Probabilistic Analysis of Programs with Numerical Uncertainties

Debasmita Lohar

under the supervision of

Eva Darulova

PhD-iFM 2019
• **Reals** are implemented in **Floating point/Fixed point** data type

```python
def func(x: Real, y: Real, z: Real): Real = {
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res
}
```
Programming with Numerical Uncertainties

```python
(x:Float32, y:Float32, z:Float32): Float32
def func(x:Real, y:Real, z:Real): Real = {
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res
}
```

- Reals are implemented in Floating point/Fixed point data type
- Introduces Round-off error in the computation
Why should we care about Round-off Errors?

Reals are implemented in Floating point/ Fixed point data type

Introduces Round-off error in the computation

Program can take a wrong decision

```python
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    val res = -3.79 * x - 5.44 * y + 9.73 * z + 4.52
    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    return res
}
```

real valued program

finite precision program
Several tools compute the Worst case error analysis

```python
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res
}
```

State-of-the-art: Worst Case Error Analysis

Daisy, Gappa, FLUCTUAT, rosa, FPTaylor, ....
A program **always** takes the wrong path in the **worst case**
Worst Case Analysis for Discrete Decisions

def func(x: Float32, y: Float32, z: Float32): Float32 = {
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    return res

A program always takes the wrong path in the worst case

Need to consider the probability distributions of inputs
A program **always** takes the wrong path in the **worst case**.

Need to consider the **probability distributions** of inputs.

What happens if we have **Approximate Hardware**?
Approximate Hardware

Resource Efficient but has Probabilistic Error behaviors
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res
}

Error Specification: <0.00199, 0.9>, <0.00499, 0.1>

• Has Probabilistic Error Specification
Error Resilient Applications

def func(x: Float32, y: Float32, z: Float32): Float32 = {
  require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
  val res = -3.79*x - 5.44*y + 9.73*z + 4.52
  return res
} ensuring (res +/- 0.00199, 0.85)

Error Specification: <0.00199, 0.9>, <0.00499, 0.1>

Application tolerates big errors occurring with 0.15 probability

- Has Probabilistic Error Specification
- Applications may tolerate large infrequent errors
Worst Case Analysis for Error Resilient Application

\[
\begin{align*}
(x &: \text{Float64}, y &: \text{Float64}, z &: \text{Float64}): \text{Float64} \\
\text{def func}(x &: \text{Float32}, y &: \text{Float32}, z &: \text{Float32}): \text{Float32} &= \{ \\
\text{require } (0.0 \leq x \leq 4.6 \land 0.0 \leq y, z \leq 10.0) \\
\text{val res} &= -3.79 \times x - 5.44 \times y + 9.73 \times z + 4.52 \\
\text{return res} \\
\} \text{ ensuring } (\text{res} +/\ - 0.00199, 0.85)
\end{align*}
\]

Worst Case Error Analysis

error: 0.002
Worst Case Analysis = Low Resource Utilization

\[
\text{def func}(x: \text{Float32}, y: \text{Float32}, z: \text{Float32}): \text{Float32} = \{
\text{require} \ (0.0 \leq x \leq 4.6 \land 0.0 \leq y, z \leq 10.0)
\text{val res} = -3.79 \times x - 5.44 \times y + 9.73 \times z + 4.52
\text{return res}
\} \text{ ensuring } (\text{res} +\ 0.00199, 0.85)
\]

Worst Case Error Analysis

error: 0.002

Occurs only with probability 0.002!
Worst Case Analysis = Low Resource Utilization

```python
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res
}
```

ensuring (res +/- 0.00199, 0.85)

Worst Case Error Analysis

error: 0.002

Occurs only with probability 0.002!

Need to consider the probability distributions of inputs
Two Problems

def func(x: Float32, y: Float32, z: Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    if (res <= 0.0)
        raiseAlarm() real valued program
    else
        doNothing() finite precision program
    return res
}

How often does a program take a wrong decision?
by taking into account the probability distribution of inputs

How do we compute a precise bound on the error

https://github.com/malyzajko/daisy/tree/probabilistic
How often does a program take a wrong decision?

