Detecting Floating-Point Errors via Chain Conditions

Liqian Chen

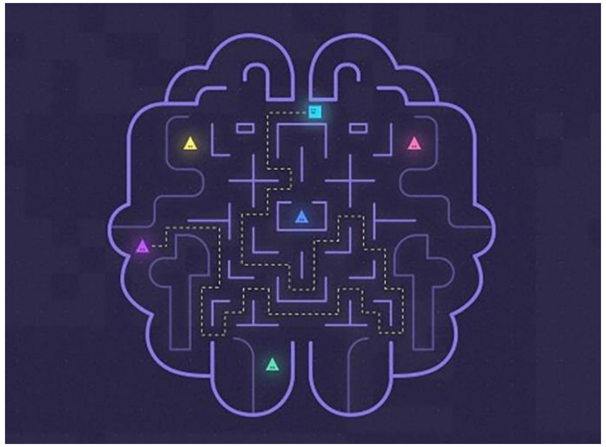
National University of Defense Technology, China

(Joint work with Xin Yi, Hengbiao Yu, Xiaoguang Mao, Ji Wang)

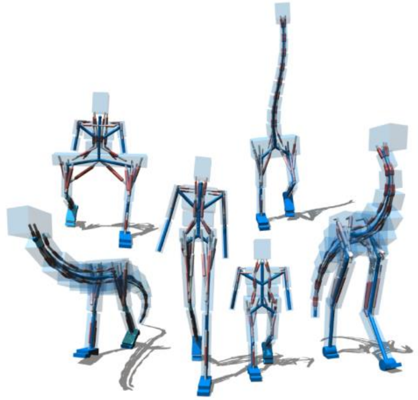
Overview

- Motivation
- Approach
- Experiment
- Conclusion

Wide use of numerical libraries

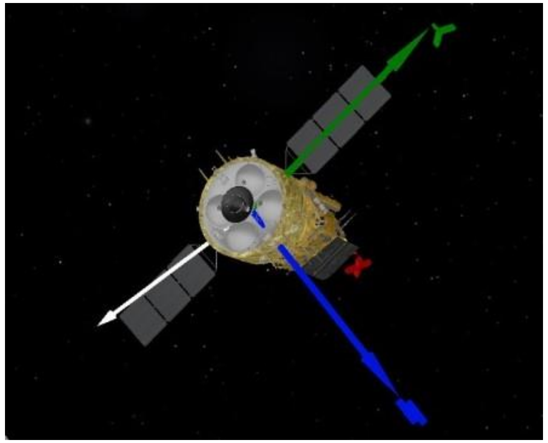


Machine learning



Physical simulation

Statistical analysis



Control software

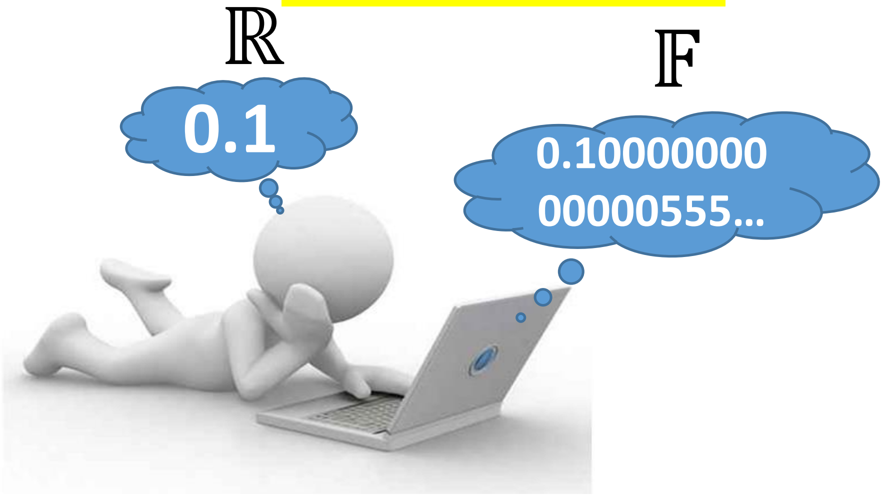
Numerical libraries: math.h, GSL, NumPy, SciPy…

Floating-point arithmetic

Inexactness of floating-point (FP)

$\mathbb{R}$		$\mathbb{F}$
f	$\neq$ →	f_p
Mathematical function	∃ floating-point errors	Numerical program

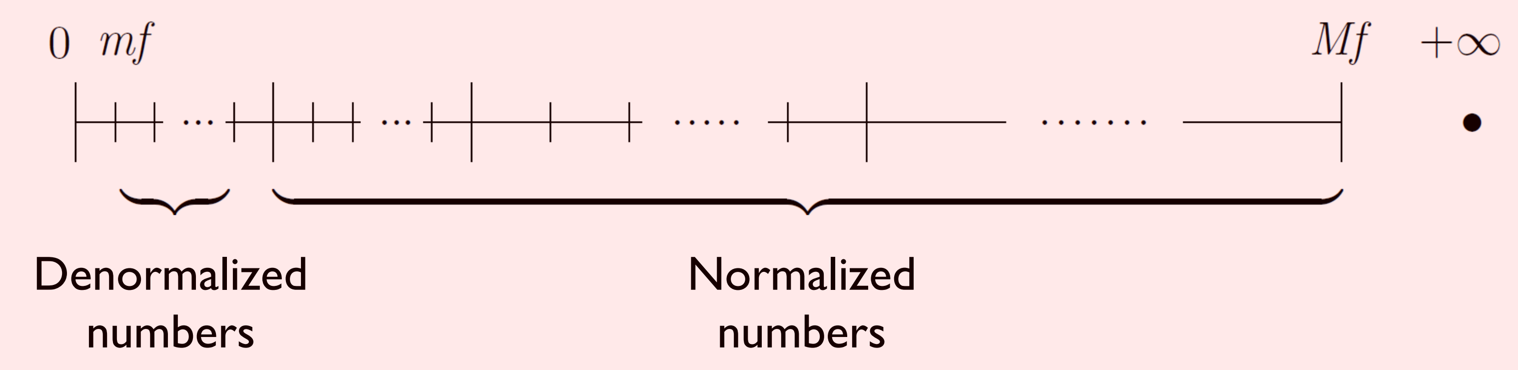
Rounding error




```
>>> a = 0.1; b = 0.2; c = a + b;
>>> e = 0.3;
>>> print(c == e);
False
>>>
```

Pitfalls of floating-point computation

- Nonuniform distribution of floating-point numbers



- Known algebraic properties (such as associativity and distributivity) over the reals do not hold for floating-point arithmetic

```
>>> (1e20+1)-1e20
0.0
>>> (1e20-1e20)+1
1.0
>>>
```

