Compact Position Reporting Algorithm

A verified floating-point implementation in C

Mariano M. Moscato

mariano.moscato@nianet.org

The Algorithm

The ADS-B System

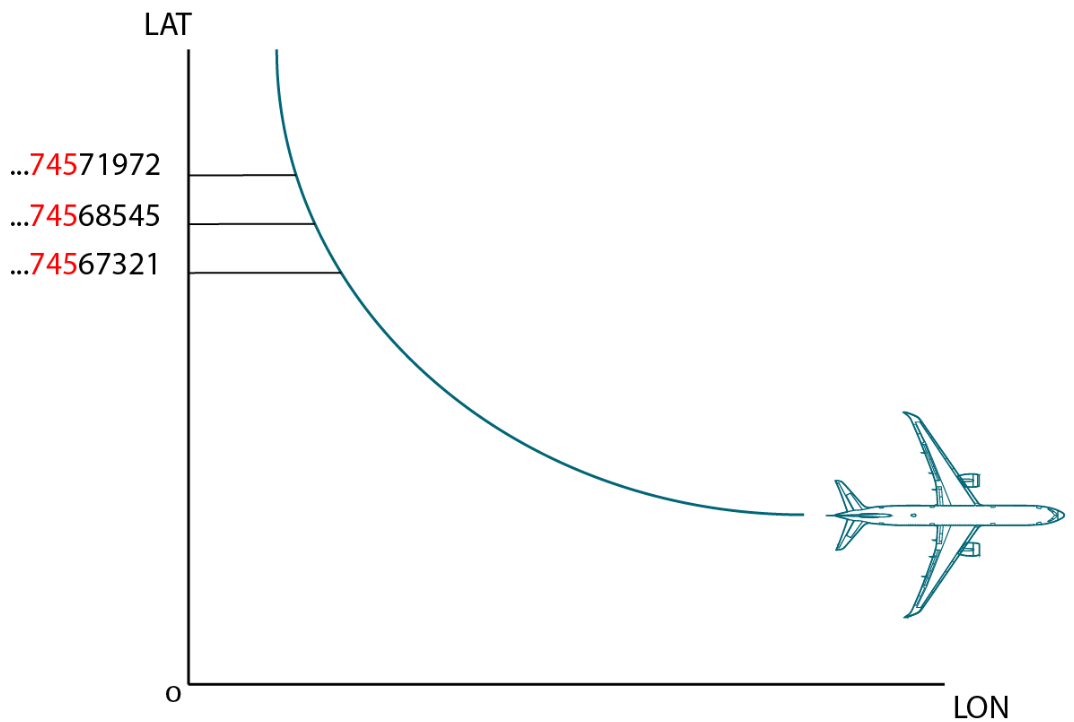
- Automatic Dependent Surveillance - Broadcast
  - Supports NextGen
    - Next generation of air traffic management systems
  - Aircraft periodically *broadcasts* accurate surveillance information to ground stations and near aircraft
    - position and velocity
  - *Automatic* no pilot intervention needed
  - *Dependent* on navigation system
- Mandatory on Jan 1, 2020 (in USA and Europe)
  - More than 40000 aircraft currently equipped

