Programming with Numerical Uncertainties

Eva Darulova

Max
Planck
Institute
for
Software Systems

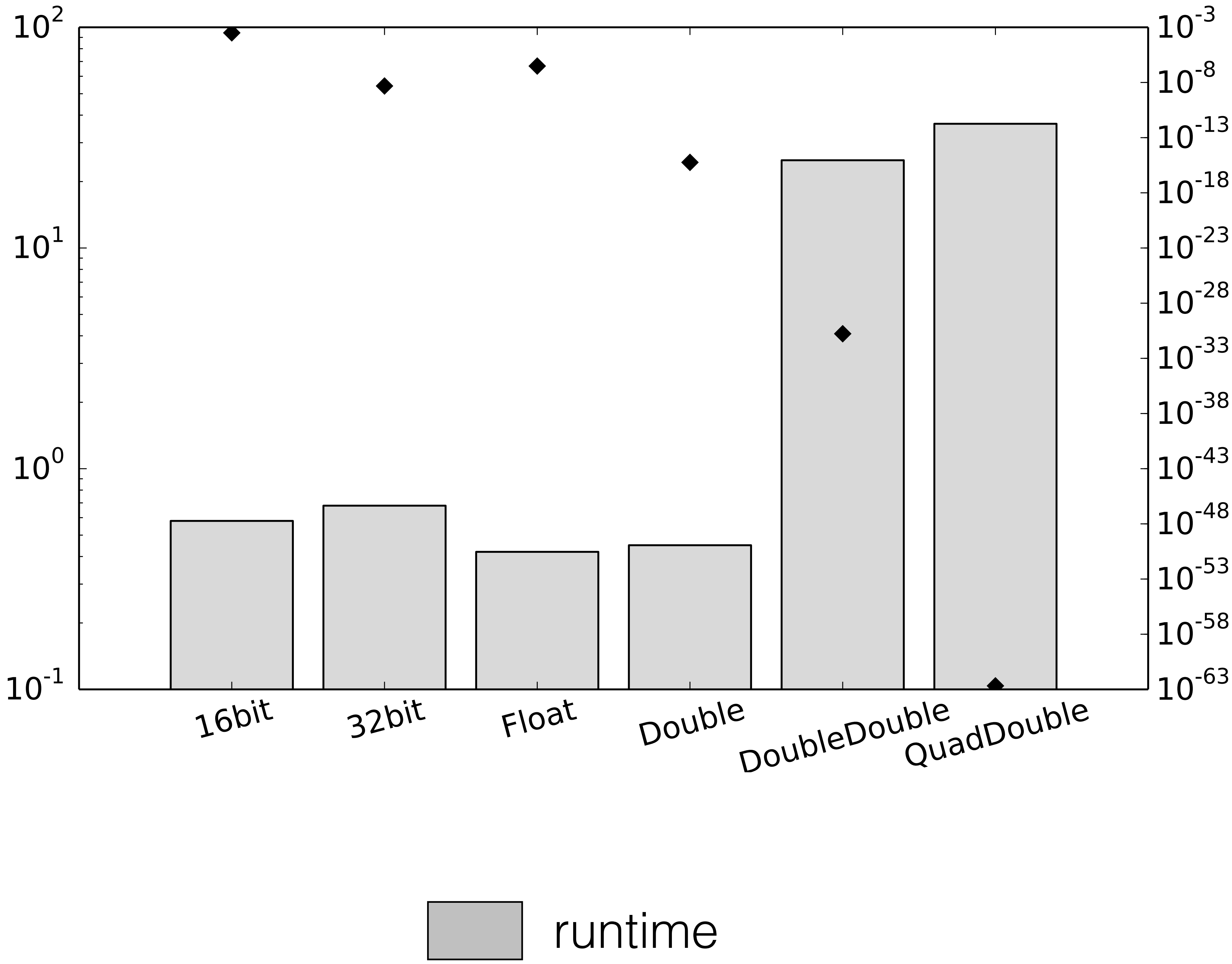
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Programming