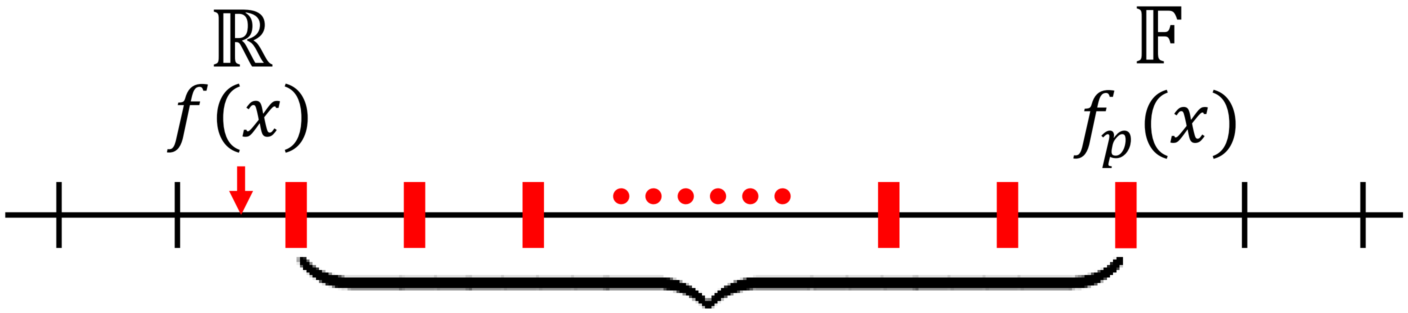
$$(2^{24} \oplus_{32,?} -2^{24}) \oplus_{32,?} 1 = 1$$

$$(2^{24} \oplus_{32,-\infty} 1) \oplus_{32,-\infty} -2^{24} = 0$$

$$(2^{24} \oplus_{32,+\infty} 1) \oplus_{32,+\infty} -2^{24} = 2$$

Floating-point error

- Floating-point error



n ULPs (Unit in the Last Place)

$$Err_{\text{abs}}(f(x), f_p(x)) = |f(x), f_p(x)|$$

$$Err_{rel}(f(x), f_p(x)) = \left|\frac{f(x) - f_p(x)}{f(x)}\right|$$

- High floating-point error

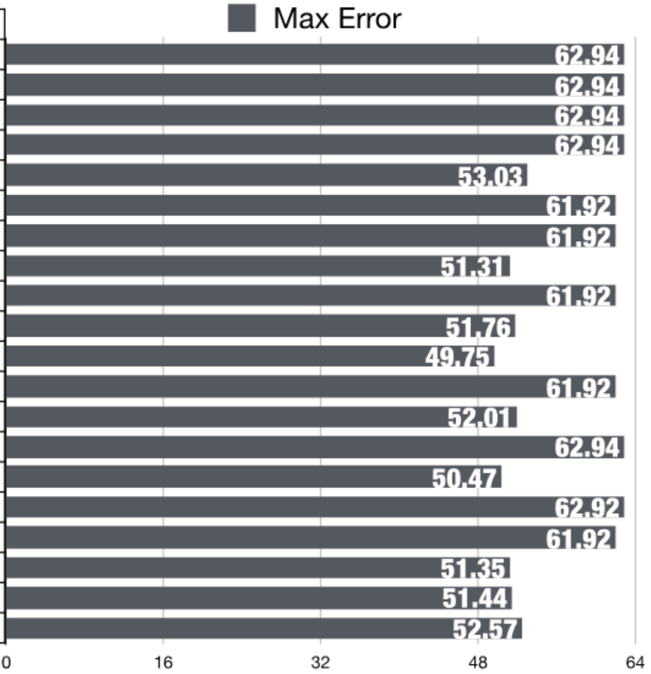
$$Err(f(x), f_p(x)) > \varepsilon$$

Error threshold

High FP errors in numerical libraries

- Programs in numerical libraries
  - Expert code
  - Well maintained
- High FP errors may still exist in numerical libraries
  - E.g., caused by ill-conditioned problems which are in the nature of the mathematical feature of many functions in numerical libraries

Program ID	Benchmark
P1	gsl_sf_airy_Ai
P2	gsl_sf_airy_Bi
P3	gsl_sf_airy_Ai_deriv
P4	gsl_sf_airy_Bi_deriv
P5	gsl_sf_bessel_J0
P6	gsl_sf_bessel_J1
P7	gsl_sf_bessel_Y0
P8	gsl_sf_bessel_Y1
P9	gsl_sf_clausen
P10	gsl_sf_expint_Ei
P11	gsl_sf_legendre_P2
P12	gsl_sf_legendre_P3
P13	gsl_sf_legendre_Q1
P14	gsl_sf_psi
P15	gsl_sf_Chi
P16	gsl_sf_Ci
P17	gsl_sf_lnsinh
P18	gsl_sf_zeta
P19	gsl_sf_eta
P20	gsl_sf_psi_1



[4]Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. Efficient automated repair of high floating-point errors in numerical libraries. POPL'19

Our goal in this talk

Goal: Automatically finding and localizing high floating-point errors

Finding ➡ Localizing

high floating-point errors

The literature on floating-point error detection methods

BGRT[PPoPP14]
- Binary Search
- Blackbox+MPFR
- Multiple inputs (NPB)

DEMC[POPL19]
- Global Condition+MCMC
- Blackbox+mpmath
- Single Input (GSL)

FPGen[ICSE20]
- Symbolic excution
- WhiteBox+long double
- Multiple inputs (GSL+MM)

Relying in expensive high precision computation

… 2014 2015 2017 2019 2020 2024

LSGA[ICSE15]
- Genetic Alg.
- Blackbox+MPFR
- Single Input (GSL)

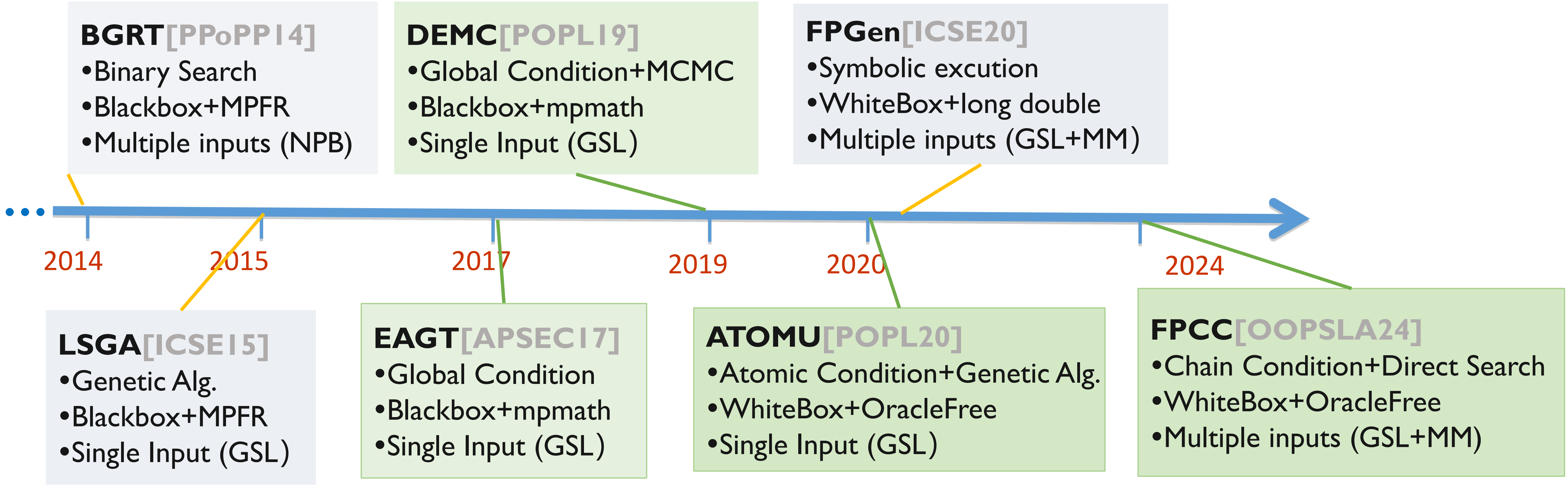
EAGT[APSEC17]
- Global Condition
- Blackbox+mpmath
- Single Input (GSL)

ATOMU[POPL20]
- Atomic Condition+Genetic Alg.
- WhiteBox+OracleFree
- Single Input (GSL)

Prone to false positives

[1]Wei-Fan Chiang, Ganesh Gopalakrishnan, Zvonimir Rakamaric, and Alexey Solovyev. Efficient search for inputs causing high floating-point errors. PPoPP'14
[2]Daming Zou, Ran Wang, Yingfei Xiong, Lu Zhang, Zhendong Su, and Hong Mei. A Genetic Algorithm for Detecting Significant Floating-Point Inaccuracies. ICSE'15
[3]Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. Efficient Global Search for Inputs Triggering High Floating-Point Inaccuracies. APSEC'17
[4]Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. Efficient automated repair of high floating-point errors in numerical libraries. POPL'19
[5]Hui Guo and Cindy Rubio-González. Efficient generation of error-inducing floating-point inputs via symbolic execution. ICSE'20.
[6]Daming Zou, Muhan Zeng, Yingfei Xiong, Zhoulai Fu, Lu Zhang, and Zhendong Su. Detecting floating-point errors via atomic conditions. POPL'20.

