Karlsruhe Institute of Technology

Analyzing Floating-Point Programs

Debasmita Lohar

Formal Systems II: Applications, Summer 2025

In this Part

you will learn the fundamentals of analyzing floating-point programs*:

27.06.

- Challenges
- Basics of Floating-Point Arithmetic
- Dataflow analysis
  - Interval Arithmetic (IA)
  - Floating-Point IA
  - Analysis of roundoff errors with IA
  - Demo: Daisy

Today

More on Dataflow Analysis

- Pro’s and con’s of Intervals
- Affine Arithmetic (AA)
- Analysis of roundoff errors with AA
- Interval Subdivision
- Other Approaches and Recent Directions
- Demo: Daisy

*Slides are based on Program Analysis Course (WS 20/21) at UdS + RPTU

Pro's and con's of Intervals

Pro's:

- Conceptually simple (important for correctness)
- Fast efficient machine-independent implementation
- Several successful applications

Con's:

- Can be imprecise: not relational, i.e. cannot track correlations between variables

```
x = [-1, 2]
x - x = [-1, 2] - [-1, 2]
      = [-3, 3]
      ≠ [0, 0]
```

1min

In a chained computation intervals can become too wide to be useful!

Affine Arithmetic (AA)

- Improves over intervals in that it tracks *linear* correlations
- Represents each range as a linear (affine) form:

$$\hat{x} := x_0 + \sum_{i=1}^{n} x_i \epsilon_i \qquad \epsilon \in [-1, 1]$$

  - ➡ x_0 is the *central value*
  - ➡ $x_i \epsilon_i$ are *noise terms*, tracking deviation from x_0
  - ➡ ϵ_i are symbolic variables, tracking correlations

- The range represented by an affine form:

$$[\hat{x}] = [\ x_0 - \sum_{i=1}^{n} |x_i|, \quad x_0 + \sum_{i=1}^{n} |x_i|\ ]$$

- AA tracks linear correlations: $\hat{x} = x_0 + x_1 \epsilon_1$

$$\hat{x} - \hat{x} = x_0 + x_1\epsilon_1 - (x_0 + x_1\epsilon_1) = x_0 - x_0 + x_1\epsilon_1 - x_1\epsilon_1 = 0$$

Affine Arithmetic: concepts and applications, L. H. De Fiqueiredo, J. Stolfi, Numerical Algorithms'04

Quiz

2min

$$\hat{x} := x_0 + \sum_{i=1}^{n} x_i \epsilon_i \qquad \epsilon \in [-1,1]$$

What is the affine form corresponding to the following interval?

$$x = [2,\ 10]$$

a)

$$\hat{x} = 6 + 4\epsilon_1$$

b)

$$\hat{x} = 2 + 8\epsilon_1$$

c)

$$\hat{x} = 6 + 8\epsilon_1$$

AA Arithmetic Operations

- Addition, subtraction, unary minus are linear and can thus be computed exactly:

$$\hat{z} = \hat{x} + \hat{y} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i + y_0 + \sum_{i=1}^{n} y_i \epsilon_i$$