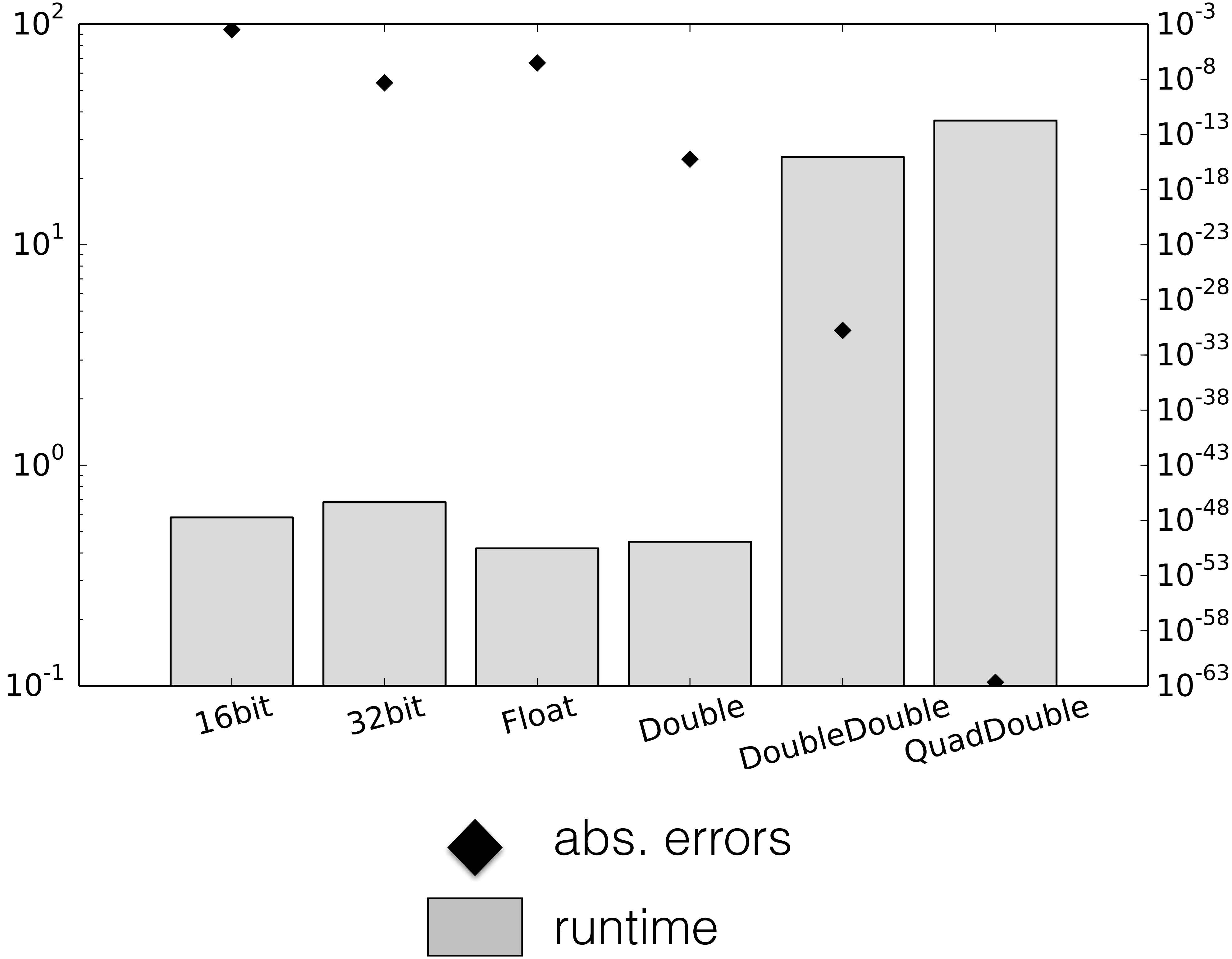
- implement a real-valued arithmetic function

```
def sine(x: ???): ??? = {
  x - (x*x*x)/6.0 + (x*x*x*x*x)/120.0 - (x*x*x*x*x*x*x)/5040.0
}
```

Which data type?



Which data type?



Programming

- implement a real-valued arithmetic function

… with Numerical Uncertainties

- roundoff errors
- input errors (e.g. measurement uncertainties)

… correctly

Programming

- implement a real-valued arithmetic function

… with Numerical Uncertainties

- roundoff errors
- input errors (e.g. measurement uncertainties)
- deliberate approximations (approximate computing)

… correctly

Outline

- a new programming model
- *estimating* errors soundly & accurately
    - nonlinearity
    - discontinuities
    - loops
- *improving* accuracy

A Real-Valued Spec

```
def sine(x: Real): Real = {

  x - (x*x*x)/6.0 + (x*x*x*x*x)/120.0 - (x*x*x*x*x*x*x)/5040.0

}
```

A Real-Valued Spec

input intervals

input errors

```
def sine(x: Real): Real = {
  require(x > -1.57079632679 && x < 1.57079632679 && x +/- 1e-11)

  x - (x*x*x)/6.0 + (x*x*x*x*x)/120.0 - (x*x*x*x*x*x*x)/5040.0

} ensuring(res => res +/- 1.001e-11)
```

max error tolerated

A Real-Valued Spec

```
def sine(x: Real): Real = {
  require(x > -1.57079632679 && x < 1.57079632679 && x +/- 1e-11)

  x - (x*x*x)/6.0 + (x*x*x*x*x)/120.0 - (x*x*x*x*x*x*x)/5040.0

} ensuring(res => res +/- 1.001e-11)
```

- what a scientist has on paper/in mind
- what you may have proven correct
- ideal baseline
- allows compiler optimisations