"Discrete Choice in the Presence of Numerical Uncertainties", EMSOFT'18

Eva Darulova

Sylvie Putot

Eric Goubault
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    return res
}
def func(x:Float32, y:Float32, z:Float32): Float32 = {
  require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)

  x := gaussian(0.0, 4.6)
  y := gaussian(0.0, 10.0)
  z := gaussian(0.0, 10.0)

  val res = -3.79*x - 5.44*y + 9.73*z + 4.52
  if (res <= 0.0)
    raiseAlarm()
  else
    doNothing()
  return res
}
Our Goal: Probabilistic Analysis

def func(x:Float32, y:Float32, z:Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)

    x := gaussian(0.0, 4.6)
    y := gaussian(0.0, 10.0)
    z := gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52

    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    return res
}

Compute Wrong Path Probability
Overview: Sound Analysis

Finite Precision Program with Probabilistic Inputs

Wrong Path Probability (WPP)
Round-off Error Analysis

Overview: Sound Analysis

Worst case

Round-off Error \( (e) \)

\[ e = 0.01 \]
Overview: Sound Analysis

Finite Precision Program with Probabilistic Inputs

Worst case

Round-off Error (e)

Decision Threshold (T)

\( e = 0.01 \)

if \( \text{res} \leq 0.0 \) raiseAlarm()
else doNothing()
Overview: Sound Analysis

Worst case

Round-off Error (e)

Finite Precision Program with Probabilistic Inputs

Decision Threshold (T)

Critical Interval [T-e, T+e]

Wrong Path Probability (WPP)

T = 0.0

[ 0.0 - 0.01, 0.0 + 0.01 ]

e = 0.01
Distribution Propagation

Finite Precision Program with Probabilistic Inputs

Worst case

Round-off Error (e)

Decision Threshold (T)

Critical Interval [T-e, T+e]

Probabilistic Analysis

Symbolic Inference (PSI)

OR

Static Analysis

"PSI: Exact Symbolic Inference for Probabilistic Programs", S. Misailovic, M. Vechev, and T. Gehr, CAV 2016
Distribution Propagation

Finite Precision Program with Probabilistic Inputs

Probabilistic Analysis
- Symbolic Inference (PSI)
- Static Analysis

Worst case
- Round-off Error (e)

Decision Threshold (T)

Critical Interval [T-e, T+e]

Intersection

Wrong Path Probability (WPP)
Distribution Propagation

Finite Precision Program with Probabilistic Inputs

Worst case

Probabilistic Analysis

Does not scale!

Decision Threshold (T)

Round-off Error (e)

Critical Interval [T-e, T+e]

Intersection

Wrong Path Probability (WPP)
Distribution Propagation

Finite Precision Program with Probabilistic Inputs

Worst case

Round-off Error (e)

Decision Threshold (T)

Critical Interval [T-e, T+e]

Intersection

Probabilistic Analysis

Symbolic Inference

OR

Static Analysis

Wrong Path Probability (WPP)
Using **Probabilistic Static Analysis**

- Input Intervals
- n subdivided Intervals
- Weighted sum over all subdomains
- Probability = 0.0
- Probabilistic Analysis

Is the critical interval reachable?
- not reachable
- reachable / timeout
# Results: Wrong Path Probability

Wrong Path Probability for 32 bit floating-point round-off errors and uniform input distributions

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>#ops</th>
<th>WPP using Sym. Inf.</th>
<th>WPP using Probabilistic with Subdiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>sine</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sqrt</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>turbine1</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>traincar2</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doppler</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bspline1</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rigidbody1</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>traincar1</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bspline0</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sineorder3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results: Wrong Path Probability

Wrong Path Probability for 32 bit floating-point round-off errors and uniform input distributions

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>#ops</th>
<th>WPP using Sym. Inf.</th>
<th>WPP using Probabilistic with Subdiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>sine</td>
<td>18</td>
<td>7.61E-07</td>
<td></td>
</tr>
<tr>
<td>sqrt</td>
<td>14</td>
<td>8.74E-06</td>
<td></td>
</tr>
<tr>
<td>turbine1</td>
<td>14</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>traincar2</td>
<td>13</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>doppler</td>
<td>10</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>bspline1</td>
<td>8</td>
<td>2.54E-06</td>
<td></td>
</tr>
<tr>
<td>rigidbody1</td>
<td>7</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>traincar1</td>
<td>7</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>bspline0</td>
<td>6</td>
<td>1.05E-05</td>
<td></td>
</tr>
<tr>
<td>sineorder3</td>
<td>4</td>
<td>1.90E-06</td>
<td></td>
</tr>
</tbody>
</table>
## Results: Wrong Path Probability

Wrong Path Probability for 32 bit floating-point round-off errors and uniform input distributions