The ADS-B Protocol

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

Compact Position Reporting

Contiguous positions share prefixes

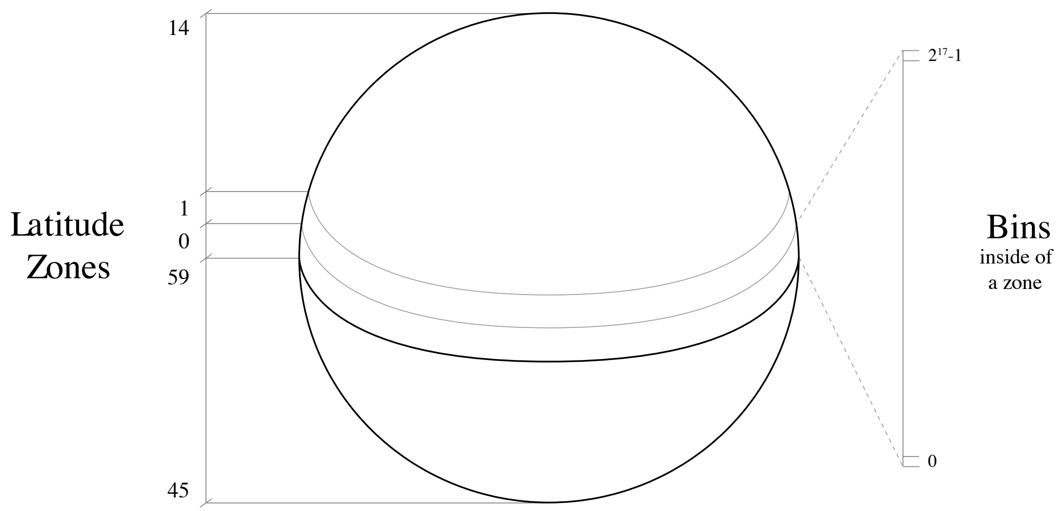


Idea: transmit only 17 less significative bits

Focus on Latitude First

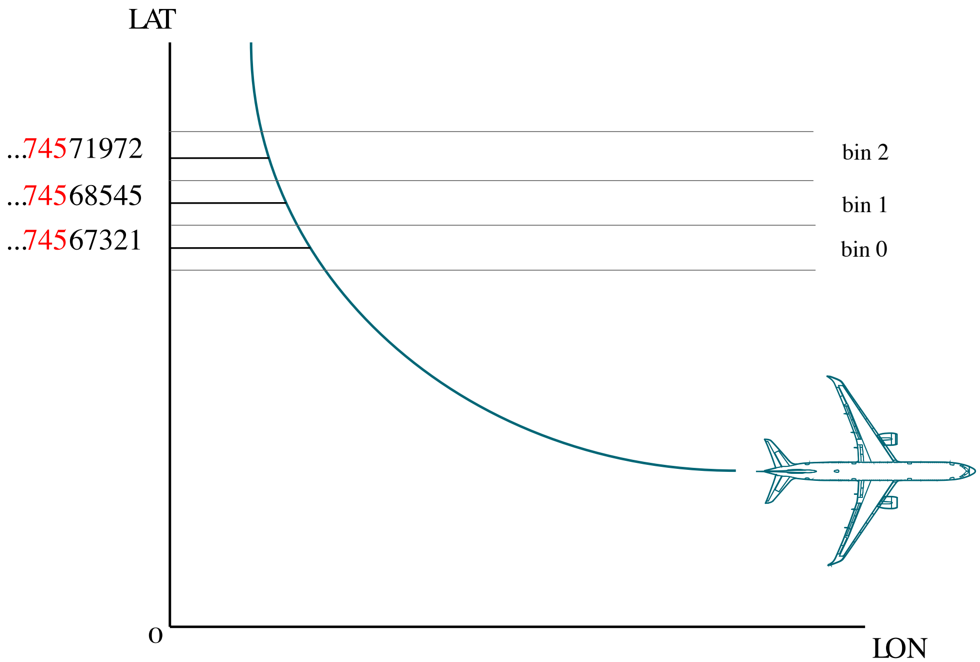
Latitude Zones

- Divide the globe into 60 equally sized zones
- Divide each zone in 2^{17} *bins*



Zone Size: $\mathrm{Dlat} = 360/60 = 6$ degrees

Reported Latitude



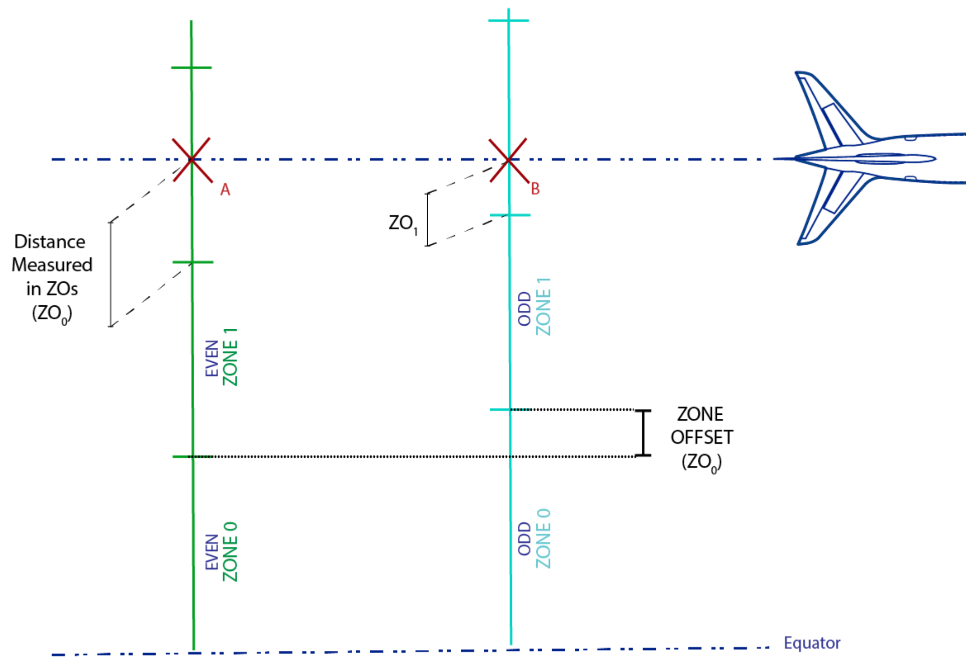
Broadcast only the corresponding *bin number* (YZ)

