ARFA

An Agile Regime-Based Floating-Point Optimization Approach for Rounding Errors

Jinchen Xu [1], Mengqi Cui [1], Fei Li [1], Zuoyan Zhang [2],

Hongru Yang [1], Bei Zhou [1], Jie Zhao [2]

[1] Information Engineering University, Zhengzhou, China
[2] Hunan University, Changsha, China

Outline

- Rounding errors
- Existing approaches and difficulties
- Regime-based rewriting of ARFA

MOTIVATION

- Error optimization of ARFA
- How ARFA works?

APPROACH

EVALUATION

- Precision optimization effect
- Validation interval inference

CONCLUSION

- Summary
- Future work

Rounding errors

- Some inputs may trigger significant floating-point errors
- Consider:

$$f(x) = \frac{\tan(x) - \sin(x)}{x^3} \qquad \lim_{x \to 0} f(x) = 0.5$$

```
double f(double x) {
    double num = tan(x) – sin(x);
    double den = x * x * x;
    return num / den;
}
```

```
>>>  f(1e-7) // 64 bits result
0.5029258124322410

Accurate result //128 bits result
0.500000000000012
```

Rounding errors

- The root cause is:
  Finite precision bits cannot represent all real numbers exactly

⬇

- And the rounding errors can be amplified by floating-point operations

⬇

- Large errors may lead to catastrophic software failures
  - Missile yaw *[skeel' 92]*
  - Stock trading disorder *[Quinn' 83]*
  - Rocket launch failure *[Lions' 96]*

Precision optimization is a crucial work

How to solve it?

Through **rewriting**

- Changing the order of floating-point operations can reduce errors

❶ Equivalent rewriting	double a = 1.0e8, b = -1.0e8, c = 0.1;
	printf(“%.10lf”, **(a + b) + c**); // 0.1000000000
	printf(“%.10lf”, **a + (b + c)**); // 0.0999999940

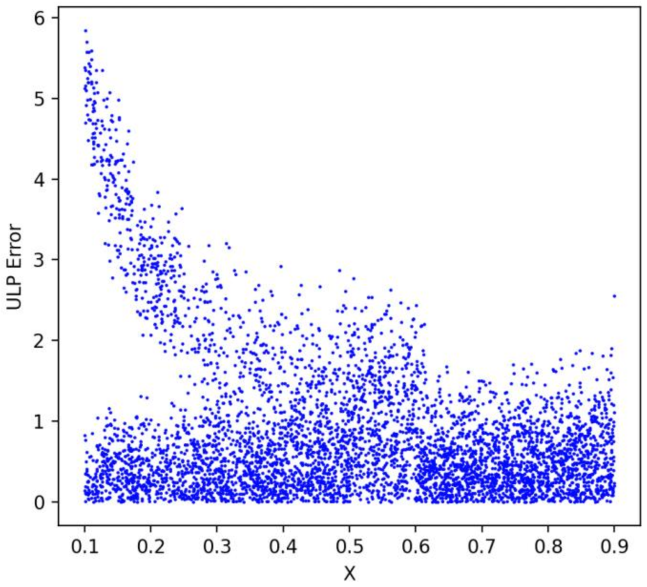
❷ Approximate rewriting	double x = 0.1e-6;
	printf(“%.10lf”, **(1-cos(x)) / (x * x)**); // 0.4996003611
	printf(“%.10lf”, **1.0 / 2.0 – (x * x) / 24.0 + (x * x * x * x) / 720.0**); // 0.5000000000