The literature on floating-point error detection methods

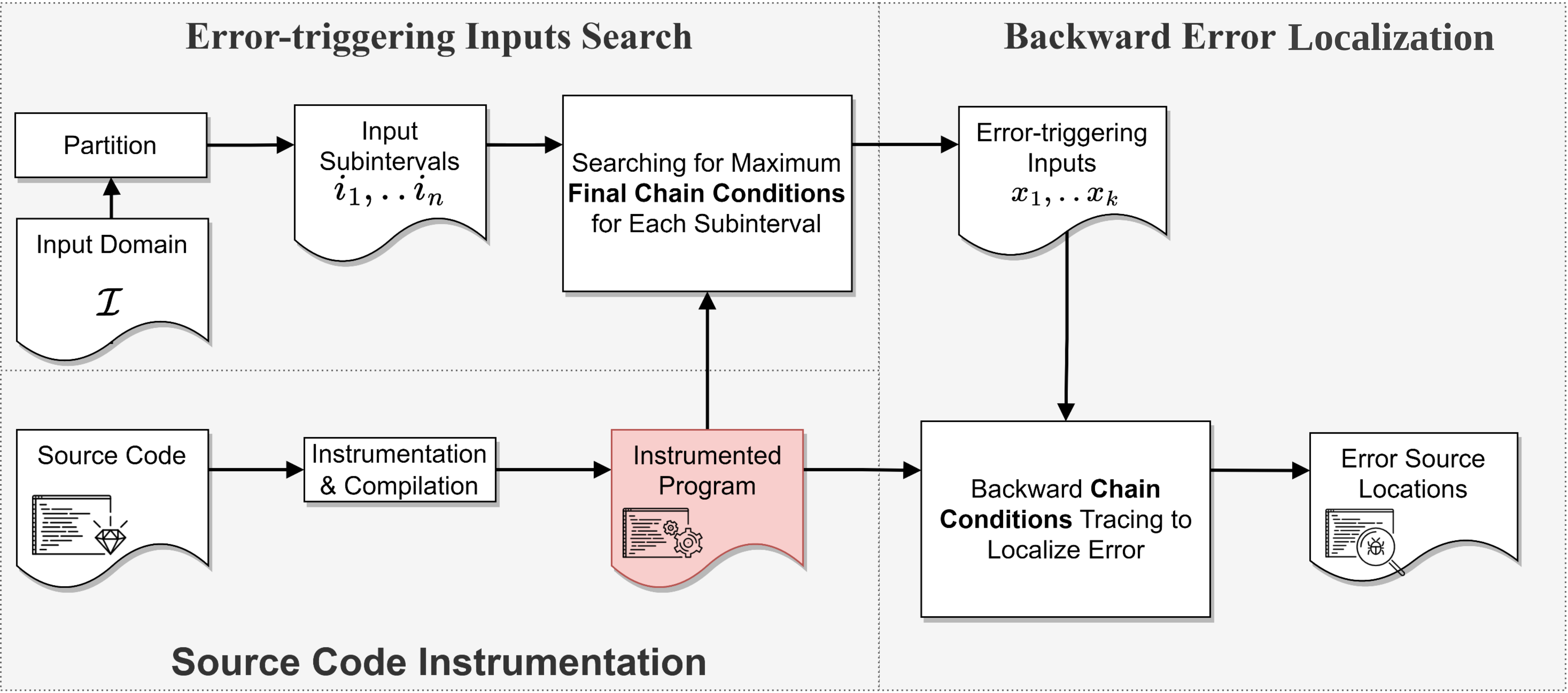


[1]Wei-Fan Chiang, Ganesh Gopalakrishnan, Zvonimir Rakamaric, and Alexey Solovyev. Efficient search for inputs causing high floating-point errors. PPoPP'14
[2]Daming Zou, Ran Wang, Yingfei Xiong, Lu Zhang, Zhendong Su, and Hong Mei. A Genetic Algorithm for Detecting Significant Floating-Point Inaccuracies. ICSE'15
[3]Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. Efficient Global Search for Inputs Triggering High Floating-Point Inaccuracies. APSEC'17
[4]Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. Efficient automated repair of high floating-point errors in numerical libraries. POPL'19
[5]Hui Guo and Cindy Rubio-González. Efficient generation of error-inducing floating-point inputs via symbolic execution. ICSE'20.
[6]Daming Zou, Muhan Zeng, Yingfei Xiong, Zhoulai Fu, Lu Zhang, and Zhendong Su. Detecting floating-point errors via atomic conditions. POPL'20.

Overview

- Motivation
- Approach
- Experiment
- Conclusion

Work-flow


Error-triggering Inputs Search
Backward Error Localization
Partition
Input Subintervals $i_1, .. i_n$
Searching for Maximum **Final Chain Conditions** for Each Subinterval
Error-triggering Inputs $x_1, .. x_k$
Input Domain $\mathcal{I}$
Source Code
Instrumentation & Compilation
Instrumented Program
Backward **Chain Conditions** Tracing to Localize Error
Error Source Locations
Source Code Instrumentation

Concepts — Condition number

- A function's condition number measures its sensitivity to small perturbation of the input

Relative error for unary-input operation

$$\left|\frac{f(x+\Delta x)-f(x)}{f(x)}\right| \approx \left|\frac{f'(x)\Delta x}{f(x)}\right|$$

$$\approx \left|\frac{xf'(x)}{f(x)}\right| \cdot \left|\frac{\Delta x}{x}\right|$$

Condition number $C_f(x) = \left|\frac{xf'(x)}{f(x)}\right|$

Concepts — Condition number

- A function's condition number measures its sensitivity to small perturbation of the input

Relative error for binary-input operation $op(x,y)$

$$
\begin{aligned}
\left|\frac{f(x,y)-f(x+\Delta x, y+\Delta y)}{f(x,y)}\right| &= \left|\frac{f(x,y)-f(x,y+\Delta y)+f(x,y+\Delta y)-f(x+\Delta x, y+\Delta y)}{f(x,y)}\right| \\
&= \left|\frac{f(x,y)-f(x,y+\Delta y)}{f(x,y)}\right| + \left|\frac{f(x,y+\Delta y)-f(x+\Delta x, y+\Delta y)}{f(x,y)}\right| \\
&\approx C_{f,y}(x,y)\cdot\left|\frac{\Delta y}{y}\right| + C_{f,x}(x,y+\Delta y)\cdot\left|\frac{\Delta x}{x}\right| \\
&\approx C_{f,y}(x,y)\cdot\left|\frac{\Delta y}{y}\right| + C_{f,x}(x,y)\cdot\left|\frac{\Delta x}{x}\right|
\end{aligned}
$$