Encoding Latitude

- To encode lat, calculate:
  1. Distance from southern edge of enclosing zone
     - $\operatorname{mod}(\mathrm{lat}, \mathrm{Dlat})$
  2. Proportion w.r.t. the entire zone
     - $\operatorname{mod}(\mathrm{lat}, \mathrm{Dlat}) \cdot \frac{1}{\mathrm{Dlat}}$
  3. Correspondent *bin* number
     - $\operatorname{mod}(\mathrm{lat}, \mathrm{Dlat}) \cdot \frac{1}{\mathrm{Dlat}} \cdot 2^{17}$
  4. Round to the nearest integer
     - $\mathrm{ZY} = \left\lfloor \operatorname{mod}(\mathrm{lat}, \mathrm{Dlat}) \cdot \frac{1}{\mathrm{Dlat}} \cdot 2^{17} + \frac{1}{2} \right\rfloor$

How to Recover the Zone Index

Assuming Parallel-to-Equator Trajectory



Zone Index: $ZI := \lfloor ZO_0 - ZO_1 + 1/2 \rfloor$

More in General

Relaxing parallel-to-the-Equator restriction

- According to the standard, if two latitudes *A* and *B* are less than half zone offset appart from each other,
  - *A* and *B* lie in the same zone, or
  - *A* is one zone ahead w.r.t. *B*
- To deal with both cases

$$\mathrm{ZI} = \begin{cases} \operatorname{mod}\left(\lfloor \mathrm{ZO}_0 - \mathrm{ZO}_1 + 1/2 \rfloor, 60\right) & \text{even zone index} \\ \operatorname{mod}\left(\lfloor \mathrm{ZO}_0 - \mathrm{ZO}_1 + 1/2 \rfloor, 59\right) & \text{odd zone index} \end{cases}$$

Global Decoding

Given an *even* and an *odd* bin number YZ_0 and YZ_1, the recovered latitude Rlat_i is defined as

$$\mathrm{Rlat}_\mathrm{i}(\mathrm{YZ}_0, \mathrm{YZ}_1) := \mathrm{Dlat}_\mathrm{i}\left(\mathrm{mod}\left(\lfloor \mathrm{ZO}_0 - \mathrm{ZO}_1 + 1/2 \rfloor, 60 - \mathrm{i}\right) + \mathrm{YZ}_\mathrm{i}\frac{1}{2^{17}}\right)$$

where the zone offset (ZO_i) can be calculated as $\mathrm{ZO}_\mathrm{i} := \frac{\mathrm{Dlat}_\mathrm{i}}{\mathrm{ZO}} \cdot \frac{\mathrm{YZ}_\mathrm{i}}{2^{17}}$ with $\mathrm{i} \in \{0, 1\}$

Note that Rlat_i returns the <u>center</u> of the *bin* where the input latitude lies. Decoded latitude is at most at half-bin size from the input latitude

What About Longitudes?