Consider: for an expression $\frac{log(1-x)}{log(1+x)}$ in interval $[0.1,0.9]$

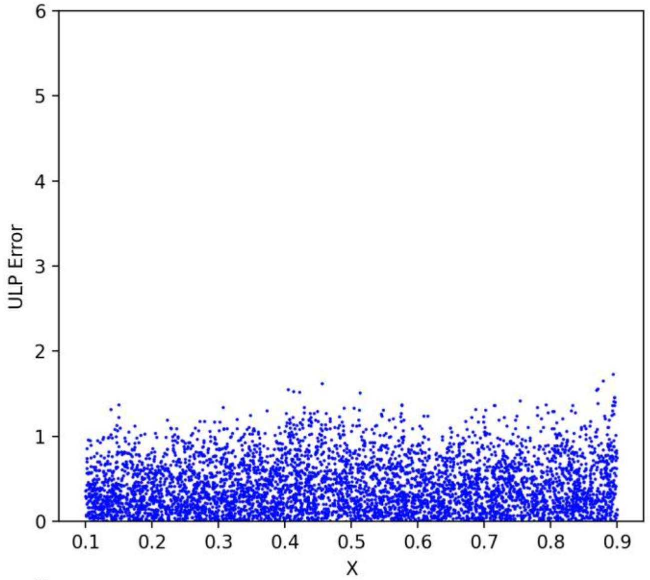
$$\frac{log(1-x)}{log(1+x)} \begin{cases} \frac{log1p(-x)}{log1p(x)} \\ \frac{log1p(x*(-x))}{log1p(x)} - 1 \end{cases}$$

How to solve it?

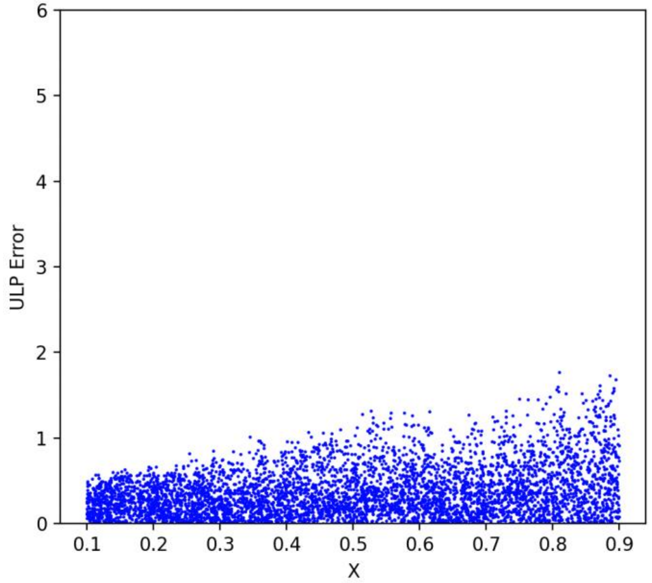
$$\frac{log(1-x)}{log(1+x)}$$



$$\frac{log1p(-x)}{log1p(x)}$$



$$\frac{log1p(x*(-x))}{log1p(x)}-1$$

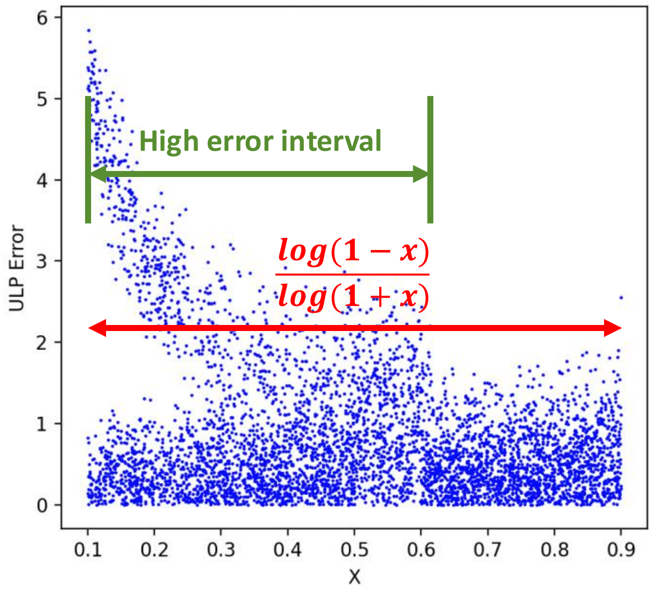


What to do next?