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>#ops</th>
<th>WPP using Sym. Inf.</th>
<th>WPP using Probabilistic with Subdiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>sine</td>
<td>18</td>
<td>7.61E-07</td>
<td>6.45E-05</td>
</tr>
<tr>
<td>sqrt</td>
<td>14</td>
<td>8.74E-06</td>
<td>9.38E-05</td>
</tr>
<tr>
<td>turbine1</td>
<td>14</td>
<td>TO</td>
<td>4.82E-02</td>
</tr>
<tr>
<td>traincar2</td>
<td>13</td>
<td>TO</td>
<td>9.17E-02</td>
</tr>
<tr>
<td>doppler</td>
<td>10</td>
<td>TO</td>
<td>2.17E-02</td>
</tr>
<tr>
<td>bspline1</td>
<td>8</td>
<td>2.54E-06</td>
<td>1.95E-05</td>
</tr>
<tr>
<td>rigidbody1</td>
<td>7</td>
<td>TO</td>
<td>7.06E-02</td>
</tr>
<tr>
<td>traincar1</td>
<td>7</td>
<td>TO</td>
<td>1.86E-02</td>
</tr>
<tr>
<td>bspline0</td>
<td>6</td>
<td>1.05E-05</td>
<td>6.06E-05</td>
</tr>
<tr>
<td>sineorder3</td>
<td>4</td>
<td>1.90E-06</td>
<td>1.23E-04</td>
</tr>
</tbody>
</table>
Two Problems

def func(x: Float32, y: Float32, z: Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    return res
}

How often does a program take a wrong decision?
by taking into account the probability distribution of inputs

How do we compute a precise bound on the error?

https://github.com/malyzajko/daisy/tree/probabilistic
How do we compute a precise bound on the error?

"Sound Probabilistic Numerical Error Analysis", iFM'19

The talk is on 6th!
Our Goal

```python
def func(x:Float32, y:Float32, z:Float32): Float32 = {
    x := gaussian(0.0, 4.6)
    y := gaussian(0.0, 10.0)
    z := gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
}
```

Ensuring (error <= 0.00199, 0.85)
Our Goal

def func(x:Float32, y:Float32, z:Float32): Float32 = {
    x := gaussian(0.0, 4.6)
    y := gaussian(0.0, 10.0)
    z := gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
} ensuring (error <= 0.00199, 0.85)

- Compute probability distribution of error
Our Goal

```python
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    x := gaussian(0.0, 4.6)
    y := gaussian(0.0, 10.0)
    z := gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
}

ensuring (error <= 0.00199, 0.85)
```

- Compute **probability distribution** of **error**
- Compute a **smaller error** given a **threshold**
In this talk

```python
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    x := gaussian(0.0, 4.6)
    y := gaussian(0.0, 10.0)
    z := gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
}
```

ensuring (error <= 0.00199, 0.85)

- Compute probability distribution of error
- Compute a smaller error given a threshold considering Probabilistic Error Specification
  \[<0.019, 0.9>, <0.049, 0.1>\]
Finite Precision Program with Probabilistic Inputs

Probabilistic Round-off Error Analysis

Error Metric Extraction

Threshold Probability

Error, Probability

def func(..) {
    x := gaussian(0.0, 4.6)
    y := gaussian(0.0, 10.0)
    z := gaussian(0.0, 10.0)
    res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res
}

Error Spec: \(<0.019, 0.9>, <0.049, 0.1>\)
Probabilistic Error Analysis

Finite Precision Program with Probabilistic Inputs

Probabilistic Interval Subdivision

Probabilistic Error Analysis

Error Metric Extraction

Threshold Probability

Error, Probability
Probabilistic Interval Subdivision

- Keeps the probabilities of the subdomains
- Generates a set of subdomains with their probabilities
Probabilistic Error Analysis

- Keeps the probabilities of the subdomains
- Generates a set of subdomains with their probabilities
- **Probabilistic Error Analysis** for each subdomain
- Normalize error distribution with probabilities of subdomains
Finite Precision Program with Probabilistic Inputs

Probabilistic Error Analysis

Probabilistic Interval Subdivision

Probabilistic Error Analysis

Error Metric Extraction

Threshold Probability

Error, Probability

<[-0.0048, -0.0017], 0.09>,
<[-0.0043, 0.0014], 0.05>,
<[-0.0042, 0.0014], 0.02>,
<[-0.0040, 0.0019], 0.04>,
........
<[-0.0018, 0.0034], 0.04>
Threshold probability = 0.85

<table>
<thead>
<tr>
<th>Cumulative Prob.</th>
<th>Error Range</th>
<th>Cumulative Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>$[-0.0048, -0.0017], 0.09$</td>
<td></td>
</tr>
<tr>
<td>0.91</td>
<td>$[-0.0043, 0.0014], 0.05$</td>
<td></td>
</tr>
<tr>
<td>0.86</td>
<td>$[-0.0042, 0.0014], 0.02$</td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>$[-0.0040, 0.0019], 0.04$</td>
<td></td>
</tr>
<tr>
<td>0.04</td>
<td>$[-0.0018, 0.0034], 0.04$</td>
<td></td>
</tr>
</tbody>
</table>