Dealing with Longitudes

- Goal: same encoding resolution everywhere
  - as close to a constant as possible all around the globe
- Same idea
  - ~Equally sized zones divided in 2^{17} bins
- One distinctive feature
  - Longitude range shrink when approaching the poles
  - Number of longitude zones is a function of latitude
    - reducing the number of zones as latitude increases

NL Function

- $\mathrm{NL}(\mathrm{lat})$: number of even longitude zones at latitude lat

$$
\mathrm{NL}(\mathrm{lat}) = \begin{cases} 59 & \text{if } \mathrm{lat} = 0, \\ \left\lfloor 2\pi \left(\arccos\left(1 - \frac{1-\cos\left(\frac{\pi}{30}\right)}{\cos^2\left(\frac{\pi}{180}|\mathrm{lat}|\right)} \right) \right)^{-1} \right\rfloor & \text{if } |\mathrm{lat}| < 87, \\ 2 & \text{if } |\mathrm{lat}| = 87, \\ 1 & \text{if } |\mathrm{lat}| > 87. \end{cases}
$$

- In practice, computing this function is inefficient
  - A lookup table of transition latitudes is pre-calculated

Global Decoding

- Latitude, given two encoded latitudes

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

- Longitude, given two encoded positions

  $$\mathrm{Rlon_i}(\mathrm{YZ_0}, \mathrm{YZ_1}, \mathrm{XZ_0}, \mathrm{XZ_1}) := \mathrm{Dlon_i}\left(\mathsf{mod}\left(\left\lfloor \frac{(\mathrm{nl}-1)\mathrm{XZ_0} - \mathrm{nl}\cdot\mathrm{XZ_1}}{2^{17}} + \frac{1}{2} \right\rfloor, \mathrm{nl}'_i\right)\right)$$

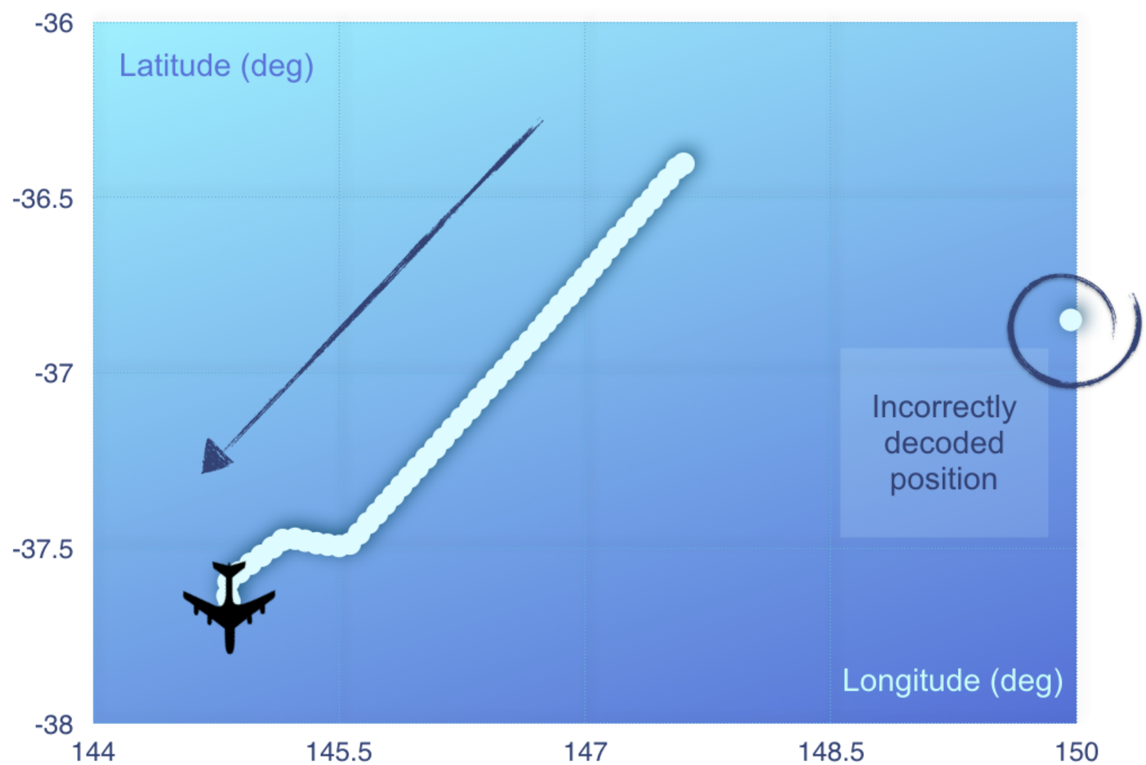
  where

  - $\mathrm{nl} := \mathrm{NL}(\mathrm{Rlat_0}(\mathrm{YZ_0}, \mathrm{YZ_1}))$, must be equal to $\mathrm{NL}(\mathrm{Rlat_1}(\mathrm{YZ_0}, \mathrm{YZ_1}))$
  - $\mathrm{nl}'_i := \max(\mathrm{nl} - \mathrm{i}, 1)$, since nl is 1 if $|\mathrm{Rlat_i}(\mathrm{YZ_0}, \mathrm{YZ_1})| > 87$
  - $\mathrm{Dlon_i} := 360/\mathrm{nl}'_i$