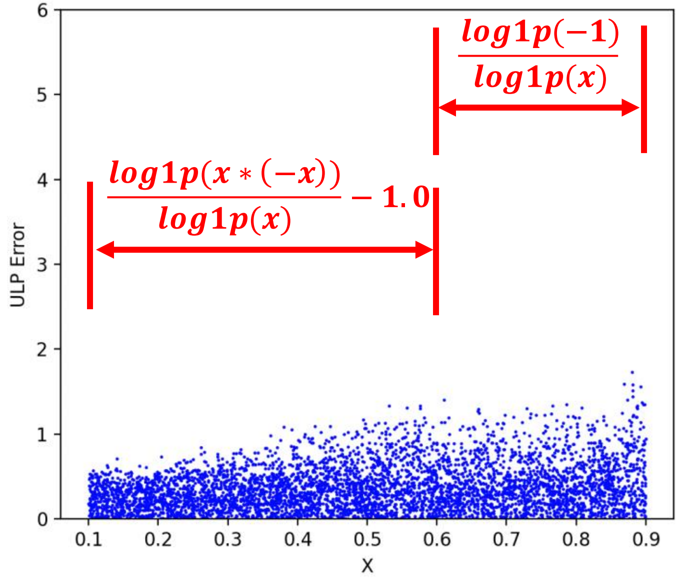
How to solve it?

We can divide the high error interval and use rewriting to optimize its precision

Regime-based rewriting



Before precision optimization



After precision optimization

Two problems:

- How to determine the rewriting interval?
- How to rewriting?

Existing approaches and difficulties

Regime-based rewriting is the main method of precision optimization

1 Herbie

- Cannot find the regime in many cases
- It's difficult to search for the optimal rewriting

2 Regina

- The number of regimes are often much larger
- Low performance

A generally applicable and effective regime inference algorithm and optimal rewriting search algorithm are still missing

Regime-based rewriting of ARFA

In order to solve the two problems

❶ **How to determine regimes more accurately?**

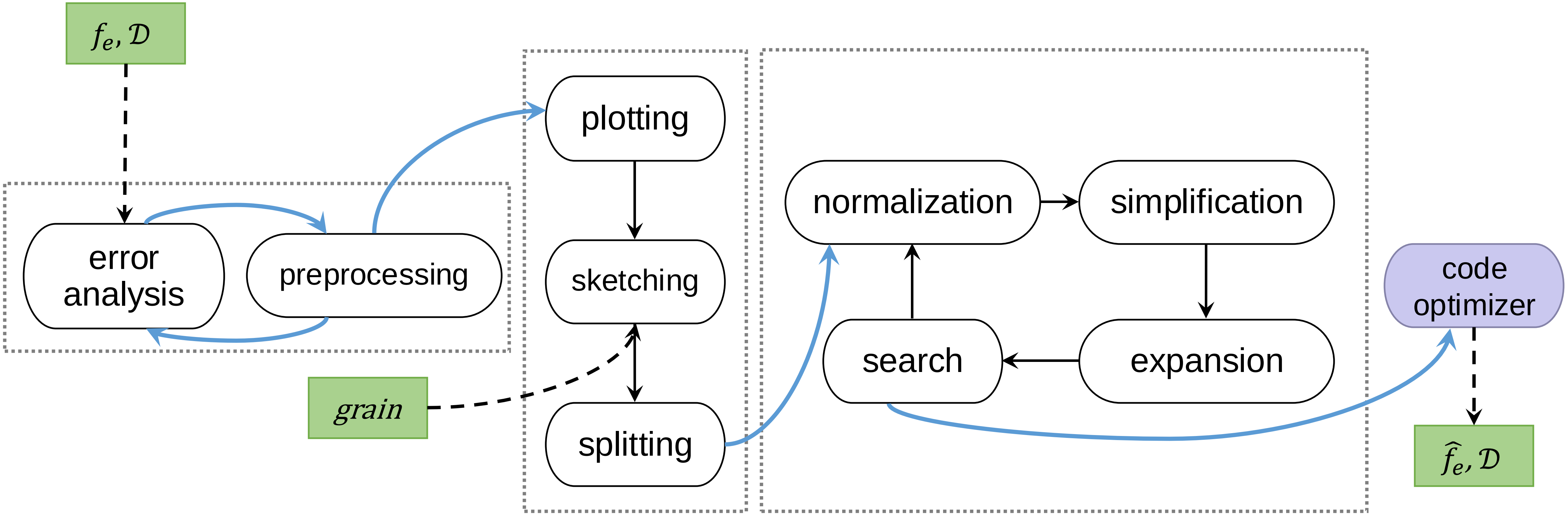
- Get a startup expression with the lowest possible error
- Determine the high error regime based on the error distribution contour