Concepts — Condition number

- A function's condition number measures its sensitivity to small perturbation of the input

$$C_f(x) = \left| \frac{x f'(x)}{f(x)} \right|$$

$$\left| \frac{f(x,y) - f(x+\Delta x, y+\Delta y)}{f(x,y)} \right| \approx C_{f,y}(x,y) \cdot \left| \frac{\Delta y}{y} \right| + C_{f,x}(x,y) \cdot \left| \frac{\Delta x}{x} \right|$$

Example

$$op(x,y) = x + y$$

$$C_+(x) = \left| \frac{x}{x+y} \right| \qquad C_+(y) = \left| \frac{y}{x+y} \right|$$

Concepts — Atomic condition

- Atomic condition[6]: the condition number of an atomic floating-point operation
- ATOMU[6]: atomic condition-guided search to find error-inducing inputs
  - Pros
    - Oracle-free, …
  - Cons
    - Prone to false positives: an operation triggering a large value of atomic condition may be suppressed by the later operation, resulting in a small final relative error

[6] Daming Zou, Muhan Zeng, Yingfei Xiong, Zhoulai Fu, Lu Zhang, and Zhendong Su. Detecting floating-point errors via atomic conditions. POPL'20

Concepts — Chain condition

- Given an operation sequence $\langle op_0, \ldots, op_i, \ldots, op_n \rangle (0 \le i \le n)$, operation op_i's chain condition CC_{op_i} evaluates how the input floating-point errors are amplified by the operation sequence $\langle op_0, \ldots, op_i \rangle$

How to calculate the chain condition of an operation sequence?

Calculating chain conditions

- Calculation rules for chain conditions

$$\frac{y = op_j(x) \wedge \nexists op_i \rightsquigarrow op_j}{CC_{op_j} = C_{op_j}(x)} \quad \text{(Init-1)}$$

$$\frac{z = op_j(x, y) \wedge \nexists op_i \rightsquigarrow op_j}{CC_{op_j} = C_{op_j}(x) + C_{op_j}(y)} \quad \text{(Init-2)}$$

$$\frac{y = op_i(x, \dots) \wedge z = op_j(y) \wedge op_i \rightsquigarrow op_j}{CC_{op_j} = CC_{op_i} \cdot C_{op_j}(y)} \quad \text{(Unary)}$$

$$\frac{y_1 = op_i(x_1, \dots) \wedge z = op_k(y_1, y_2) \wedge op_i \rightsquigarrow op_k \wedge \nexists j \neq i, op_j \rightsquigarrow op_k}{CC_{op_k} = CC_{op_i} \cdot C_{op_k}(y_1) + C_{op_k}(y_2)} \quad \text{(Binary-1)}$$

$$\frac{y_1 = op_i(x_1, \dots) \wedge y_2 = op_j(x_2, \dots) \wedge z = op_k(y_1, y_2) \wedge op_i \rightsquigarrow op_k \wedge op_j \rightsquigarrow op_k}{CC_{op_k} = CC_{op_i} \cdot C_{op_k}(y_1) + CC_{op_j} \cdot C_{op_k}(y_2)} \quad \text{(Binary-2)}$$

Calculating chain conditions

- Calculation rules for chain conditions

Example

$$\frac{y = op_i(x, \dots) \wedge z = op_j(y) \wedge op_i \rightsquigarrow op_j}{CC_{op_j} = CC_{op_i} \cdot C_{op_j}(y)} \quad \text{(Unary)}$$

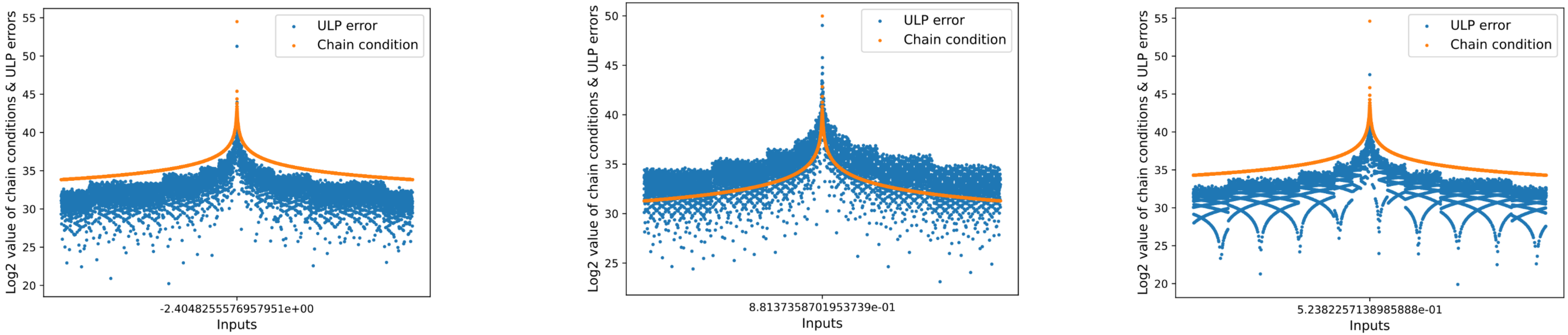
$$y = op_1(x_1, x_2); z = op_2(y)$$

Proof:

$$\begin{aligned} \varepsilon_y &\approx C_{op_1}(x_1) \cdot \varepsilon_{x_1} + C_{op_1}(x_2) \cdot \varepsilon_{x_2} \\ &\approx \varepsilon_{\bar{x}} \cdot (C_{op_1}(x_1) + C_{op_1}(x_2)) \\ &= \varepsilon_{\bar{x}} \cdot CC_{op_1} \qquad \text{(Init-2)} \\ \varepsilon_z &\approx C_{op_2}(y) \cdot \varepsilon_y \\ &\approx \varepsilon_{\bar{x}} \cdot CC_{op_1} \cdot C_{op_2}(y) \end{aligned}$$

Chain condition-guided global search

- **Observations**
  - There exists a notable consistency between the distribution of final chain conditions and the distribution of floating-point errors
  - The distribution of final chain conditions exhibits a clear trend of gradual increase



(a) gsl_sf_bessel_J0 (b) gsl_sf_lnsinh (c) gsl_sf_Chi

Distributions of chain conditions and ULP errors w.r.t. inputs

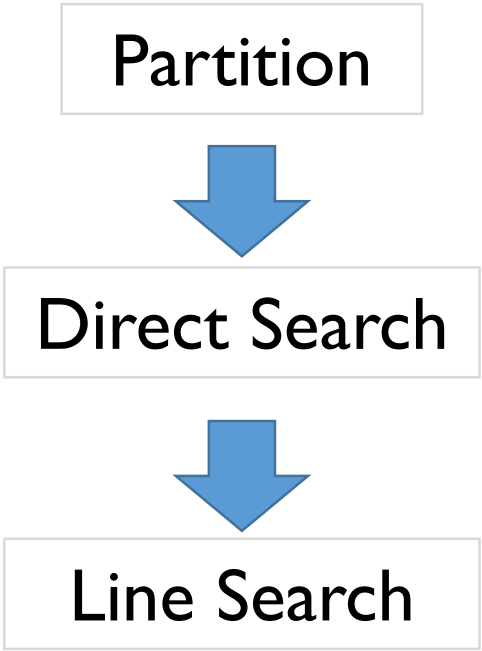
Chain condition-guided global search

- Search algorithm

Algorithm 1: Chain Condition-Guided Global Search

input : An instrumented floating-point program $\mathcal{P}$ and an input domain $\mathcal{I}$
output: A list $\mathcal{X}$ for the inputs that trigger large chain conditions