$$z_0 = x_0 + y_0 \qquad z_i \epsilon_i = (x_i + y_i)\epsilon_i$$

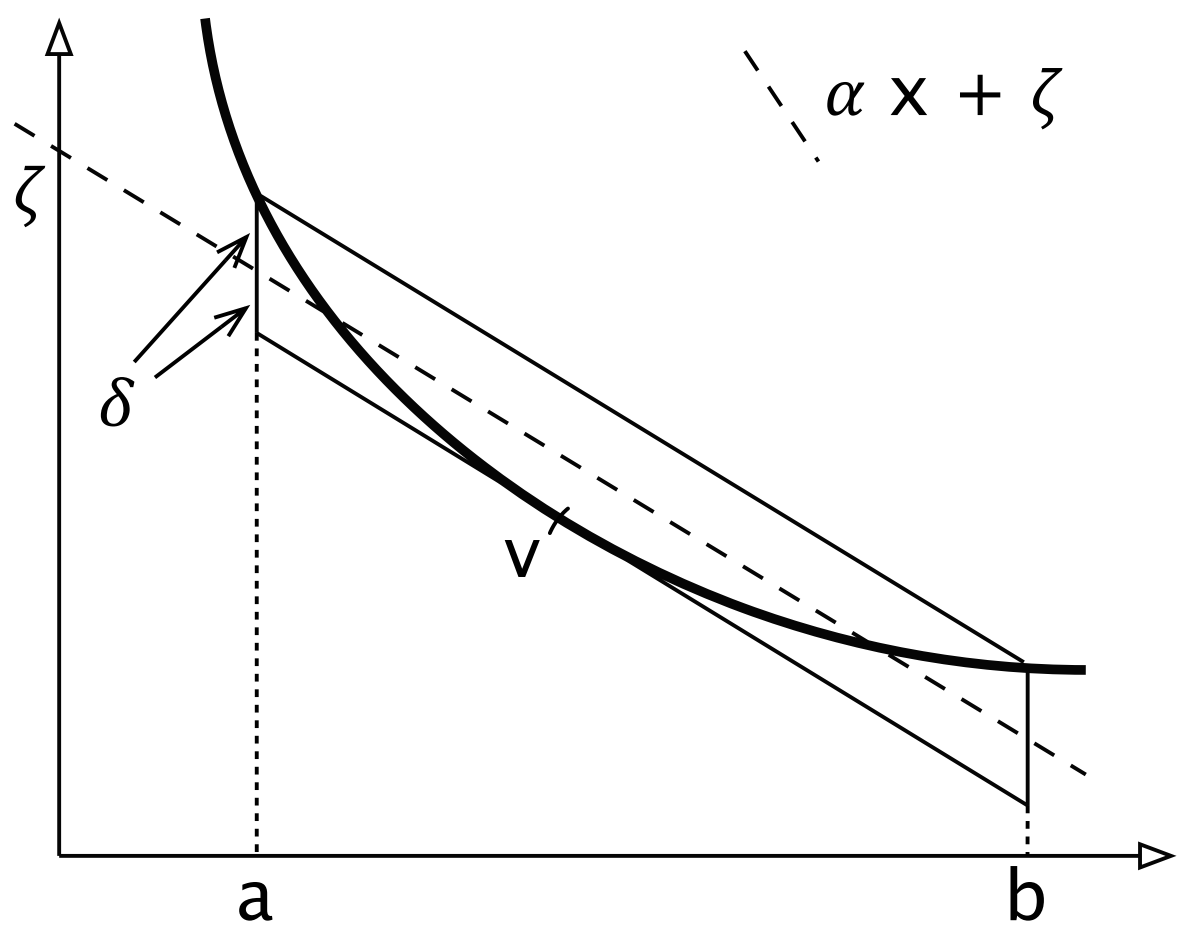
- Multiplication is nonlinear:

$$\begin{aligned}
\hat{z} = \hat{x} \cdot \hat{y} &= (x_0 + \sum_{i=1}^{n} x_i \epsilon_i) \cdot (y_0 + \sum_{i=1}^{n} y_i \epsilon_i) \\
&= (x_0 \cdot y_0) + (x_0 \cdot \sum_{i=1}^{n} y_i \epsilon_i) + (y_0 \cdot \sum_{i=1}^{n} x_i \epsilon_i) + (\sum_{i=1}^{n} x_i \epsilon_i \cdot \sum_{i=1}^{n} y_i \epsilon_i) \\
&= (x_0 \cdot y_0) + \sum_{i=1}^{n} (x_0 y_i + y_0 x_i)\epsilon_i + \sum_{1 \le i,j \le n} |x_i y_j| \, \epsilon_{n+1}
\end{aligned}$$

over-approximation!