❷ **How to generate better rewriting expressions?**

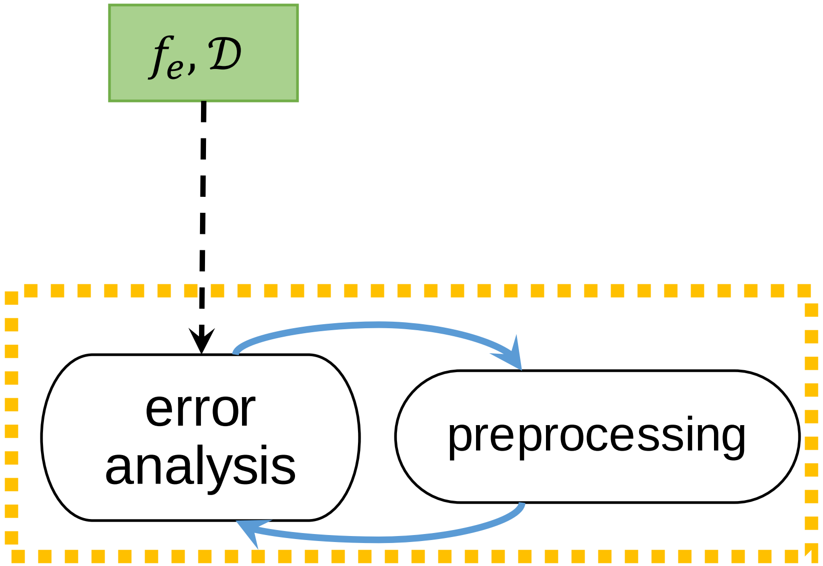
- Rewriting based on the order of operations
- Supporting customization and extension of rewriting rules
- Dynamically detect rewriting expressions instead of cost model

Error optimization using ARFA

Architecture of ARFA

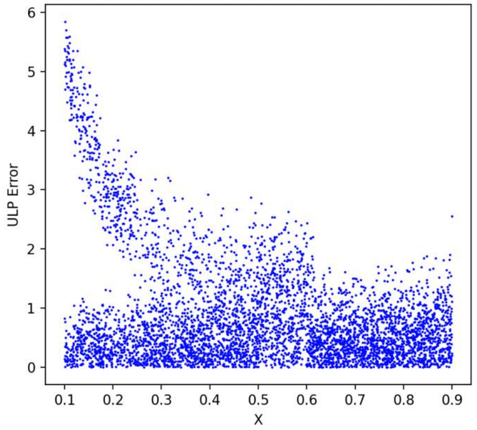


Error optimization using ARFA



Error analysis

- Use MPFR to obtain dynamic errors



Preprocessing

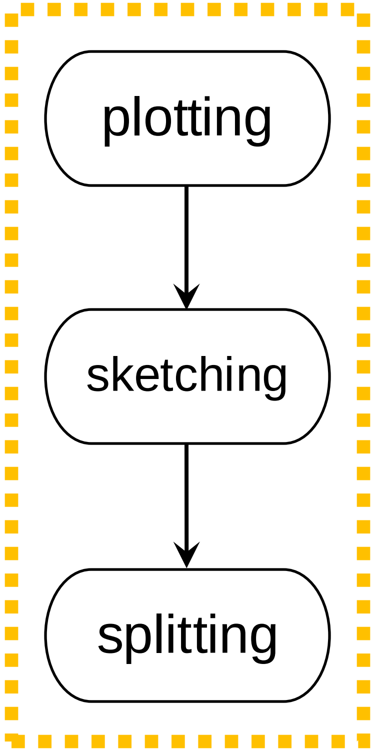
- Choose a better start-up expression by comparing the original with Herbie and Daisy's rewriting expression