$I_s \leftarrow partition(\mathcal{I})$
$CC_l \leftarrow \emptyset$
$TempCC_l \leftarrow \emptyset$
for $i \in I_s$ **do**
  $(x_i, cc_i) \leftarrow DirectSearch(i, \mathcal{P})$
  $TempCC_l.append([x_i, cc_i])$
end
Sort($TempCC_l$)
$k \leftarrow 0$
for $tc \in TempCC_l$ **do**
  $x_k \leftarrow tc.x_k$
  $cc_k \leftarrow tc.cc_k$
  if $k < limit$ **then**
    $(x_k, cc_k) \leftarrow LineSearch(x_k, , \mathcal{P})$
  end
  if $CC_k > threadhold$ **then**
    $CC_l.append([x_k, cc_k])$
  end
  $k \leftarrow k + 1$
end
Sort(CC_l)
$X \leftarrow GetInputs(CC_l)$
return X

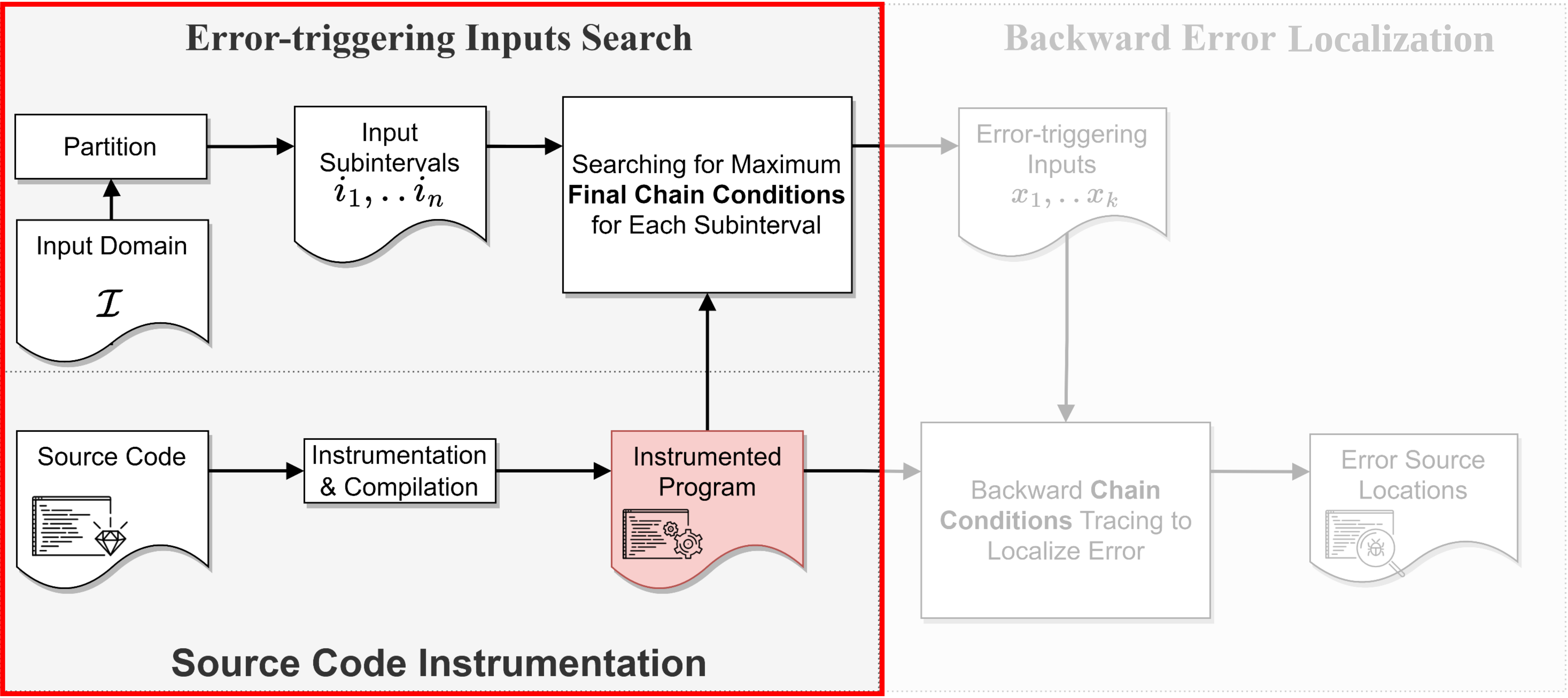


Chain condition-guided global search

Example $P_1(x) = 0.375 + (a - x * (0.25 + b))$ $P_2(x) = 0.375 * (a - x * (0.25 + b))$

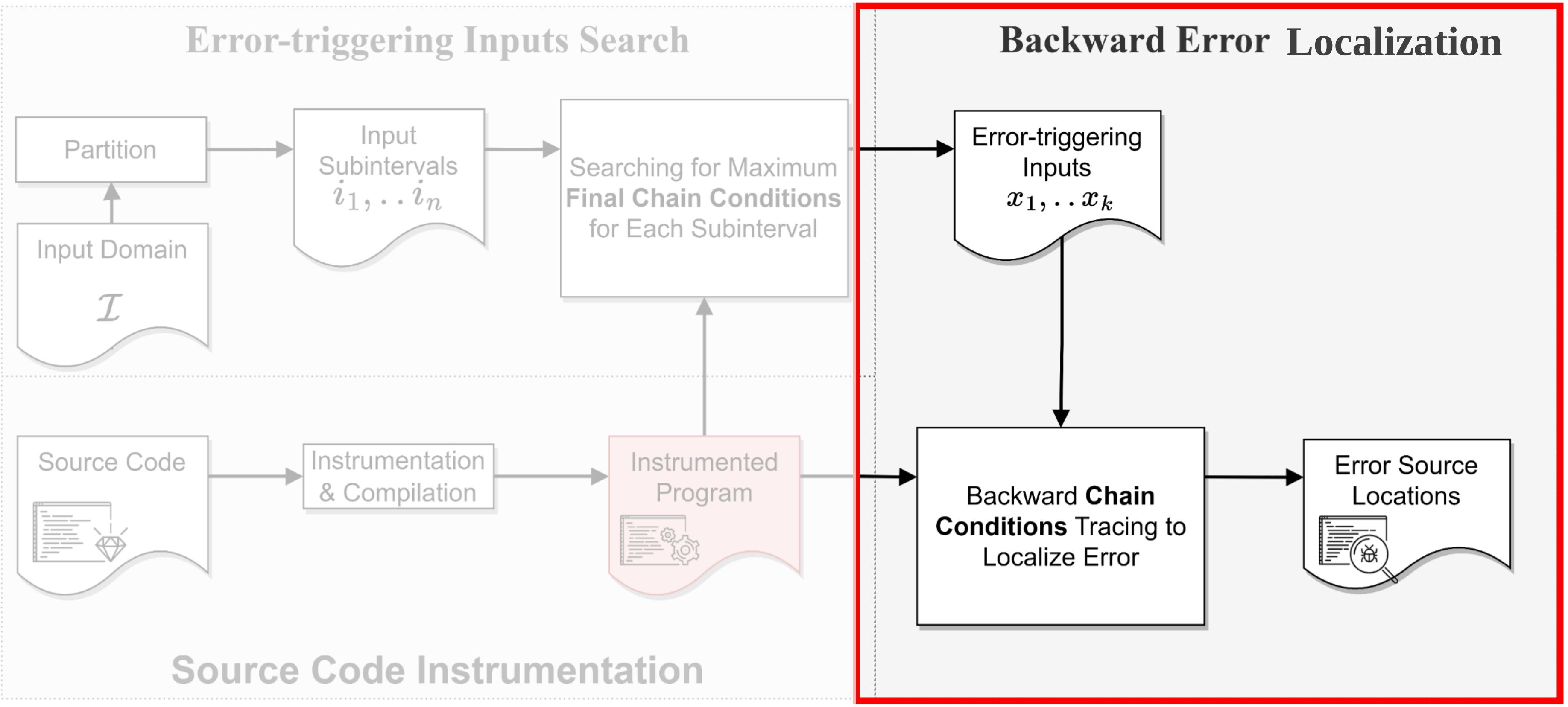
$P_1(x)$	Operand(s)	Chain condition	Atomic condition	Operation result	Relative error
v1 = 0.25+b	0.25, 0.008809451767620686893 4	$C_+(0.25) = 0.966$, $C_+(b) = 0.034$, $CC(v1) = 1.0$	1.0	0.25880945176762071291	1.01e-16
v2 = x*v1	-0.07727402791033162 5, 0.25880945176762071291	$C_*(x) = 1.0$, $CC(v1) * C_*(v1) = 1.0$, $CC(v2) = 2.0$	2.0	-0.01999924879934875110 4	1.86e-16
v3 = a-v2	-0.01999924879934875804 3, -0.01999924879934875110 4	$C_-(a) = 2.88e{+}15$, $CC(v2) * C_-(v2) = 5.76e{+}15$, $CC(v3) = 8.64e{+}15$	5.76e+15	-6.938893903907228377 6e-18	3.49e-1
v4 = 0.375+v3	0.375, -6.938893903907228377 6e-18	$CC_-(0.375) = 1.0$, $CC(v3) * C_-(v3) = 0.16$, $CC(v4) = 1.16$	1.0	0.375	2.84e-17
return v4					
$P_2(x)$					
v5 = 0.375*v3	0.375, -6.938893903907228377 6e-18	$CC_*(0.375) = 1.0$, $CC(v3) * C_*(v3) = 8.64e{+}15$, $CC(v5) = 8.64e{+}15$	1.0	2.6020852139652106e-18	3.49e-1
return v5					