Rosa - the Verifying Compiler

```
def sine(x: Real): Real = {
  require(x > -1.57079632679 && x < 1.57079632679 && x +/- 1e-11)

  x - (x*x*x)/6.0 + (x*x*x*x*x)/120.0 - (x*x*x*x*x*x*x)/5040.0

} ensuring(res => res +/- 1.001e-11)
```

floating-point

fixed-point

```
Float
Double
DoubleDouble
QuadDouble
```

```
def sineWithError(x : Long): Long = {
 require(-1.5707963268 <= x && x <= 1.5707963268)

 val _tmp1 = ((x * x) >> 31)
 val _tmp2 = ((_tmp1 * x) >> 30)
 val _tmp3 = ((_tmp2 << 30) / 1610612736l)
 val _tmp4 = ((x << 1) - _tmp3)
 val _tmp5 = ((x * x) >> 31)
 val _tmp6 = ((_tmp5 * x) >> 30)
 val _tmp7 = ((_tmp6 * x) >> 31)
 val _tmp8 = ((_tmp7 * x) >> 31)
 val _tmp9 = ((_tmp8 << 28) / 2013265920l)
 val _tmp10 = ((_tmp4 + _tmp9) >> 1)
               ...
```

Fork me on GitHub

Rosa - the Verifying Compiler

```
def sine(x: Real): Real = {
  require(x > -1.57079632679 && x < 1.57079632679 && x +/- 1e-11)

  x - (x*x*x)/6.0 + (x*x*x*x*x)/120.0 - (x*x*x*x*x*x*x)/5040.0

} ensuring(res => res +/- 1.001e-11)
```

https://github.com/malyzajko/rosa

floating-point

fixed-point

Float

Double

DoubleDouble

QuadDouble

```
def sineWithError(x : Long): Long = {
 require(-1.5707963268 <= x && x <= 1.5707963268)

 val _tmp1 = ((x * x) >> 31)
 val _tmp2 = ((_tmp1 * x) >> 30)
 val _tmp3 = ((_tmp2 << 30) / 1610612736l)
 val _tmp4 = ((x << 1) - _tmp3)
 val _tmp5 = ((x * x) >> 31)
 val _tmp6 = ((_tmp5 * x) >> 30)
 val _tmp7 = ((_tmp6 * x) >> 31)
 val _tmp8 = ((_tmp7 * x) >> 31)
 val _tmp9 = ((_tmp8 << 28) / 2013265920l)
 val _tmp10 = ((_tmp4 + _tmp9) >> 1)
                    ...
```

State of the Art

quantifying numerical errors

- numerical analysis [e.g. Higham’02]
  - manual analysis by expert
- interactive theorem proving [e.g. in Coq: Boldo’11]
  - partly manual, requires expert guidance
- testing [Benz’12, Chiang’14,Paganelli’13, …]
  - gives no strong guarantees
- static analysis
  - automated, with guarantees - and reasonable accuracy

Quantifying Numerical Errors

Goal: statically, soundly, accurately and automatically

Quantifying Numerical Errors

in loop-free, branch-free code

Sound absolute error bound:

- worst-case roundoff errors
- worst-case error propagation
  - *both depend on the ranges*

```
for each arithmetic operation
  1. compute real range for intermediate value
  2. propagate existing errors
  3. compute new roundoff error
```

Standard Range Arithmetic

- interval arithmetic

$$x \in [-a, a]$$
$$x - x = [-a, a] - [-a, a] = [-2a, 2a]$$

- affine arithmetic (AA)

$$\hat{x} = x_\circ + \sum_{i=1}^{n} x_i \epsilon_i, \quad \epsilon_i \in [-1, 1] \qquad x \in \left[x_\circ - \sum_{i=1}^{n} |x_i|, \quad x_\circ + \sum_{i=1}^{n} |x_i| \right]$$

$$\begin{aligned} x - x &= x_\circ + a\epsilon_1 - (x_\circ + a\epsilon_1) \\ &= x_\circ - x_\circ + a\epsilon_1 - a\epsilon_1 = 0 \end{aligned}$$

nonlinear operations have to be approximated interval arithmetic:

[**-24.0**, 26.0]

[-∞, ∞]

```
def jetEngine(x: Real, y: Real): Real = {
  require(-5 <= x && x <= 5 && -20 <= y && y <= 5)

  val t = 3*x*x + 2*y - x
  val t2 = x*x + 1
  x + ((2*x*(t/t2)*(t/t2 - 3) + x*x*(4*(t/t2)-6))*t2 +
    3*x*x*(t/t2) + x*x*x + x + 3*((3*x*x + 2*y - x)/t2))
}
```

```
def triangleArea(a: Real, b: Real, c: Real): Real = {
require(1.0 < a && a < 9.0 && 1.0 < b && b < 9.0 && 1.0 < c && c < 9.0 &&
           a + b > c + 0.1 && a + c > b + 0.1 && b + c > a + 0.1)

  val s = (a + b + c)/2.0
  sqrt(s * (s - a) * (s - b) * (s - c))

}
```

interval arithmetic:

[**-24.0**, 26.0]

[-∞, ∞]

```
def jetEngine(x: Real, y: Real): Real = {
  require(-5 <= x && x <= 5 && -20 <= y && y <= 5)

  val t = 3*x*x + 2*y - x
  val t2 = x*x + 1
  x + ((2*x*(t/t2)*(t/t2 - 3) + x*x*(4*(t/t2)-6))*t2 +
    3*x*x*(t/t2) + x*x*x + x + 3*((3*x*x + 2*y - x)/t2))
}
```

```
def triangleArea(a: Real, b: Real, c: Real): Real = {
require(1.0 < a && a < 9.0 && 1.0 < b && b < 9.0 && 1.0 < c && c < 9.0 &&
           a + b > c + 0.1 && a + c > b + 0.1 && b + c > a + 0.1)

  val s = (a + b + c)/2.0
  sqrt(s * (s - a) * (s - b) * (s - c))

}
```

interval arithmetic:
error

Quantifying Numerical Errors

Goal: statically, soundly, accurately and automatically

Challenges:

- nonlinearity
  - range estimation
  - error computation & propagation

Interval arithmetic meets SMT

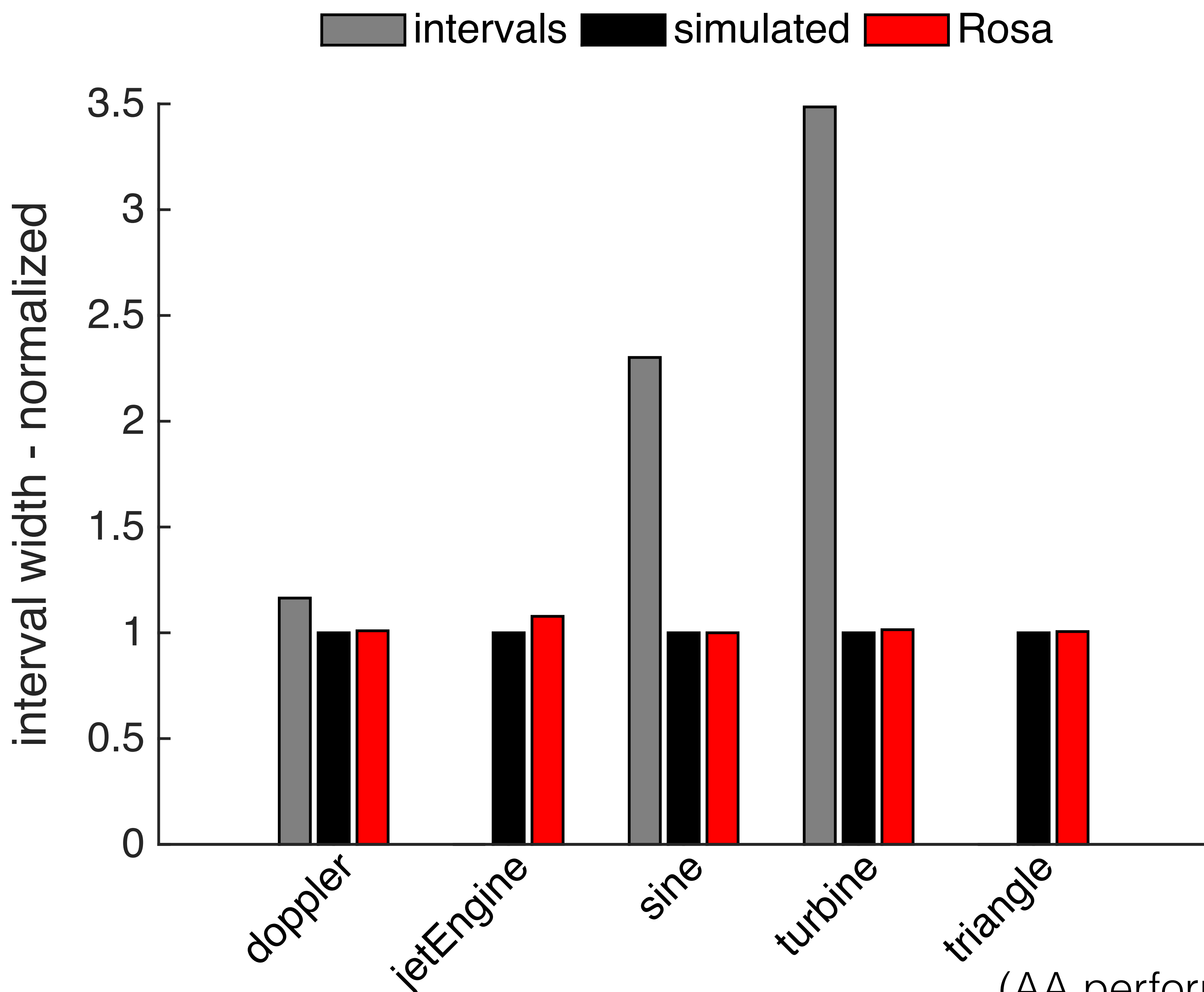
Use Z3's *nonlinear solver*:

```
for every computation step
    - get initial range with interval arithmetic
    - refine bounds with binary search with Z3
```

- includes any correlations between variables
- includes additional constraints
  - includes runtime error checks (overflow, div-by-zero, …)

selected Experiments

comparing computed normalised interval width



(AA performs like intervals)

$$|f(x) - \tilde{f}(\tilde{x})|$$

total error

Separation of Errors

$$|f(x) - \tilde{f}(\tilde{x})| \leq |f(x) - f(\tilde{x})| + |f(\tilde{x}) - \tilde{f}(\tilde{x})|$$

total error

propagation of initial errors

new roundoff errors

Separation of Errors

$$|f(x) - \tilde{f}(\tilde{x})| \leq |f(x) - f(\tilde{x})| + |f(\tilde{x}) - \tilde{f}(\tilde{x})|$$

total error

propagation of initial errors

global

real-valued property

- *derivatives*

new roundoff errors

local

finite-precision artifact

- *affine arithmetic*

Tracking Roundoff Errors

with affine arithmetic

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

for each arithmetic operation

- propagate existing x_i's
- add a new x_{n+1} for the new roundoff error
  determined according to finite-precision format