Local Decoding

- Additional decoding method
- Uses a *reference position* and one position message
  - instead of two position messages
- Positions appart for no more than half of a zone
  - According to the standard
  - Allows for bigger separation between received positions
- Idea: create a sliding region 1 zone wide
  - Centered on reference position
  - Each bin number occurs only once in the region

Known Issues



Reported by Airservices Australia (2007)

Analysis of the Algorithm

- Accomplishments:
  1. Found technical issues in the standard
     - Counterexamples for the real-valued model
  2. Amended version proven correct
     - Prototype Verification System (PVS)
  3. Proposed simpler formulation
     - reducing numerical complexity
  4. Prototype implementation formally verified
     - C, PVS, Frama-C, Gappa, Alt-Ergo

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

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

Technical Issues

- Counterexamples found for both decoding settings
  - Even Assuming (exact) real-valued arithmetics
  - For example, in the *global decoding* case
    - $\mathrm{lat}_0 = 363373617 \cdot 360/2^{32} \approx 30.4576247279$
    - $\mathrm{lat}_1 = 363980245 \cdot 360/2^{32} \approx 30.5084716994$
    - decoded positions are further away for more than a *bin*
- Correctness proved on tightened requirements
  - max. distance of input positions decreased by half-bin size

Numerical Simplifications

- Mathematically equivalent expressions suggested
  - Numerically simpler
  - Equivalence formally proven
- Example: equivalent calculation of NL lookup table
  - removing four operations in total
  - $\mathrm{lat}_{\mathrm{NL}}(\mathrm{nl}) := \frac{180}{\ }\arccos\left(\frac{\sin(\ \ /60)}{\sin(\ \ /\mathrm{nl})}\right).$
- Example: cancellation instead of division
  - Reducing complexity of encoding algorithm
  - $\frac{\mathrm{mod}(a,b)}{b} = \frac{a-b*\left\lfloor\frac{a}{b}\right\rfloor}{b} = \frac{a}{b} - \left\lfloor\frac{a}{b}\right\rfloor$

Example: Latitude Global Decoding

- According to the standard:

$$\mathrm{Rlat}_0(\mathrm{YZ}_0, \mathrm{YZ}_1) := \mathrm{Dlat}_0 \left(\mathsf{mod}\left(\left\lfloor \frac{59\mathrm{YZ}_0 - 60\mathrm{YZ}_1}{2^{17}} + \frac{1}{2}\right\rfloor, 60\right) + \frac{\mathrm{YZ}_0}{2^{17}}\right)$$

- Simplified version of global decoding (i=0) in ACSL

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

Example: Latitude Global Decoding

- Simplified version of global decoding (i=0) in ACSL

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

- Translated by hand into a PVS declaration

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

- Proven to be equivalent to version from the standard

Example: Latitude Global Decoding

```
/*@ requires 0 <= yz0 <= 131071; requires 0 <= yz1 <= 131071;
    requires \floor(yz0) == yz0; requires \floor(yz1) == yz1;
    ensures \abs(\result - rLatr(yz0,yz1)) <= 0.000022888; */
fp rLatf (int yz0, int yz1) {
  fp res, rLat1; fp dLatf = 360.0 / 60.0;
  fp j1f = (59.0 * yz0 - 60.0 * yz1 + 0x1.0p+16) * 0x1.0p-17;
  /*@ assert j1f:
    \let j1r = (59.0 * yz0 - 60.0 * yz1 + 0x1.0p+16) *0x1.0p-17;
    j1f == j1r; */
  fp jf = floor(j1f);
  /*@ assert jf:
    \let j1r = (59.0 * yz0 - 60.0 * yz1 + 0x1.0p+16) *0x1.0p-17;
    \let jr = \floor(j1r);
    jf == jr; */
  /*@ assert values_for_jf: -60.0 <= jf <= 59.0; */
  /*@ assert jf represents an integer: \floor(jf) == jf; */
```