Chain condition-guided global search



Chain condition-based error localization

- Goal: to localize the (root-cause) source code of FP errors
- Method: backward tracing of chain conditions to identify FP operations that introduce large chain conditions and propagate to the output



Approach: Chain condition-based error localization

- Algorithm

Algorithm 2: Chain Condition-Based Error Localization Algorithm

Input: $\mathcal{P}$:an instrumented floating-point program; x: an error-triggering input
Output: $SubSeq$: the sequence of localized source code.

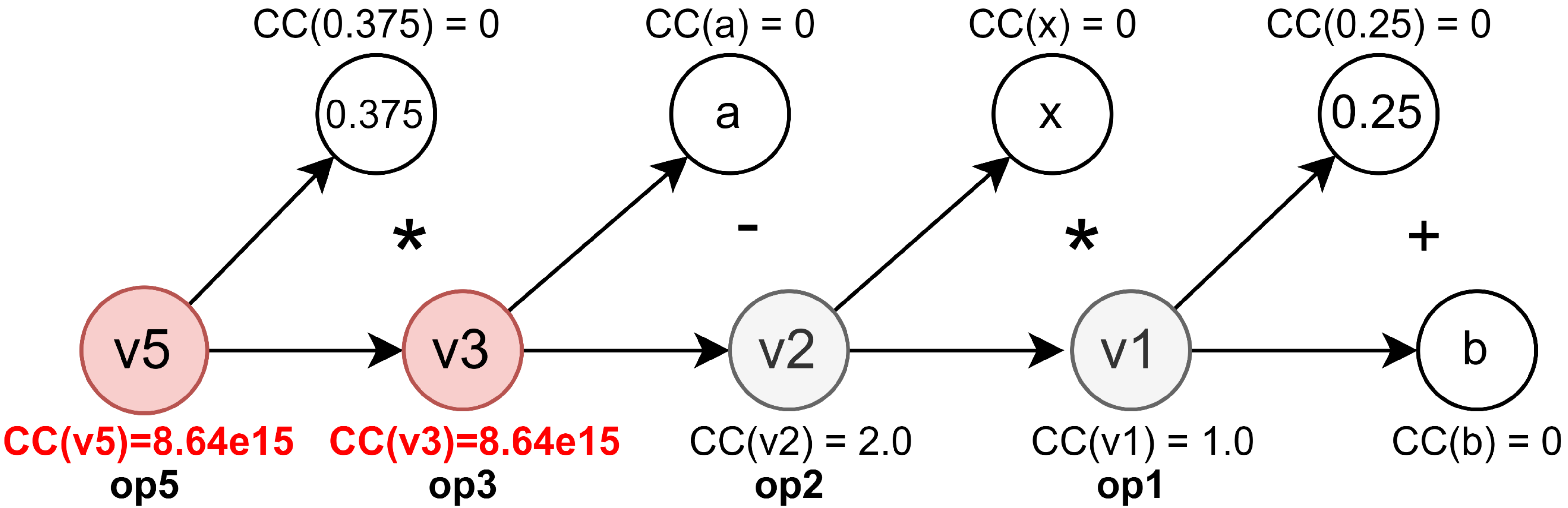
Seq = $\{st_0, \ldots, st_n\} \leftarrow \mathcal{P}(x)$
$SubSeq \leftarrow \emptyset$
BackwardErrorTrace($st_n, Seq, SubSeq$)
return $SubSeq$
Function BackwardErrorTrace($st_n, Seq, SubSeq$):
    $st_i, st_j \leftarrow$ GetPredecessors(Seq, st_n)
    $SubSeq \leftarrow st_n$
    if $CC(st_i) > CC(st_n)/\omega$ **then**
        BackwardErrorTrace($Seq, st_i, SubSeq$)
    end
    if $CC(st_j) > CC(st_n)/\omega$ **then**
        BackwardErrorTrace($Seq, st_j, SubSeq$)
    end
    return 0
End Function

chain condition value exceeding threshold ($CC(st_n)/\omega$)

Approach: Chain condition-based error localization

- Example：

v1 = 0.25+b; v2 = x*v1; v3 = a-v2; v4 = 0.375+v3; v5 = 0.375*v3;
//i.e., P(x) = 0.375 $*$ ($a - x *$ (0.25 + b))