Error Propagation

$$|f(x) - f(y)|$$

Error Propagation

based on Lipschitz continuity

$$|f(x) - f(y)| \leq K|x - y| \qquad\qquad K_i = \sup_{x,y} \frac{\partial f}{\partial w_i}$$

- constants K_i capture maximum steepness of f
  - ability to magnify errors
- partial derivatives computed symbolically
- *bounded with Z3-based range computation*
  - *automatic*

```
require(-5 <= x && x <= 5 && -20 <= y && y <= 5 &&
        x +/- 1e-11 && y +/- 1e-11)
```

```
val t = (3*x*x + 2*y - x)

val t2 = x*x + 1
```

roundoff errors
become initial errors

both affine arithmetic and Lipschitz continuity
introduce over-approximations

- AA is fine for small errors, for shorter computations
- derivatives most useful on larger expressions

The ‘Competition’

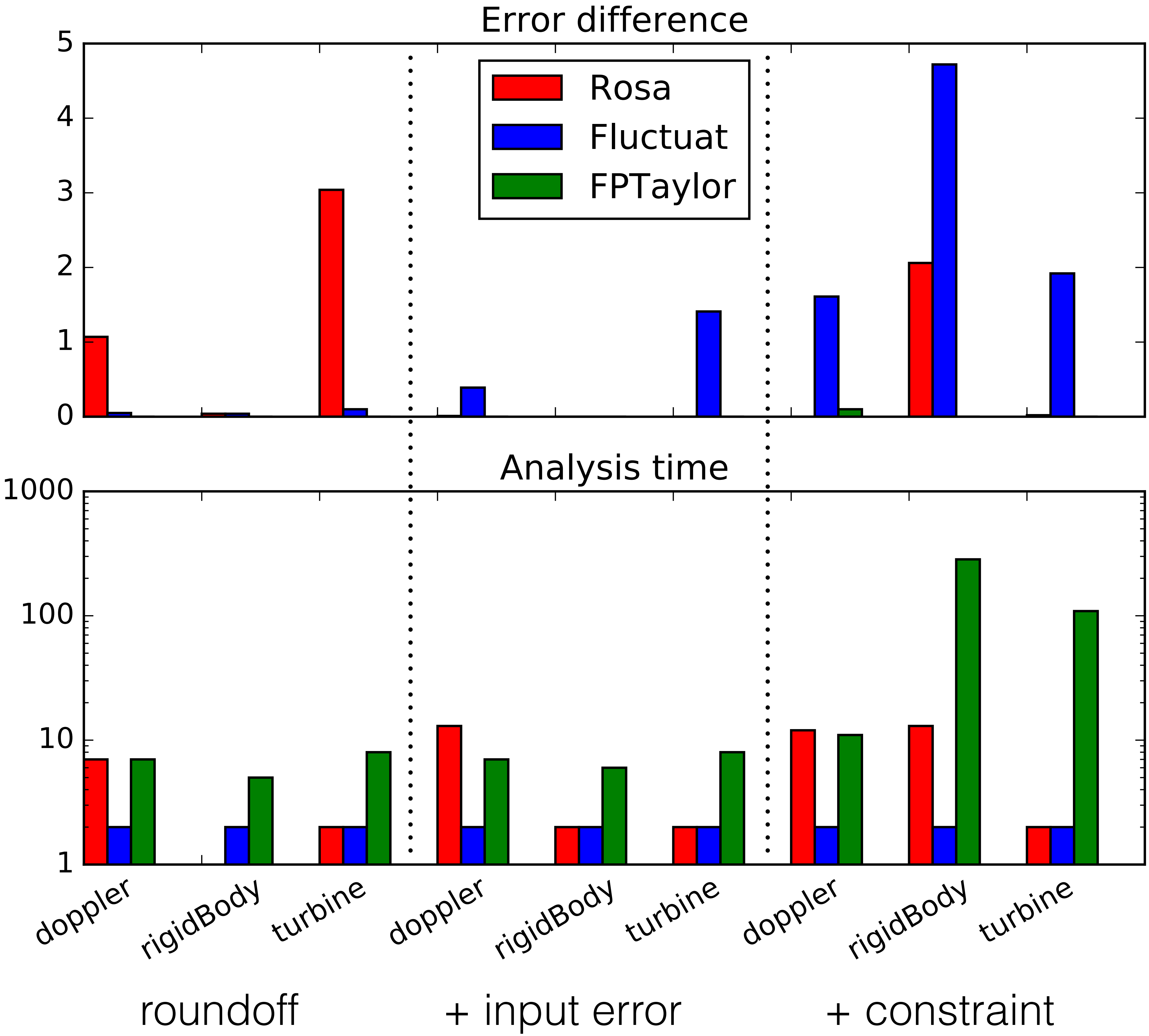
Fluctuat

- developed concurrently
- AA-based
- constraints on affine terms
- interval subdivision

