\
\texttt{\}}
\end{array}
$$

# Case Study: DAIDALUS

- PRECiSA output: *numeric* function

/*@

$$\texttt{ensures}\ \forall\ \textbf{real}\ s_x, s_y, v_x, v_y;\ \forall\ \textbf{double}\ \boxed{s_x}, \boxed{s_y}, \boxed{v_x}, \boxed{v_y};$$

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

$$\left|\boxed{s_x} - s_x\right| \leq ulp\left(s_x\right)/2 \wedge \left|\boxed{s_y} - s_y\right| \leq ulp\left(s_y\right)/2 \wedge$$

$$\left|\boxed{v_x} - v_x\right| \leq ulp\left(s_x\right)/2 \wedge \left|\boxed{v_y} - v_y\right| \leq ulp\left(v_y\right)/2$$

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

*/

$\texttt{double' tcpa\_num(double}\ \boxed{s_x}\texttt{, double}\ \boxed{v_x}\texttt{, double}\ \boxed{s_y}\texttt{, double}\ \boxed{v_y}\texttt{) \{}$

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

# Case Study: DAIDALUS

- PRECiSA output: *numeric* function

$$
\begin{array}{l}
\texttt{/*@} \\
\texttt{ensures}\ \forall\ \textbf{real}\ s_x, s_y, v_x, v_y; \forall\ \textbf{double}\ \boxed{s_x}, \boxed{s_y}, \boxed{v_x}, \boxed{v_y}; \\
\quad 1 < v_x < 1000 \wedge 1 < v_y < 1000 \wedge 1 < s_x < 1000 \wedge 1 < s_y < 1000 \wedge \\
\quad \left|\boxed{s_x} - s_x\right| \leq ulp(s_x)/2 \wedge \left|\boxed{s_y} - s_y\right| \leq ulp(s_y)/2 \wedge \\
\quad \left|\boxed{v_x} - v_x\right| \leq ulp(s_x)/2 \wedge \left|\boxed{v_y} - v_y\right| \leq ulp(v_y)/2 \\
\quad \Longrightarrow\ \texttt{|(result - tcpa(}s_x\texttt{, }v_x\texttt{, }s_y\texttt{, }v_y\texttt{)|} \leq 9.106570e-07\texttt{)|}\ \texttt{;} \\
\texttt{*/} \\
\texttt{double' tcpa\_num(double}\ \boxed{s_x}\texttt{, double}\ \boxed{v_x}\texttt{, double}\ \boxed{s_y}\texttt{, double}\ \boxed{v_y}\texttt{) \{} \\
\quad \texttt{return tcpa\_fp (}sx\texttt{, }vx\texttt{, }sy\texttt{, }vy\texttt{, 4.602043e-10);} \\
\texttt{\}}
\end{array}
$$

# Case Study: DAIDALUS

- PRECiSA output: Predicate

```
/*@
predicate WCV_interval⇕•(real b, t, s_x, s_y, s_z, v_x, v_y, v_z) = ....
predicate WCV_interval⇕•_fp(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z]) = ....;

ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => WCV_interval⇕•(b, t, s_x, s_y, s_z, v_x, v_y, v_z) ∧
       WCV_interval⇕•_fp([b, t, s_x, s_y, s_z, v_x, v_y, v_z])
*/
bool' WCV_interval⇕•_plus(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]){...}

/*@

ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => ¬ WCV_interval⇕•(b, t, s_x, s_y, s_z, v_x, v_y, v_z) ∧
       ¬ WCV_interval⇕•_fp([b, t, s_x, s_y, s_z, v_x, v_y, v_z])
*/
bool' WCV_interval⇕•_mns(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]){...}
```

# Case Study: DAIDALUS

- PRECiSA output: Predicate

/*@

predicate WCV_interval$_{\Updownarrow\bullet}$(real $b, t, s_x, s_y, s_z, v_x, v_y, v_z$) = ....

predicate WCV_interval$_{\Updownarrow\bullet}$_fp(double $\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z}$) = ....;

ensures: $\forall$ real $b, t, s_x, s_y, s_z, v_x, v_y, v_z$;