AA Arithmetic Operations

- Unary operations (sqrt, inverse, log, exp, etc.):
  - ➡ compute linear approximation



Quiz

2min

What is the result of the following computation?

$$\hat{x} = 3 + 2\epsilon_1 + 4\epsilon_2 \qquad \hat{y} = 1 + 1\epsilon_2 + 5\epsilon_3$$

$$\hat{z} = \hat{x} - \hat{y} =$$

a)

$$\hat{z} = 2 + 3\epsilon_2$$

c)

$$\hat{z} = 2 + 2\epsilon_1 + 3\epsilon_2$$

b)

$$\hat{z} = 2 + 2\epsilon_1 + 3\epsilon_2 - 5\epsilon_3$$

Roundoff Analysis with AA

For arithmetic operations: $eval(x \odot y)$

- real-valued range: $x_{range} \odot y_{range}$
- error: $propagatedError + newRoundoffError$

Recall the roundoff error analysis

Roundoff Analysis with AA

For arithmetic operations: $eval(x \odot y)$

- real-valued range: $x_{range} \odot y_{range}$
- error: $propagatedError + newRoundoffError$

To improve the precision, we have several options to replace IA with AA:

Option 1: AA for ranges and errors

- ➡ operations over AA use AA rules
- ➡ error propagation remains the same

Roundoff Analysis with AA

For arithmetic operations: $eval(x \odot y)$

- real-valued range: $x_{range} \odot y_{range}$
- error: $propagatedError + newRoundoffError$

To improve the precision, we have several options to replace IA with AA:

Option 1: AA for ranges and errors

- ➡ operations over AA use AA rules
- ➡ error propagation remains the same

Option 2: AA for ranges and IA for errors

- ➡ operations over AA use AA rules
- ➡ error propagation remains nearly the same
  - ➡ <u>need to cast AA into an interval</u>

Roundoff Analysis with AA

For arithmetic operations: $eval(x \odot y)$

- real-valued range: $x_{range} \odot y_{range}$
- error: $propagatedError + newRoundoffError$

To improve the precision, we have several options to replace IA with AA:

Option 1: AA for ranges and errors

➡ operations over AA use AA rules

➡ error propagation remains the same

Option 2: AA for ranges and IA for errors

➡ operations over AA use AA rules

➡ error propagation remains nearly the same

➡ <u>need to cast AA into an interval</u>

Option 3: IA for ranges and AA for errors

➡ operations over AA use AA rules

➡ error propagation remains nearly the same

➡ <u>need to cast an interval into an AA</u>

Different Kinds of Approximations

Affine arithmetic is not universally better than interval arithmetic.

They commit different kinds of (over-)approximations:

- ➡ interval arithmetic looses correlations (`x - x ≠ 0`)
- ➡ affine arithmetic over-approximates nonlinear operations

- AA over-approximation can be larger than due to interval's loss of correlations
- AA gives best possible result for linear programs
- AA over-approximation is smaller when input intervals are smaller
  - ✓ Daisy uses AA by default to track errors, and intervals to track ranges

Interval Subdivision

is an additional technique to reduce over-approximations

➡ standard technique in numerical analysis

<u>Idea:</u>

- split input intervals into smaller subintervals
- form cartesian product to get all possible subdomains
- run analysis on each subdomain
- worst-case error is the overall maximum

Interval Subdivision

Example:

`x = [-10, 10], y = [0, 5]`

Splitting each input domain into 4:

x_1 = [−10, −5], x_2 = [−5, 0], x_3 = [0, 5], x_4 = [5, 10]

y_1 = [0, 1.25], y_2 = [1.25, 2.5], y_3 = [2.5, 3.75],

y_4 = [3.75, 5]

Run analysis on each subdomain:

(x_1, y_1), (x_1, y_2), (x_1, y_3), (x_1, y_4), (x_2, y_1), ...

Alternative Roundoff Error Analysis

Recall the absolute error:

$$\max_{x \in I} |f(x) - \tilde{f}(\tilde{x})|$$

This is fundamentally an **optimization problem**

- ➡ suggest an alternative static analysis based on global optimization

Problem:

- ➡ $\tilde{f}$ is highly discontinuous and complex
- ➡ formulation as-is cannot be handled by optimization tools

Idea:

- ➡ approximate $\tilde{f}$ using floating-point abstraction ($x \circ_{fp} y = (x \circ y) \cdot (1 + \delta)$)
- ➡ simplify formula further using first-order Taylor approximation

Rigorous Estimation of Floating-Point Round-off Errors with Symbolic Taylor Expansions, A. Solovyev, C. Jacobsen, Z. Rakamaric, G. Gopalakrishnan, FM'15

Recent Research Directions

Probabilistic Error Analysis

The worst-case analysis may be too pessimistic!