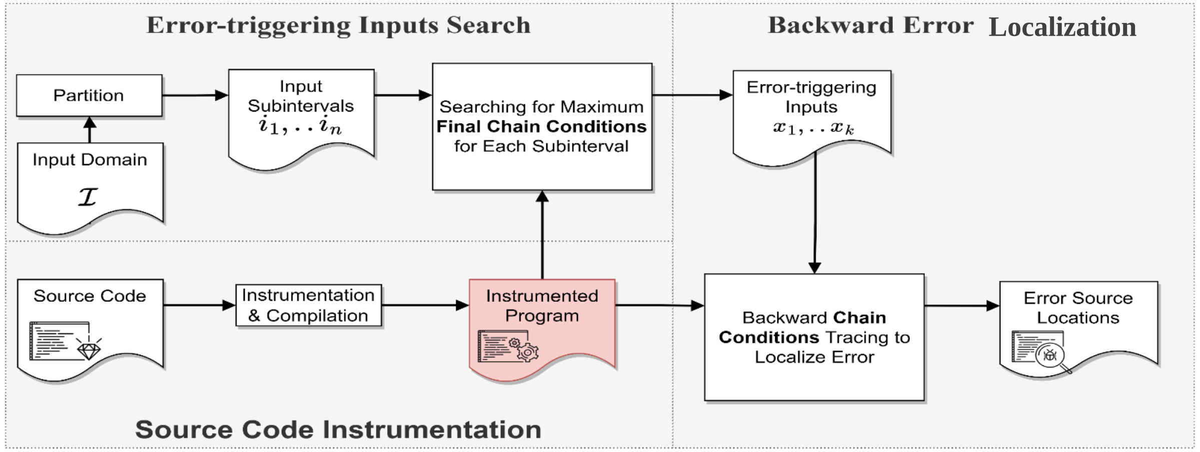


resulting {op5, op3}

Overview

- Motivation
- Approach
- Experiment
- Conclusion

Implementation and Evaluation



- Implementation
  - Tool: FPCC (https://github.com/DataReportRe/FPCC)
- Benchmarks
  - 88 univariate functions from GSL's special functions
  - 21 multiple inputs functions from FPGen[5]

[5]Hui Guo and Cindy Rubio-González. Efficient generation of error-inducing floating-point inputs via symbolic execution. ICSE'20.

Evaluation

- RQ1: How effective is FPCC in detecting functions with significant errors?

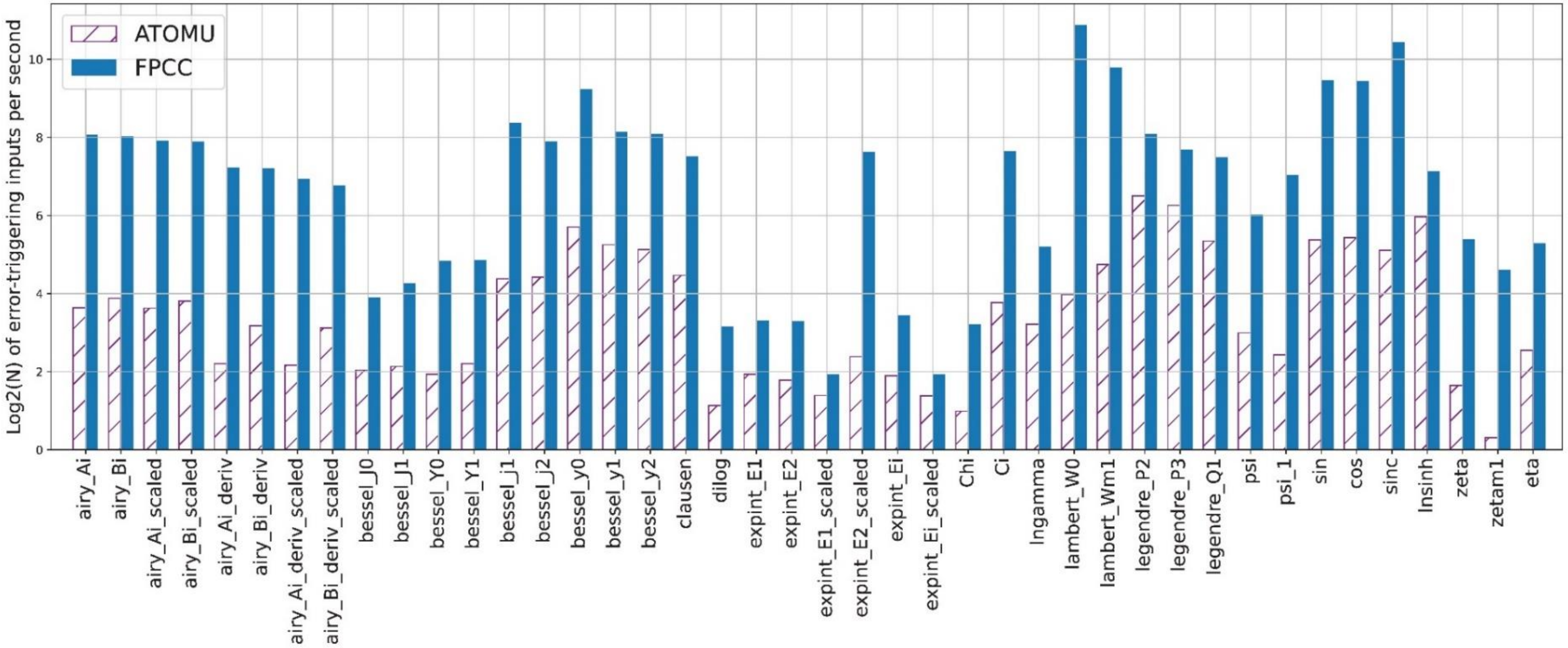
GSLfunctions	Pct. of Rank-1 Inputs		Error-triggering Inputs		Max RelErr		Max ULPErr		Time		Speedup
	FPCC	ATOMU	FPCC	ATOMU	FPCC	ATOMU	FPCC	ATOMU	FPCC	ATOMU	
airy_Ai	100%	0%	58/58	7/26	1.38E+298	2.91E+04	63.54	62.72	0.217	0.562	2.59
airy_Bi	100%	5%	58/58	8/27	1.35E+302	1.12E+10	63.55	62.99	0.221	0.538	2.43
airy_Ai_scaled	100%	0%	58/58	7/26	1.38E+298	1.53E+04	63.54	62.79	0.239	0.567	2.37
airy_Bi_scaled	100%	1%	58/58	8/27	1.35E+302	5.81E+09	63.55	62.99	0.245	0.561	2.29
airy_Ai_deriv	100%	100%	10/10	1/16	4.34E+00	3.46E-02	63.22	46.88	0.067	0.216	3.23
airy_Bi_deriv	100%	100%	10/10	2/16	2.58E+00	8.03E-01	63.20	50.03	0.068	0.222	3.27
airy_Ai_deriv_scaled	100%	100%	10/10	1/15	4.34E+00	4.02E-02	63.22	47.05	0.082	0.222	2.72
airy_Bi_deriv_scaled	100%	100%	10/10	2/15	2.58E+00	3.23E-01	63.20	49.59	0.092	0.228	2.49
bessel_J0	100%	100%	6/6	2/14	3.90E-01	1.65E-01	51.26	49.04	0.402	0.419	1.04
bessel_J1	100%	100%	8/8	2/15	1.87E-01	1.04E-01	50.52	48.68	0.417	0.422	1.01
bessel_Y0	100%	100%	5/5	2/24	2.86E+00	1.84E-01	61.92	49.46	0.176	0.392	2.23
bessel_Y1	100%	89%	5/5	2/24	1.17E-01	8.68E-02	49.43	48.56	0.172	0.401	2.34
bessel_j1	100%	81%	4/4	1/3	3.10E-02	3.38E-02	47.80	42.55	0.012	0.039	3.23
bessel_j2	100%	91%	4/6	1/4	3.61E-01	7.74E-02	51.03	45.94	0.017	0.042	2.53
bessel_y0	100%	100%	46/47	7/14	8.32E+119	1.36E+04	63.13	62.72	0.077	0.132	1.72
bessel_y1	100%	100%	48/48	13/23	9.29E+114	2.51E+09	63.14	62.92	0.170	0.334	1.96
bessel_y2	100%	0%	46/47	13/25	8.32E+119	8.44E+10	63.13	62.92	0.170	0.360	2.12
clausen	100%	100%	18/18	3/11	6.60E-01	1.01E+00	52.66	57.31	0.098	0.143	1.46
dilog	100%	36%	1/1	0/10	5.52E-01	3.24E-01	52.16	20.16	0.112	0.165	1.48
expint_E1	100%	100%	1/1	1/16	4.58E-01	1.81E-01	51.76	49.23	0.101	0.261	2.58
expint_E2	100%	100%	1/1	1/17	2.65E+02	6.17E+01	62.92	56.18	0.102	0.290	2.85
expint_E1_scaled	100%	99%	1/1	1/16	4.58E-01	1.57E-01	51.84	48.69	0.263	0.378	1.44
expint_E2_scaled	100%	100%	58/58	2/17	7.76E+291	3.40E+288	62.92	62.54	0.295	0.383	1.30
expint_Ei	100%	100%	1/1	1/16	4.58E-01	1.71E-01	51.76	49.07	0.092	0.268	2.91
expint_Ei_scaled	100%	100%	1/1	1/16	4.58E-01	1.49E-01	51.84	49.07	0.261	0.386	1.48
Chi	100%	100%	2/2	1/17	4.40E-02	8.57E-02	47.56	48.39	0.216	0.504	2.34
Ci	100%	100%	49/49	13/36	9.29E+114	2.55E+08	63.14	62.58	0.245	0.931	3.80
lngamma	100%	0%	2/2	2/21	1.35E+00	3.98E+00	62.93	53.02	0.054	0.170	3.14
lambert_W0	100%	50%	62/62	1/8	1.00E+00	3.51E-01	61.76	50.49	0.033	0.063	1.91
lambert_Wm1	100%	99%	31/31	2/9	1.00E+00	8.94E-01	61.76	58.78	0.035	0.069	1.95
legendre_P2	100%	100%	2/2	1/1	4.19E-02	7.55E-02	47.48	47.95	0.007	0.011	1.51
legendre_P3	100%	100%	2/2	1/1	1.00E+00	1.14E-01	61.92	48.64	0.010	0.013	1.34
legendre_Q1	100%	100%	2/2	1/5	5.03E-01	1.32E-01	52.01	48.38	0.011	0.025	2.20
psi	100%	100%	12/12	3/19	3.49E-01	5.06E+00	51.48	51.08	0.184	0.355	1.93
psi_1	100%	0%	11/11	1/7	5.76E-01	1.07E-01	52.38	47.99	0.084	0.179	2.14
sin	100%	100%	80/80	7/14	9.29E+114	2.53E+10	63.25	62.89	0.114	0.160	1.41
cos	100%	100%	78/78	7/14	8.32E+119	1.43E+04	63.24	62.87	0.113	0.157	1.40
sinc	100%	100%	174/174	8/16	1.00E+00	1.00E+00	62.36	62.22	0.126	0.227	1.81
lnsinh	100%	100%	1/1	1/2	1.13E-01	1.65E-01	49.03	49.36	0.007	0.016	2.25
zeta	100%	0%	6/6	2/38	6.24E-01	2.39E-02	51.35	45.98	0.144	0.589	4.10
zetam1	100%	2%	3/3	1/42	5.65E-02	6.27E-03	48.15	43.09	0.124	0.646	5.22
eta	100%	0%	6/6	4/43	6.24E-01	1.84E-02	51.33	46.23	0.154	0.624	4.06
Summary	100%	72.69%	1049/1053 99.62%	141/723 19.45%					0.139	0.302	2.17