    \result $\neq \omega$ $\wedge \cdots$ // bounds on errors

    $\wedge$ \result

    => WCV_interval$_{\Updownarrow\bullet}$($b, t, s_x, s_y, s_z, v_x, v_y, v_z$) $\wedge$

       WCV_interval$_{\Updownarrow\bullet}$_fp($\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z}$)

*/

bool' WCV_interval$_{\Updownarrow\bullet}$_plus(double $\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3}$){...}

/*@

ensures: $\forall$ real $b, t, s_x, s_y, s_z, v_x, v_y, v_z$;

    \result $\neq \omega$ $\wedge \cdots$ // bounds on errors

    $\wedge$ \result

    => $\neg$ WCV_interval$_{\Updownarrow\bullet}$($b, t, s_x, s_y, s_z, v_x, v_y, v_z$) $\wedge$

       $\neg$ WCV_interval$_{\Updownarrow\bullet}$_fp($\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z}$)

*/

bool' WCV_interval$_{\Updownarrow\bullet}$_mns(double $\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3}$){...}

# Case Study: DAIDALUS

- PRECiSA output: Predicate

```
/*@
predicate WCV_interval⇕•(real b,t,s_x,s_y,s_z,v_x,v_y,v_z) = ....
predicate WCV_interval⇕•_fp(double [b,t,s_x,s_y,s_z,v_x,v_y,v_z]) = ....;

ensures: ∀ real b,t,s_x,s_y,s_z,v_x,v_y,v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => WCV_interval⇕•(b,t,s_x,s_y,s_z,v_x,v_y,v_z) ∧
       WCV_interval⇕•_fp([b,t,s_x,s_y,s_z,v_x,v_y,v_z])
*/
bool' WCV_interval⇕•_plus(double [b,t,s_x,s_y,s_z,v_x,v_y,v_z,e_1,e_2,e_3]){...}

/*@

ensures: ∀ real b,t,s_x,s_y,s_z,v_x,v_y,v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => ¬ WCV_interval⇕•(b,t,s_x,s_y,s_z,v_x,v_y,v_z) ∧
       ¬ WCV_interval⇕•_fp([b,t,s_x,s_y,s_z,v_x,v_y,v_z])
*/
bool' WCV_interval⇕•_mns(double [b,t,s_x,s_y,s_z,v_x,v_y,v_z,e_1,e_2,e_3]){...}
```

# Case Study: DAIDALUS

- PRECiSA output: Predicate

```
/*@
predicate WCV_interval⇕•(real b, t, s_x, s_y, s_z, v_x, v_y, v_z) = ....
predicate WCV_interval⇕•_fp(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z]) = ....;

ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => WCV_interval⇕•(b, t, s_x, s_y, s_z, v_x, v_y, v_z) ∧
       WCV_interval⇕•_fp([b, t, s_x, s_y, s_z, v_x, v_y, v_z])
*/
bool' WCV_interval⇕•_plus(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]){...}

/*@

ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => ¬ WCV_interval⇕•(b, t, s_x, s_y, s_z, v_x, v_y, v_z) ∧
       ¬ WCV_interval⇕•_fp([b, t, s_x, s_y, s_z, v_x, v_y, v_z])
*/
bool' WCV_interval⇕•_mns(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]){...}
```

# 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  ∧ ··· // bounds on errors
    ∧ \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  ∧ ··· // bounds on errors
    ∧ \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

- 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  ∧ ··· // bounds on errors
    ∧ \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  ∧ ··· // bounds on errors
    ∧ \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

- PRECiSA output: Predicate

```
/*@
predicate WCV_interval⇕•(real b, t, s_x, s_y, s_z, v_x, v_y, v_z) = ....
predicate WCV_interval⇕•_fp(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z]) = ....;

ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => WCV_interval⇕•(b, t, s_x, s_y, s_z, v_x, v_y, v_z) ∧
       WCV_interval⇕•_fp([b, t, s_x, s_y, s_z, v_x, v_y, v_z])
*/
bool' WCV_interval⇕•_plus(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]){...}

/*@

ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => ¬ WCV_interval⇕•(b, t, s_x, s_y, s_z, v_x, v_y, v_z) ∧
       ¬ WCV_interval⇕•_fp([b, t, s_x, s_y, s_z, v_x, v_y, v_z])
*/
bool' WCV_interval⇕•_mns(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]){...}
```