✔ for applications that may tolerate large infrequent errors.

Probabilistic Analysis of Errors:

➡ considers probability distribution of x

➡ propagates the distribution through the program using probabilistic AA

$$\hat{x} := x_0 + \sum_{i=1}^{n} x_i \epsilon_i \qquad \epsilon \in [-1,1]$$

noise symbol propagates discretized distributions

$$d_x :< [a_1, b_1], w_1 > , \cdots < [a_n, b_n], w_n >$$

set of <interval, weight>

➡ combined with interval subdivisions

Sound Probabilistic Numerical Error Analysis, D. Lohar, M. Prokop, E. Darulova, iFM'19

Rewriting Optimization

If the accuracy is still not acceptable:

- ➡ increase precision and check again
- ➡ rewrite equations, check accuracy

Examples:

$$\texttt{a + (b + c)} \xrightarrow{\text{mathematical identities}} \texttt{(a + b) + c}$$

$$\texttt{a * (b + c)} \longrightarrow \texttt{(a * b) + (a * c)}$$

Finding an optimal order is computationally infeasible!

- ➡ Daisy uses a genetic (heuristic) search to find better expressions
  - tournament selection of an expression based on fitness
  - randomly mutate
  - evaluate fitness of the new ones

Sound Mixed Precision Optimization with Rewriting, E. Darulova, E. Horn, S. Sharma, ICCPS'18

Mixed-Precision Tuning

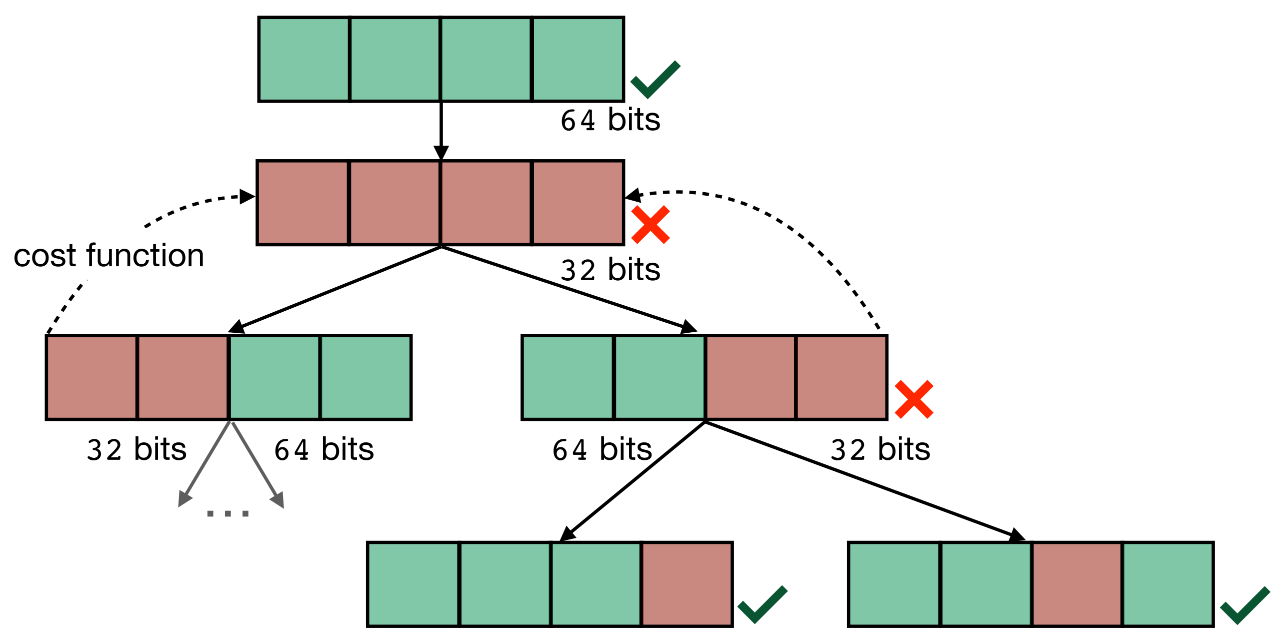
Assigning one precision to all variables may be suboptimal:

➡ Optimize precision to increase performance

```
minimize: precision and static cost
```

`subject to:` $\epsilon_n \leq \epsilon_{target}$

➡ Daisy uses a delta debugging search to find the optimal precision



Sound Mixed Precision Optimization with Rewriting, E. Darulova, E. Horn, S. Sharma, ICCPS'18

Limitations

Static analyses compute sound worst-case and probabilistic roundoff errors, and support rewriting and mixed-precision tuning.

But, they have limitations in:

- scalability for large programs
  - (any) scalable analysis + static analysis*
- efficiency for high-precision analysis
- fine-grained optimizations
  - tailor to application contexts**

*A Two-Phase Approach for Conditional Floating-Point Verification, D. Lohar, C. Jeangoudoux, J. Sobel, E. Darulova, M. Christakis, TACAS'21

**Sound Mixed Fixed-Point Quantization of Neural Networks", D. Lohar, C. Jeangoudoux, A. Volkova, E. Darulova, EMSOFT'23

**Of Good Demons and Bad Angels: Guaranteeing Safe Control under Finite Precision, S. Teuber, D. Lohar, B. Beckert, FMCAD'25

Demo ~35min

Installation:

```
- sudo docker pull dlohar/daisy
- sudo docker run -it dlohar/daisy
```

`float-analysis-class/Test.scala:`

```
import daisy.lang._
import Real._
object Test {
  def test(x: Real, y: Real): Real = {
    require(1 <= x && x <= 3 && -5 <= y && y <= 4)

    val z = x * y + x + y + 2*x*x
    z
  } ensuring (res => res +/- 1e-5)
}
```

Daisy - Framework for Analysis and Optimization of Numerical Programs,
E. Darulova, A. Izycheva, F. Nasir, F. Ritter, H. Becker, R. Bastian, TACAS'18

Demo ~35min

Running Daisy:

```
- sbt script
- ./daisy float-analysis-class/Test.scala
   --rangeMethod=affine --errorMethod=affine
   --precision=Float32
```

Demo ~35min

Running Daisy:

```
- sbt script
- ./daisy float-analysis-class/Test.scala
   --rangeMethod=affine --errorMethod=affine
   --precision=Float32
- ./daisy float-analysis-class/Test.scala
   --rangeMethod=affine --errorMethod=affine
   --precision=Float32 --subdiv --divLimit=10
```

— for subdivision Z3 is needed which is currently not installed!

Demo ~35min

Running Daisy:

```
- sbt script
- ./daisy float-analysis-class/Test.scala
   --rangeMethod=affine --errorMethod=affine
   --precision=Float32
- ./daisy float-analysis-class/Test.scala
   --rangeMethod=affine --errorMethod=affine
   --precision=Float32 --subdiv --divLimit=10
- ./daisy float-analysis-class/Test.scala
   --precision=Float32 --rewrite
- ./daisy float-analysis-class/Test.scala
   --precision=Float32 --mixed-tuning --codegen
```

Demo ~35min

Exercise 1

- Find the lowest uniform precision that guarantees the error bound.
- Which range and error analysis method works better?
- Can you optimize more to improve accuracy/efficiency?
- Generate an optimized code (scala, apfixed)

```
import daisy.lang._
import Real._
object Test {
  def test(x: Real, y: Real): Real = {
    require(1 <= x && x <= 3 && -5 <= y && y <= 4)

    val z = x * y + x + y + 2*x*x
    z
  } ensuring (res => res +/- 1.4e-14)
}
```