- FPCC achieves 100% accuracy in detecting significant errors for the reported rank-1 inputs, while ATOMU[6] achieves 72.7% accuracy for its rank-1 inputs
- When considering all the reported inputs, FPCC identifies errors in 99.62% (1049/1053) of its reported inputs, whereas ATOMU reports errors for 19.45% (141/723)

[6] Daming Zou, Muhan Zeng, Yingfei Xiong, Zhoulai Fu, Lu Zhang, and Zhendong Su. Detecting floating-point errors via atomic conditions. POPL'20

Evaluation

- RQ2: How efficient is FPCC in detecting functions with significant errors?



- FPCC exhibits 2.17x speedup over ATOMU[6] in detecting significant FP errors
- FPCC achieves13.47x speedup over ATOMU in terms of the number of error-triggering inputs per second.

[6] Daming Zou, Muhan Zeng, Yingfei Xiong, Zhoulai Fu, Lu Zhang, and Zhendong Su. Detecting floating-point errors via atomic conditions. POPL'20

Evaluation

- RQ3: How scalable is FPCC?

Benchmarks	FP Params	Relative Error		Time(s)	
		FPCC	FPGen	FPCC	FPGen
recursive_summation	32	**9.00E+00**	1.00E+00	100	7200
pairwise_summation	32	**3.00E+00**	1.32E-16	100	7200
compensated_summation	32	**1.00E+00**	1.00E+00	100	7200
sum	4	**1.00E+00**	1.00E+00	100	7200
2norm	4	**1.76E-16**	0.00E+00	100	7200
1norm	4	1.57E-16	2.21E-16	100	7200
dot	8	**1.10E+00**	1.92E-04	100	7200
conv	8	**3.07E+00**	2.04E-04	100	7200
mv	20	**1.16E+00**	8.94E-04	100	7200
mm	32	**2.30E-05**	2.58E-14	100	7200
LU	16	**1.04E+00**	2.73E+00	100	7200
QR	16	**1.00E+00**	2.59E-14	100	7200
wmean	8	**8.52E+01**	1.00E+00	100	7200
wvariance_m	8	**6.81E-01**	7.63E-02	100	7200
wvariance_w	8	**8.71E-01**	2.85E-12	100	7200
wsd_m	8	**3.22E-01**	3.74E-02	100	7200
wsd_w	8	**6.49E-07**	1.14E-12	100	7200
wtss_m	8	**9.24E-01**	4.45E-16	100	7200
wabsdev_m	8	**4.94E-01**	1.00E+00	100	7200
wkurtosis_m	8	**8.19E+00**	2.57E+01	100	7200
wkew_m	8	**7.05E+00**	1.77E-12	100	7200

FPCC vs FPGen over 21 functions with multiple inputs

- For multiple-input benchmarks, FPCC identifies more significant errors than FPGen[5] in the majority of cases

[5]Hui Guo and Cindy Rubio-González. Efficient generation of error-inducing floating-point inputs via symbolic execution. ICSE'20.

Overview

- Motivation
- Approach
- Experiment
- Conclusion

Summary

- **Approach**: introducing chain conditions to capture the propagation of floating-point errors and to guide the search for error-inducing inputs

Calculating chain conditions → Detecting high FP errors → Localizing source code of FP errors

  - **Advantages**:
    - Oracle-free
    - Support multiple-input functions
    - Low rate of false positives

Summary

- **Approach**: introducing chain conditions to capture the propagation of floating-point errors and to guide the search for error-inducing inputs

Calculating chain conditions → Detecting high FP errors → Localizing source code of FP errors

- **Tool**:
  - FPCC

https://github.com/DataReportRe/FPCC

- **Experiments**:
  - 88 univariate functions from GSL and 21 multiple-input functions
  - 99.64% (vs. 19.45%) of the inputs reported by FPCC (vs. ATOMU) can trigger significant errors

Thank you!
Any questions?