# Case Study: DAIDALUS

- PRECiSA output: Predicate

```
/*@
predicate WCV_interval⇕•(real b, t, s_x, s_y, s_z, v_x, v_y, v_z) = ....
predicate WCV_interval⇕•_fp(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z]) = ....;

ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => WCV_interval⇕•(b, t, s_x, s_y, s_z, v_x, v_y, v_z) ∧
       WCV_interval⇕•_fp([b, t, s_x, s_y, s_z, v_x, v_y, v_z])
*/
bool' WCV_interval⇕•_plus(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]){...}

/*@

ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
    \result ≠ ω  ∧ ··· // bounds on errors
    ∧ \result
    => ¬ WCV_interval⇕•(b, t, s_x, s_y, s_z, v_x, v_y, v_z) ∧
       ¬ WCV_interval⇕•_fp([b, t, s_x, s_y, s_z, v_x, v_y, v_z])
*/
bool' WCV_interval⇕•_mns(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]){...}
```

# 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), (vx,vy,vz), (sx,sy,sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx,vy,vz), (sx,sy,sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx,vy,vz), (sx,sy,sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b,t,s_x,s_y,s_z,v_x,v_y,v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z,v_z)] − (t − coalt_t_asc_fp(s_z,v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b,t,s_x,s_y,s_z,v_x,v_y,v_z))
*/
bool' WCV_interval(double [b,t,s_x,s_y,s_z,v_x,v_y,v_z,e_1,e_2,e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b,t,s_x,s_y,s_z,v_x,v_y,v_z,e_1,e_2,e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b,t,s_x,s_y,s_z,v_x,v_y,v_z,e_1,e_2,e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx, vy, vz), (sx, sy, sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int_{⇕•}_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int_{⇕•}_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx, vy, vz), (sx, sy, sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z))] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx, vy, vz), (sx, sy, sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx, vy, vz), (sx, sy, sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3 ...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{ // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx,vy,vz), (sx,sy,sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx,vy,vz), (sx,sy,sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx, vy, vz), (sx, sy, sz)) previously defined

requires: \is_finite(e_0) ∧ e_0 > 0 ∧ ... ∧\is_finite(e_n) ∧ e_n > 0;
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |δ − v_z ∗ δ_tcoa − δ − v_z ∗ δ_tcoa| < e_0 ∧
  |(t − coalt_t_asc_fp(s_z, v_z)) − (t − coalt_t_asc_fp(s_z, v_z)| < e_1 ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...){
  bool' res;
  if (v_z > 0){ // increasing vertical separation
    if (v_x == 0.0 ∧ v_y == 0.0){ // maintaining horizontal separation
      res = WCV_int⇕•_plus(b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus(b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if (v_z < 0){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx,vy,vz), (sx,sy,sz)) previously defined

requires: \is_finite(e_0) ∧ e_0 > 0 ∧ ... ∧\is_finite(e_n) ∧ e_n > 0;
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |δ − v_z ∗ δ_tcoa − δ − v_z ∗ δ_tcoa| < e_0 ∧
  |(t − coalt_t_asc_fp(s_z, v_z)) − (t − coalt_t_asc_fp(s_z, v_z)| < e_1 ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...){
  bool' res;
  if (v_z > 0){ // increasing vertical separation
    if (v_x == 0.0 ∧ v_y == 0.0){ // maintaining horizontal separation
      res = WCV_int⇕•_plus(b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus(b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{ // altering horizontal separation
      ...
    }
  } else if (v_z < 0){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx, vy, vz), (sx, sy, sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z))] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{ // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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?($(b, t)$, $(vx, vy, vz)$, $(sx, sy, sz)$) previously defined

requires: \is_finite($\boxed{e_0}$) $\wedge$ $\boxed{e_0 > 0}$ $\wedge \ldots \wedge$\is_finite($\boxed{e_n}$) $\wedge$ $\boxed{e_n > 0}$;
ensures: $\forall$ real $b, t, s_x, s_y, s_z, v_x, v_y, v_z$;
  $\left| \boxed{\delta - v_z * \delta_{tcoa}} - \delta - v_z * \delta_{tcoa} \right| < \boxed{e_0} \wedge$
  $\left| \boxed{(t - \texttt{coalt\_t\_asc\_fp}(s_z, v_z)} - (t - \texttt{coalt\_t\_asc\_fp}(s_z, v_z) \right| < \boxed{e_1} \wedge$
  ...
  \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 $\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3 \ldots}$){
  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$_{\Updownarrow\bullet}$_plus($\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3}$);
      if (res $== \omega \vee$ res) return res;
      res = WCV_int$_{\Updownarrow\bullet}$_minus($\boxed{b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3}$);
      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