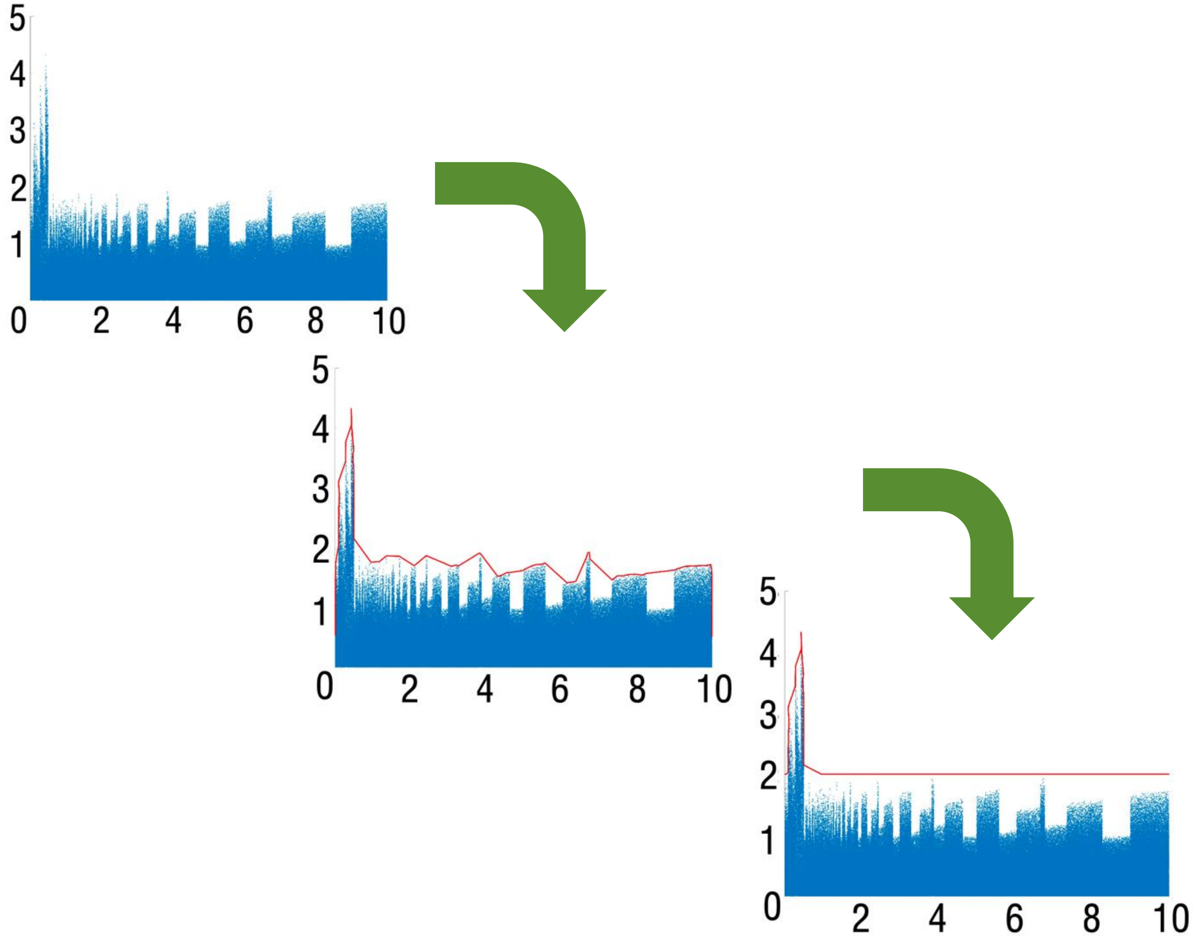
Error optimization using ARFA

Effective regime inference

Algorithm 1: Regime inference

```
   Input: Set and g
   foreach i ∈ [1, n] do
1      project onto to i-th variable dimenion;
2      M_i ← 0;
       foreach k ∈ [0, b_i) do
           if filter(x_i^(k)) = u then
3              continue;
4          lb_{M_i} ← x_i^(k);
           while filter(x_i^(k)) > u and k < b_i do
5              k ← k + 1;
6          ub_{M_i} ← x_i^(k-1); M_i ← M_i + 1;
       if M_i = 0 then
7          update u and goto line 1;
8      R_i ← 0, reset ←true;
       foreach j ∈ [0, M_i) do
           if reset = true then
9              sub-domain_{R_i} ← r_j;
10             reset ←false;
           if sizeof(sub-domain_{R_i}) < g × D_i then
11             ub_{R_i} ← ub_{j+1}; j ← j + 1;
           else
12             R_i ← R_i + 1; reset ←true;
13 intersecting sub-domains from n dimensions;
   Output: (∏_{i=1}^{n} R_i + 1) sub-domains
```