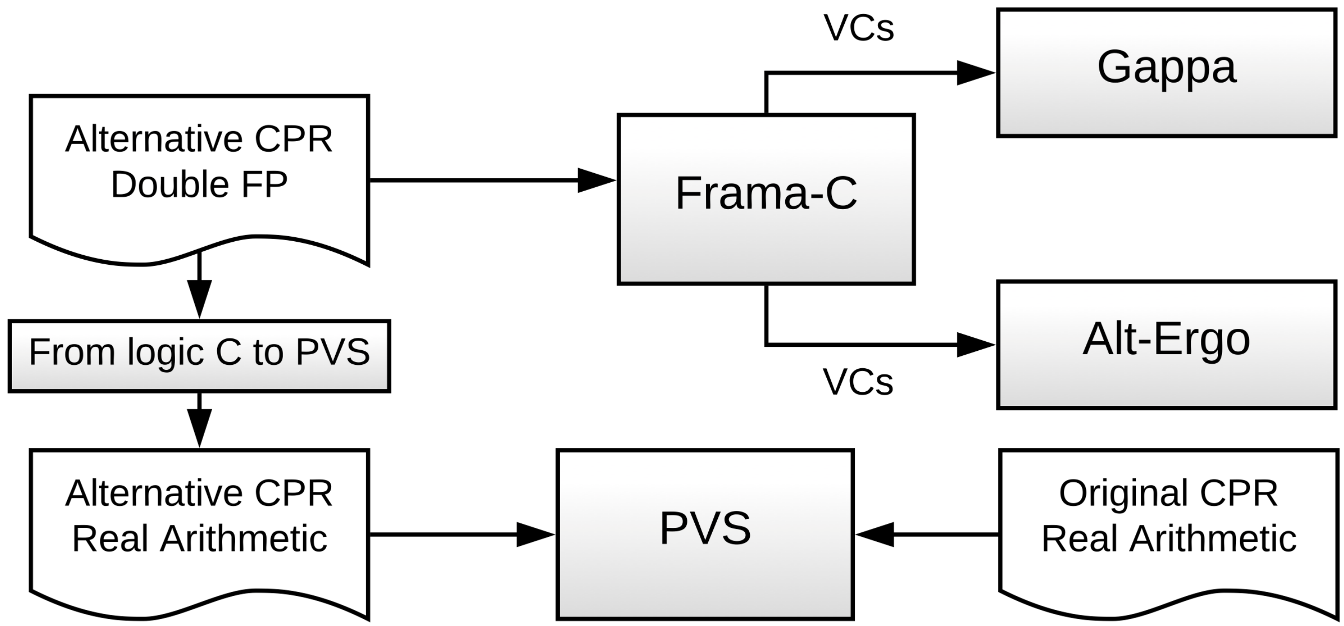
Frama-C/WP & Alt-Ergo+Gappa: the floating-point result is at most 0.000022888° apart from the logical result.

Verification Result

Floating-point version has the expected granularity: decoded and input positions are less than $\frac{1}{2}$ bin apart

- PVS version has been proved correct, i.e.,
  - decoded latitude lies in the center of a bin and
  - it is less than half bin apart from the input
- It coincides with the ACSL logic definition
- C version is less than $2.3 \cdot 10^{-5}$ apart from it

Verification Approach



- logic ACSL declarations translated to PVS by hand
- proved equivalent to existent CPR formalization
- C code verified using Frama-C/WP/Alt-Ergo/Gappa

Concluding Remarks

- Synergetic use of diverse analysis tools on
  - complex verification effort
  - relatively simple algorithm
    - no loops, no pointers, no arrays
- Proposed algorithm is being considered as reference implementation of CPR
  - RTCA DO-260B/Eurocae ED-102A

Future Work

- Extend results to other CPR modalities
  - ~~Airborne~~, Surface, Coarse TIS-B
- Develop CPR integer-valued version
  - correctness (PVS) + verified implementation (Frama-C)
- Analysis of Floating-Point Programs
  - Frama-C: WP plugin to export VCs directly to PVS
  - Floating-point programs: Frama-C + PRECiSA
    - http://precisa.nianet.org/