- Top-Layer

```
/*@
predicate wcv_in_range(real b, t, s_x, s_y, s_z, v_x, v_y, v_z) =
  // WCV?((b, t), (vx, vy, vz), (sx, sy, sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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), (vx, vy, vz), (sx, sy, sz)) previously defined

requires: \is_finite([e_0]) ∧ [e_0 > 0] ∧ ... ∧\is_finite([e_n]) ∧[e_n > 0];
ensures: ∀ real b, t, s_x, s_y, s_z, v_x, v_y, v_z;
  |[δ − v_z ∗ δ_tcoa] − δ − v_z ∗ δ_tcoa| < [e_0] ∧
  |[(t − coalt_t_asc_fp(s_z, v_z)] − (t − coalt_t_asc_fp(s_z, v_z)| < [e_1] ∧
  ...
  \result ≠ ω
   ⟹ (\result ⟺ wcv_in_range(b, t, s_x, s_y, s_z, v_x, v_y, v_z))
*/
bool' WCV_interval(double [b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3...]){
  bool' res;
  if ([v_z > 0]){ // increasing vertical separation
    if ([v_x == 0.0] ∧ [v_y == 0.0]){ // maintaining horizontal separation
      res = WCV_int⇕•_plus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω ∨ res) return res;
      res = WCV_int⇕•_minus([b, t, s_x, s_y, s_z, v_x, v_y, v_z, e_1, e_2, e_3]);
      if (res == ω) return ω;
      if (res) return false;
      return ω;
    } else{  // altering horizontal separation
      ...
    }
  } else if ([v_z < 0]){ // decreasing vertical separation
    ...
  } else { // maintaining vertical separation
   ...
  }
}
```

# 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, [ε, s_x, s_y, v_x, v_y] :double, result:double'):
    result ≠ ω =>
    1 < v_x < 1000 => 1 < v_y < 1000 => 1 < s_x < 1000 => 1 < s_y < 1000 =>
    |[s_x] − s_x| ≤ ulp(s_x)/2 => |[s_y] − s_y| ≤ ulp(s_y)/2 =>
    |[v_x] − v_x| ≤ ulp(s_x)/2 => |[v_y] − v_y| ≤ ulp(v_y)/2
    [result = l_tcpa_fp(s_x, s_y, v_x, v_y)] =>
     (FORALL (v_x, v_y:real):
        |[v_x * v_x + v_y * v_y] − (v_x^2 + v_y^2)| ≤
        (8901646138474497 / 19342813113834066795298816)) =>
           p_tcpa_stable_paths(v_x, v_y, [v_x, v_y])) =>
    |result - tcpa(s_x, v_x, s_y, v_y)| <=
    (4300455909721841 / 4722366482869645213696
```

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

$\Longrightarrow$ |(result - 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
  - Update previous PVS strategies to automate VCs proofs

# Thank you

Download PRECiSA

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

- PRECiSA execution time

| File | time (mm:ss) |
|---|---|
| First-order version | Time-Out ($>$ 1d) |
| $\bullet \ \bullet$ | 00:00.00 |
| $\Updownarrow \ \bullet$ | 00:00.14 |
| $\updownarrow \ \bullet$ | 00:00.18 |
| $\bullet \ \leftrightarrow$ | 00:08.26 |
| $\Updownarrow \ \leftrightarrow$ | 05:20.46 |
| $\updownarrow \ \leftrightarrow$ | 07:10.11 |

- Quantitative details

| Step | Specification lines | Proof lines | Proof commands |
|---|---|---|---|
| First-order version | 649 | 2529 | 880 |
| Slicing | 1083 | 7976 | 2778 |

# 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