Threshold probability = 0.85
Threshold probability = \(0.85\)

Error Metric Extraction

Return the maximum error with probability

\[
\begin{align*}
\text{Error, Probability: } & 0.0042, 0.86 \\
& \text{Cumulative Prob.} \\
& 0.85 \\
& 0.84 \\
& 0.04
\end{align*}
\]
## Results: Probabilistic Error Specification

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Worst Case Error (state-of-the-art)</th>
<th>Prob analysis + Prob subdiv (% reduction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sineOrder3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sqrt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bspline1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rigidbody2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>traincar2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>filter4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cubic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classIDX0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>polyIDX1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>neuron</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reduction % with 0.85 threshold probability for 32 bit floating-point errors, gaussian input distributions considering $4 \times \epsilon_m$ error happens with 0.1 probability.
Results: Probabilistic Error Specification

Reduction % with 0.85 threshold probability for 32 bit floating-point errors, gaussian input distributions considering $4 \times \epsilon_m$ error happens with 0.1 probability

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Worst Case Error (state-of-the-art)</th>
<th>Prob analysis + Prob subdiv (% reduction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sineOrder3</td>
<td>1.28E-06</td>
<td></td>
</tr>
<tr>
<td>sqrt</td>
<td>4.16E-04</td>
<td></td>
</tr>
<tr>
<td>bspline1</td>
<td>7.39E-07</td>
<td></td>
</tr>
<tr>
<td>rigidbody2</td>
<td>2.21E-02</td>
<td></td>
</tr>
<tr>
<td>traincar2</td>
<td>3.45E-03</td>
<td></td>
</tr>
<tr>
<td>filter4</td>
<td>3.81E-07</td>
<td></td>
</tr>
<tr>
<td>cubic</td>
<td>3.08E-06</td>
<td></td>
</tr>
<tr>
<td>classIDX0</td>
<td>9.25E-06</td>
<td></td>
</tr>
<tr>
<td>polyIDX1</td>
<td>2.18E-03</td>
<td></td>
</tr>
<tr>
<td>neuron</td>
<td>1.56E-04</td>
<td></td>
</tr>
</tbody>
</table>
### Results: Probabilistic Error Specification

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Worst Case Error (state-of-the-art)</th>
<th>Prob analysis + Prob subdiv (% reduction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sineOrder3</td>
<td>1.28E-06</td>
<td>-52.9</td>
</tr>
<tr>
<td>sqrt</td>
<td>4.16E-04</td>
<td>-56.6</td>
</tr>
<tr>
<td>bspline1</td>
<td>7.39E-07</td>
<td>-40.2</td>
</tr>
<tr>
<td>rigidbody2</td>
<td>2.21E-02</td>
<td>-13.5</td>
</tr>
<tr>
<td>traincar2</td>
<td>3.45E-03</td>
<td>-13.6</td>
</tr>
<tr>
<td>filter4</td>
<td>3.81E-07</td>
<td>-47.5</td>
</tr>
<tr>
<td>cubic</td>
<td>3.08E-06</td>
<td>-41.9</td>
</tr>
<tr>
<td>classIDX0</td>
<td>9.25E-06</td>
<td>-18.7</td>
</tr>
<tr>
<td>polyIDX1</td>
<td>2.18E-03</td>
<td>-10.6</td>
</tr>
<tr>
<td>neuron</td>
<td>1.56E-04</td>
<td>-41.7</td>
</tr>
</tbody>
</table>

Reduction % with 0.85 threshold probability for 32 bit floating-point errors, gaussian input distributions considering $4 \times e_m$ error happens with 0.1 probability.
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    return res
}

**Sound Analysis to compute**
**Wrong path probability**

**Sound Analysis to compute a precise bound on the error**

by taking into account the probability distribution of inputs
def func(x: Float32, y: Float32, z: Float32): Float32 = {
    require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    return res
}

Sound Analysis to compute Wrong path probability
by taking into account the probability distribution of inputs
Ranges and distributions were provided

Sound Analysis to compute a precise bound on the error
Ongoing Research: Scaling up

```python
def func(a: Float32, b: Float32, c: Float32): Float32 = {
    ...
    require (? <= x <= ? && ? <= y, z <= ?)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    ...
}
```

Goal: Compute the ranges automatically

Challenges:
- **Static Analysis** provides sound domain bounds, does not scale
- **Dynamic Analysis** scales for real-world programs, not sound
def func(a: Float32, b: Float32, c: Float32): Float32 = {
    ...
    require (? <= x <= ? && ? <= y, z <= ?)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    if (res <= 0.0)
        raiseAlarm()
    else
        doNothing()
    ...
}

Our Idea: Combine them to compute the ranges automatically

More ideas?