FPTaylor

- Taylor approximation wrt. errors
- optimisation-based
- transcendentals
- certificates

Real2Float


Error difference
Rosa
Fluctuat
FPTaylor
0
1
2
3
4
5
Analysis time
1
10
100
1000
doppler
rigidBody
turbine
doppler
rigidBody
turbine
doppler
rigidBody
turbine
roundoff
+ input error
+ constraint

Conclusions

- for straight-line code, differences are small
- three tools, three implementations
- different (asymptotic) limitations:
  - Rosa: SMT solver may time-out unpredictably
  - Fluctuat: predictable times, but accuracy may deteriorate
  - FPTaylor: potentially large running time with constraints

Quantifying Numerical Errors

Goal: statically, soundly, accurately and automatically

Challenges:

- nonlinearity
  - range estimation
  - error computation & propagation
- discontinuities

Discontinuities

```
def anApproximation(x: Real, y: Real): Real = {
    require(-5 <= x && x <= 5 && -5 <= y && y <= 5 &&
             x +/- 1e-11 && y +/- 1e-11)

    if (y < x)
      -0.317581 + 0.0563331*x + 0.0966019*x*x + 0.0132828*y +
        0.0372319*x*y + 0.00204579*y*y

    else
      -0.330458 + 0.0478931*x + 0.154893*x*x + 0.0185116*y -
        0.0153842*x*y - 0.00204579*y*y

  }
```

What happens
when the real and the finite precision computation
take different paths?

Discontinuity Error

```
if (c)  f1
else    f2
```

$$|f_1(x) - \tilde{f}_2(\tilde{x})|$$

Discontinuity Error

```
if (c)  f1
else    f2
```

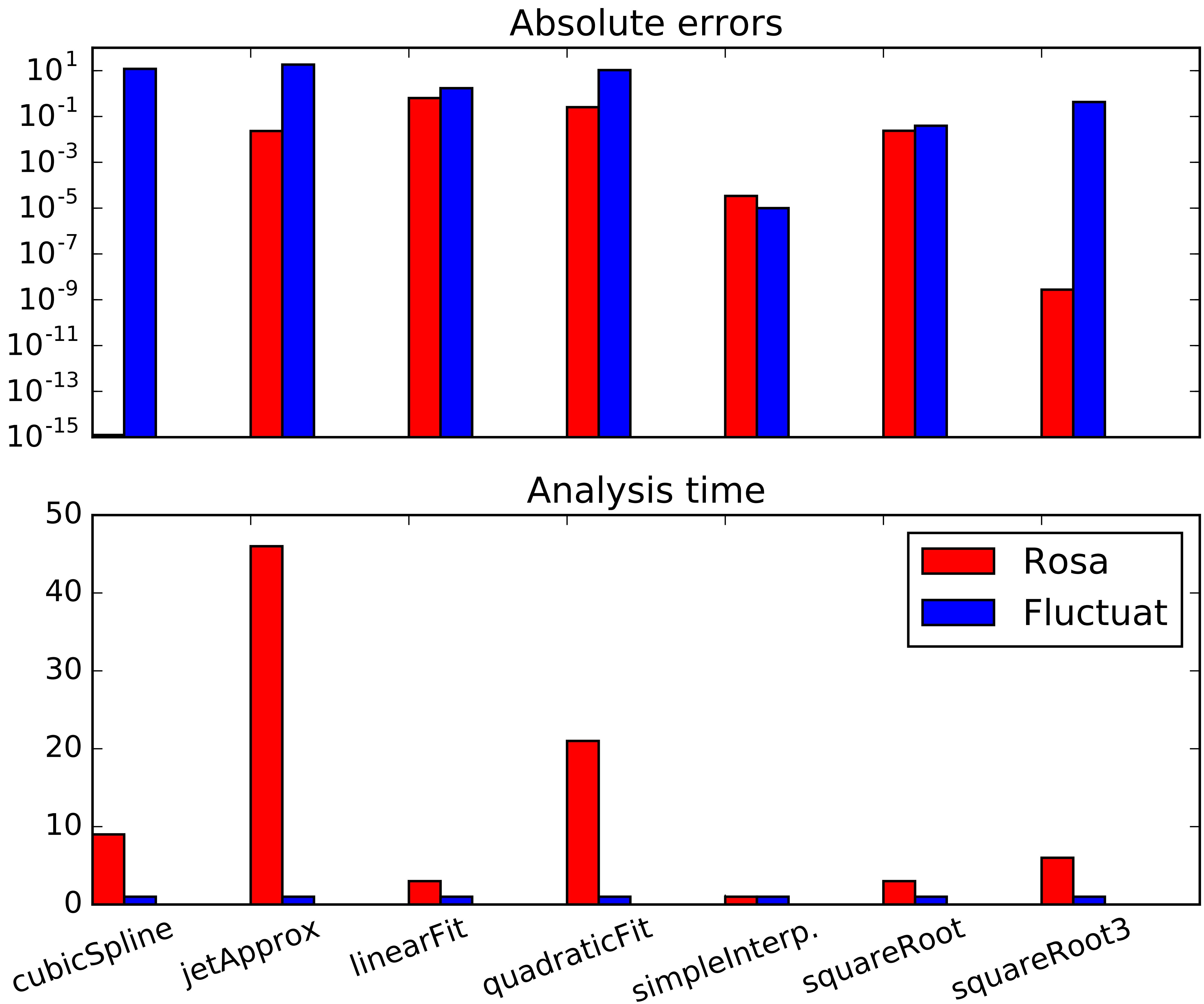
separation of errors:

$$|f_1(x) - \tilde{f}_2(\tilde{x})| \leq |f_1(x) - f_1(\tilde{x})| + |f_1(\tilde{x}) - f_2(\tilde{x})| + |f_2(\tilde{x}) - \tilde{f}_2(\tilde{x})|$$

amplification within if-branch
(Lipschitz continuity)

roundoff error
in else branch

real-valued difference
between branches


Absolute errors
10^{1}
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^{-11}
10^{-13}
10^{-15}
Analysis time
50
40
30
20
10
0
Rosa
Fluctuat
cubicSpline
jetApprox
linearFit
quadraticFit
simpleInterp.
squareRoot
squareRoot3

Quantifying Numerical Errors

Goal: statically, soundly, accurately and automatically

Challenges:

- nonlinearity
  - range estimation
  - error computation & propagation
- discontinuities
- loops

Loops

- numerical errors in general grow without bound
- one possible strategy: unrolling
- our strategy (for some loops):

  compute error as function of # iterations

$\boldsymbol{g}$: propagation error

$\boldsymbol{\sigma}$: roundoff error

$$|f^m(x) - \tilde{f}^m(\tilde{x})|$$

$\boldsymbol{g}$: propagation error

$\boldsymbol{\sigma}$: roundoff error

$$|f^m(x) - \tilde{f}^m(\tilde{x})| \leq g^m(|x - \tilde{x}|) + \sum_{i=0}^{m-1} g^i(\sigma(\tilde{f}^{m-i-1}(\tilde{x})))$$

propagation of initial error

propagation of roundoff from each iteration

IF we can compute error propagation and roundoff **globally**

$$|f^m(x) - \tilde{f}^m(\tilde{x})| \leq K^m\lambda + ((I - K)^{-1}(I - K^m))\sigma$$

initial error

propagation over all iterations

maximum roundoff over all iterations

```
def sine(x: Real): Real = {
  require(-5 <= x && x <= 5)
  x - x*x*x/6 + x*x*x*x*x/120
}

def pendulum(t: Real, w: Real, n: LoopCounter): (Real,Real) = {
  require(-2 <= t && t <= 2 && -5 <= w && w <= 5 &&
              -2.01 <= ~t && ~t <= 2.01 && -5.01 <= ~w && ~w <= 5.01)

  if (n < 1000) {
    val h: Real = 0.01
    val L: Real = 2.0
    val m: Real = 1.5
    val g: Real = 9.80665
    val k1t = w
    val k1w = -g/L * sine(t)
    val k2t = w + h/2*k1w
    val k2w = -g/L * sine(t + h/2*k1t)
    val tNew = t + h*k2t
    val wNew = w + h*k2w
    pendulum(tNew, wNew, n + 1)
  } else {
    (t, w)
  }
}
```

```
def sine(x: Real): Real = {
  require(-5 <= x && x <= 5)
  x - x*x*x/6 + x*x*x*x*x/120
}

def pendulum(t: Real, w: Real, n: LoopCounter): (Real,Real) = {
  require(-2 <= t && t <= 2 && -5 <= w && w <= 5 &&
             -2.01 <= ~t && ~t <= 2.01 && -5.01 <= ~w && ~w <= 5.01)

  if (n < 1000) {
    val h: Real = 0.01
    val L: Real = 2.0
    val m: Real = 1.5
    val g: Real = 9.80665
    val k1t = w
    val k1w = -g/L * sine(t)
    val k2t = w + h/2*k1w
    val k2w = -g/L * sine(t + h/2*k1t)
    val tNew = t + h*k2t
    val wNew = w + h*k2w
    pendulum(tNew, wNew, n + 1)
  } else {
    (t, w)
  }
}
```

assumption:
bounded ranges

for loop body

1. analyse derivative
2. compute new roundoff
3. plug into closed-form expression

```
def sine(x: Real): Real = {
  require(-5 <= x && x <= 5)
  x - x*x*x/6 + x*x*x*x*x/120
}

def pendulum(t: Real, w: Real, n: LoopCounter): (Real,Real) = {
  require(-2 <= t && t <= 2 && -5 <= w && w <= 5 &&
               -2.01 <= ~t && ~t <= 2.01 && -5.01 <= ~w && ~w <= 5.01)

  if (n < 1000) {
    val h: Real = 0.01
    val L: Real = 2.0
    val m: Real = 1.5
    val g: Real = 9.80665
    val k1t = w
    val k1w = -g/L * sine(t)
    val k2t = w + h/2*k1w
    val k2w = -g/L * sine(t + h/2*k1t)
    val tNew = t + h*k2t
    val wNew = w + h*k2w
    pendulum(tNew, wNew, n + 1)
  } else {
    (t, w)
  }
}
```

