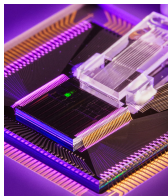


Exceptional service in the national interest



Automated Error Analysis of Numerical Kernels for High-Consequence Systems with Framac

DAHCS Late-Start LDRD 24-1299

Samuel D. Pollard (PI), Shant Hairapetian

Introduction to (Computer) Arithmetic



- Computers can only approximate real numbers
- The most common approximation is *floating point*
- Floating point can introduce error

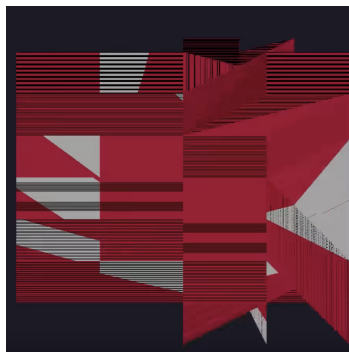
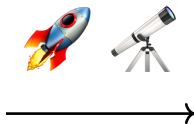
Introduction to (Computer) Arithmetic



- Computers can only approximate real numbers
- The most common approximation is *floating point*
- Floating point can introduce error (source: [4])



1km away



20,000km away

An Analogy



Always available

Vol. (μL)	Accuracy (μL)	Precision (μL)
0.5	± 0.040	≤ 0.013
1	± 0.025	≤ 0.012
5	± 0.060	≤ 0.020
10	± 0.080	≤ 0.025



versus

Rarely available

Input	Accuracy (rel. err)	Precision (relative)
[0.01,200]	0.4%	2^{-23}
[200,10 ⁵]	0.01%	2^{-23}

```
float Q_rsqrt(float x)
{
    long i;
    float x2, y;
    const float threehalfs = 1.5f;

    x2 = x * 0.5f;
    y = x;
    i = * ( long * ) &y;
    i = 0x5f3759df - ( i >> 1 );
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) );
    return y;
}
```



- High-consequence systems require scalable, generalizable verification tools



- High-consequence systems require scalable, generalizable verification tools
- Formal Methods (FM) provides mathematical, computer-checked proofs of correctness



- High-consequence systems require scalable, generalizable verification tools
- Formal Methods (FM) provides mathematical, computer-checked proofs of correctness
- FM is time consuming

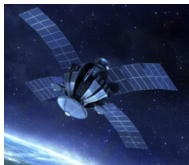


- High-consequence systems require scalable, generalizable verification tools
- Formal Methods (FM) provides mathematical, computer-checked proofs of correctness
- FM is time consuming
- Goal: improve automation for FM
 - in particular, with floating point

Specifying Software Using ACSL



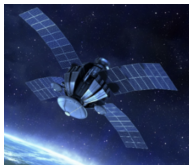
- ACSL = ANSI C Specification Language
- A first-order logic about C programs
- Frama-C transforms C + ACSL into *verification conditions* that automated reasoning tools can solve [3]



Specifying Software Using ACSL



- ACSL = ANSI C Specification Language
- A first-order logic about C programs
- Frama-C transforms C + ACSL into *verification conditions* that automated reasoning tools can solve [3]



```
/*@ requires \valid(a)
        \&& \valid(b);
   ensures *a == \old(*b)
        \&& *b == \old(*a);
   assigns *a, *b;
*/
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
    return;
}
```



- Recall: our goal is to improve automation for formal methods



- Recall: our goal is to improve automation for formal methods
- We first tried to link floating-point automated reasoning tools [2, 5] with Frama-C



- Recall: our goal is to improve automation for formal methods
- We first tried to link floating-point automated reasoning tools [2, 5] with Frama-C
- But we quickly discovered...

ACSL Isn't Quite Expressive Enough



- Propose 5 features
 1. Roundoff error
 2. uncertainty
 3. `real` vs `float` model
 4. Support `ulp`, relative error
 5. Syntax simplifications
- Require at least the first two:
 1. allows error analysis
 2. provides compositionality
- The other three improve usability

ACSL Isn't Quite Expressive Enough



- Propose 5 features

1. Roundoff error
2. uncertainty
3. real vs float model
4. Support ulp, relative error
5. Syntax simplifications

- Require at least the first two:

1. allows error analysis
2. provides compositionality



- The other three improve usability

```
1. /*@
    ensures \round_error(\result) <= 1e-8;
 */
double foo(double x);
```

```
2. /*@
    requires x \in [-5.0, 5.0]
    requires x \uncertainty(x, -1e-4, 1e-4);
 */
double foo(double x);
```

Outcomes





- Draft document of ACSL extensions for floating-point
- Grow collaboration between Sandia, the French Center for Atomic Energy (CEA-List), & NASA
- This project and related work funded by NNSA Advanced Simulation and Computing will be published in *The International Workshop on Numerical and Symbolic Abstract Domains* [1]



Software Analyzers

ANSI/ISO C Specification Language:
Floating-Point Draft Syntax
Version α

Samuel D. Pollard, Laura Titolo, Mariano Moscato, Maxime Jacquemin



Work licensed under Creative Commons BY license
<https://creativecommons.org/licenses/by/4.0/>

© 2024-2024 CEA-List, Sandia National Laboratories, NASA



- SAND report will have technical details (on OSTI)
- Upcoming technical talk for Multi-institutional Community of Practice (MiCoP)
- Software in the process of being open-sourced
- Future work
 - Agree on ACSL features and scope implementation effort
 - Link automated reasoning tools with Frama-C

`https://proof.sandia.gov`



- [1] DARIO, A., AND POLLARD, S. D.
A step-function abstract domain for granular floating-point error analysis.
In *10th International Workshop on Numerical and Symbolic Abstract Domains (2024)*, NSAD.
To appear.
- [2] FERNANDES FERREIRA, N. B., MOSCATO, M. M., TITOLO, L., AND AYALA-RINCÓN, M.
A provably correct floating-point implementation of well clear avionics concepts.
In *Formal Methods in Computer-Aided Design (2023)*, FMCAD, pp. 237–246.
- [3] KOSMATOV, N., PREVOSTO, V., AND SIGNOLES, J., Eds.
Guide to Software Verification with Frama-C (2024), Springer Cham.
- [4] MARCHEVSKY, S.
Float point single precision error in graphics, 2018.
<https://www.youtube.com/watch?v=wGhBjMcY2YQ>.
- [5] SOLOVYEV, A., JACOBSEN, C., RAKAMARIĆ, Z., AND GOPALAKRISHNAN, G.
Rigorous estimation of floating-point round-off errors with symbolic taylor expansions.
In *LNCS (Oslo, Norway, 2015)*, N. Bjørner and F. de Boer, Eds., vol. 9109 of *20th International Symposium on Formal Methods (FM)*, Springer, pp. 532–550.