Through plotting error distribution, sketching boundary lines, and dynamically set the boundary line to obtain a more accurate regime

Error optimization using ARFA

Customized rewrite generation

Algorithm 2: Build an incomplete e-graph

Input: f_o', *e-graph*, *is_fraction*

foreach $i \in [1, t-1]$ **do**

set ← rewrite transcendental functions;

$deno_o$ ← the denominator of f_o';

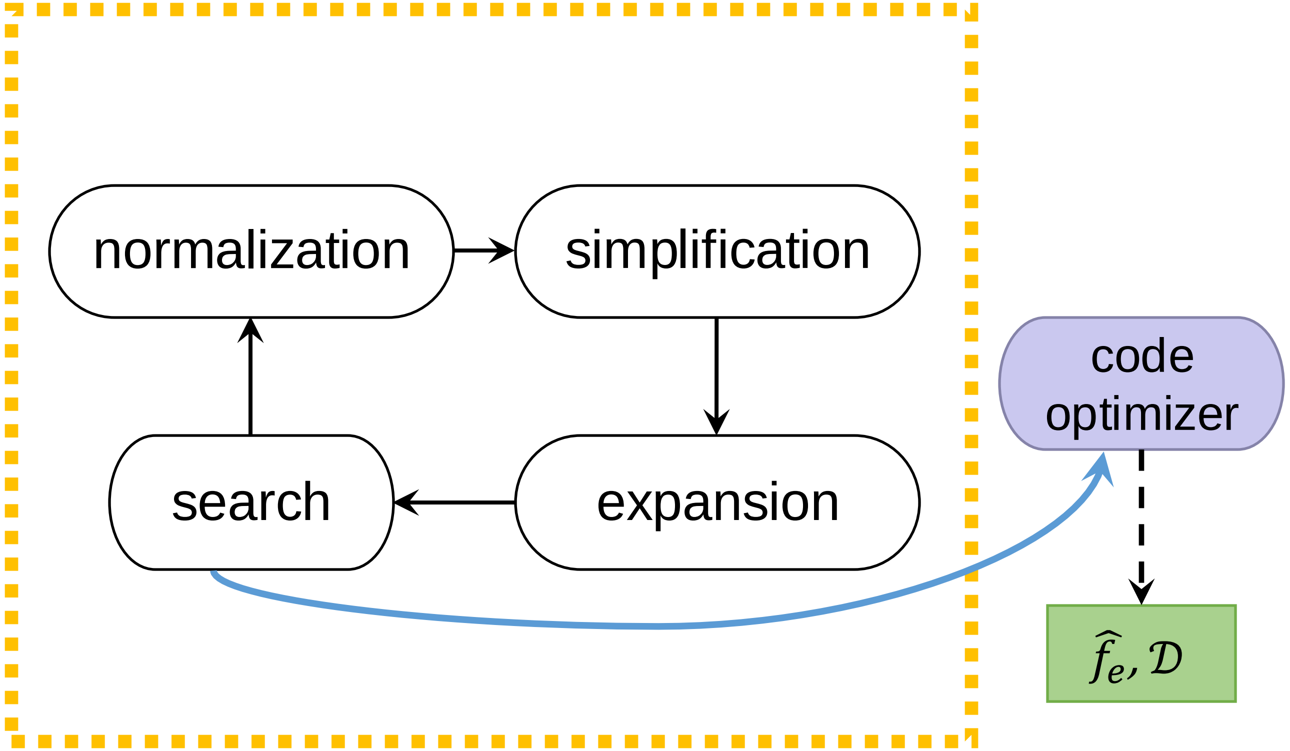
set ← rewrite transcendental functions;

else

e-graph← *e-graph* ∪ e_{tmp};

Output: *e-graph*

Normalization	$\frac{1}{x+y} + \frac{1}{x-y} + 1 \Rightarrow \frac{(x-y)+(x+y)+(x+y)(x-y)}{(x+y)(x-y)}$
Simplification	$(x-y)+(x+y)+(x+y)(x-y) \Rightarrow 2x + x^2 - y^2$
Reordering	$1 + 3x + 3x^2 + x^3 \Rightarrow 1 + (3x + (3x^2 + x^3))$
Extended rules	$log(1+x) \Rightarrow log1p(x)$



ARFA supports normalization, simplification, reordering and extended rules to generate equivalent rewriting expressions

Evaluation

Benchmarks: total 60 expressions

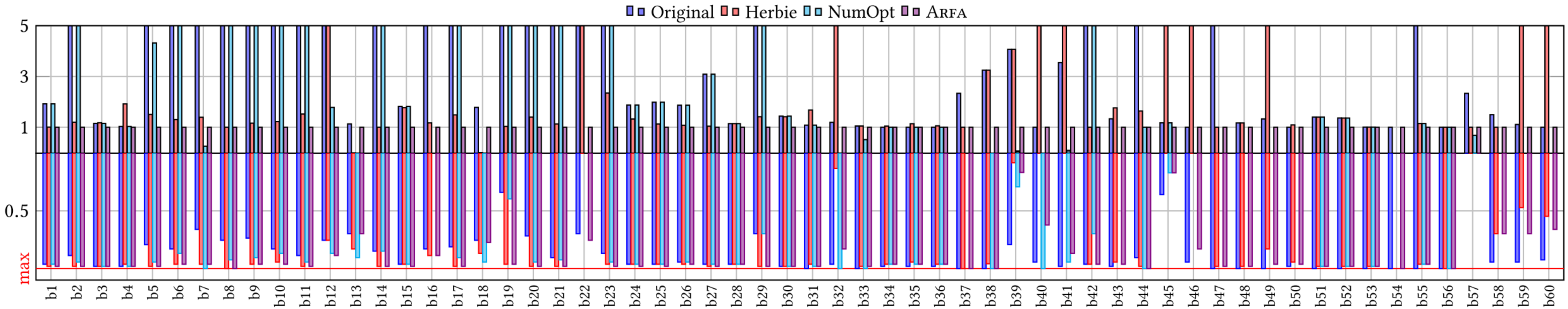
- 56 expressions are from FPBench
- 4 expressions are from real-life numerical programs

$\mathcal{D}$ is set using large but reasonable ranges

Total Benchmarks	Single-variate	Multi-variate	Control flow	Real-life
60	31	19	6	4

Evaluation — Precision optimization effect

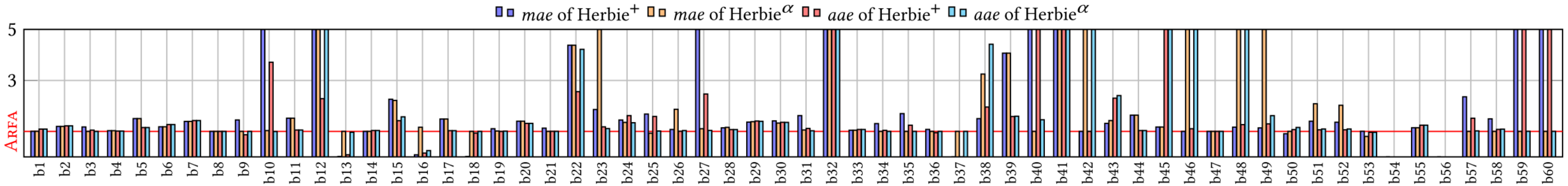
ARFA performs better than Herbie and NumOpt in 60 and 52 cases respectively



Evaluation — Quality of rewriting

Arfa allows its regime inference to work with Herbie rewrite search heuristics

ARFA's rewriting performs better than Herbie^{+} and Herbie$^{\alpha}$ in 58 and 53 cases respectively in the same regime



Conclusion

ARFA

Summary

- Effective regime inference
- Optimal rewriting generation

→

Future work

- Generalize Arfa
- Add more rewrite rules
- Integrate the RLIBM-based approaches

https://github.com/yuanyuanxia/exprAuto

THANK YOU FOR LISTENING
ANY QUESTION?

Presenter: Zuoyan Zhang

zyanz@hnu.edu.cn