# iterations	Fluctuat (unrolling)	Our tool
50	2.43e-13	2.21e-14
100	∞	8.82e-14
250	∞	2.67E-12
500	∞	6.54E-10

(time: 8s)

Quantifying Numerical Errors

Goal: statically, soundly, accurately and automatically

Challenges:

- nonlinearity
  - range estimation
  - error computation & propagation
- discontinuities
- loops

Quantifying Numerical Errors

Goal: statically, soundly, accurately and automatically

Key Ideas:

- separation of errors
- combine interval/affine arithmetic with SMT
- use derivatives with SMT

Finite-Precision is Non-Associative

Previous technique assumed fixed computation order.

error determined by simulation

`(-0.0078)*st1+0.9052*st2 +(-0.0181)*st3 + (-0.0392)*st4 +(-0.0003)*y1+0.0020*y2`	3.06e-3
exploit non-associativity	
`((0.9052*st2)+(((st3*-0.0181) +(-0.0078*st1))+ (((-0.0392*st4) +(-0.0003*y1))+(0.002*y2))))`	1.39e-3

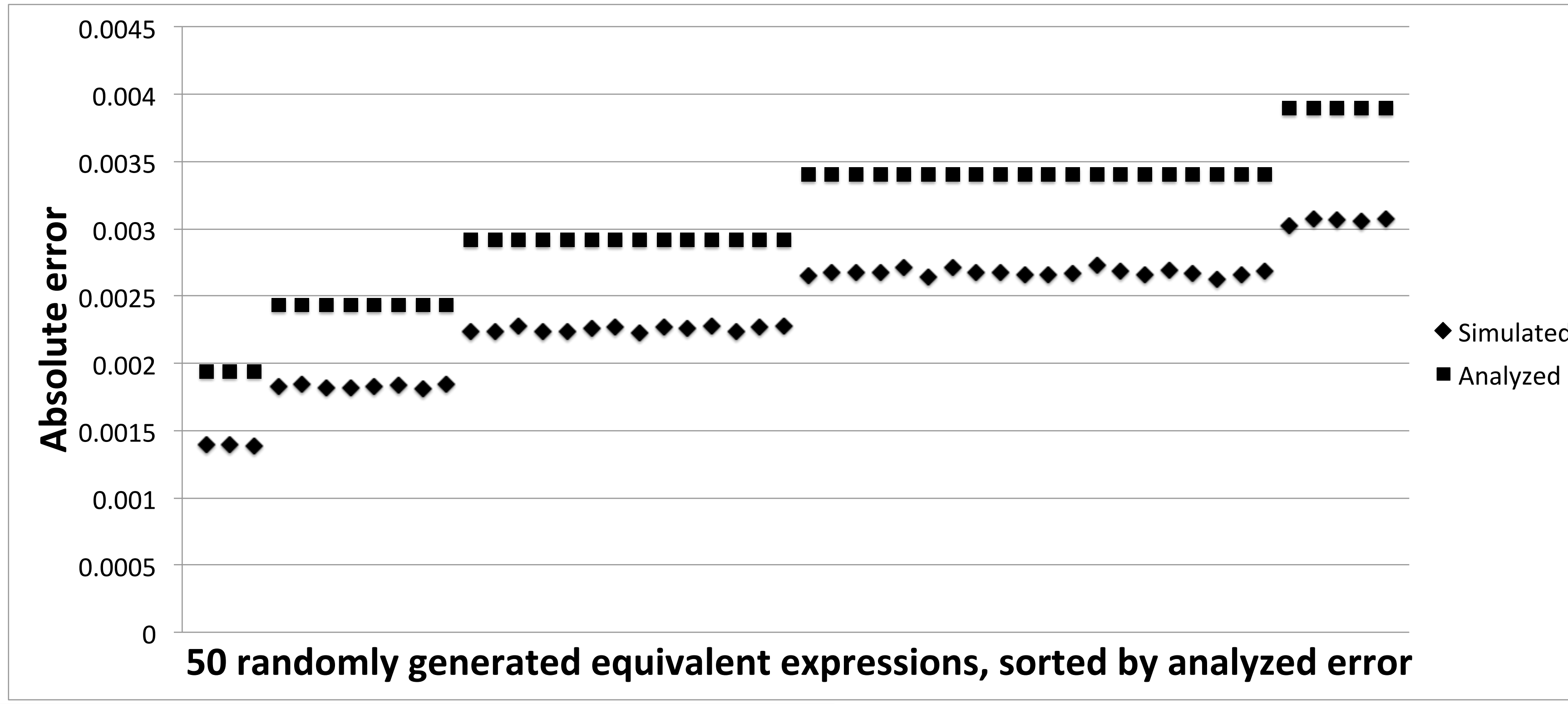
Synthesizing Accurate Expressions

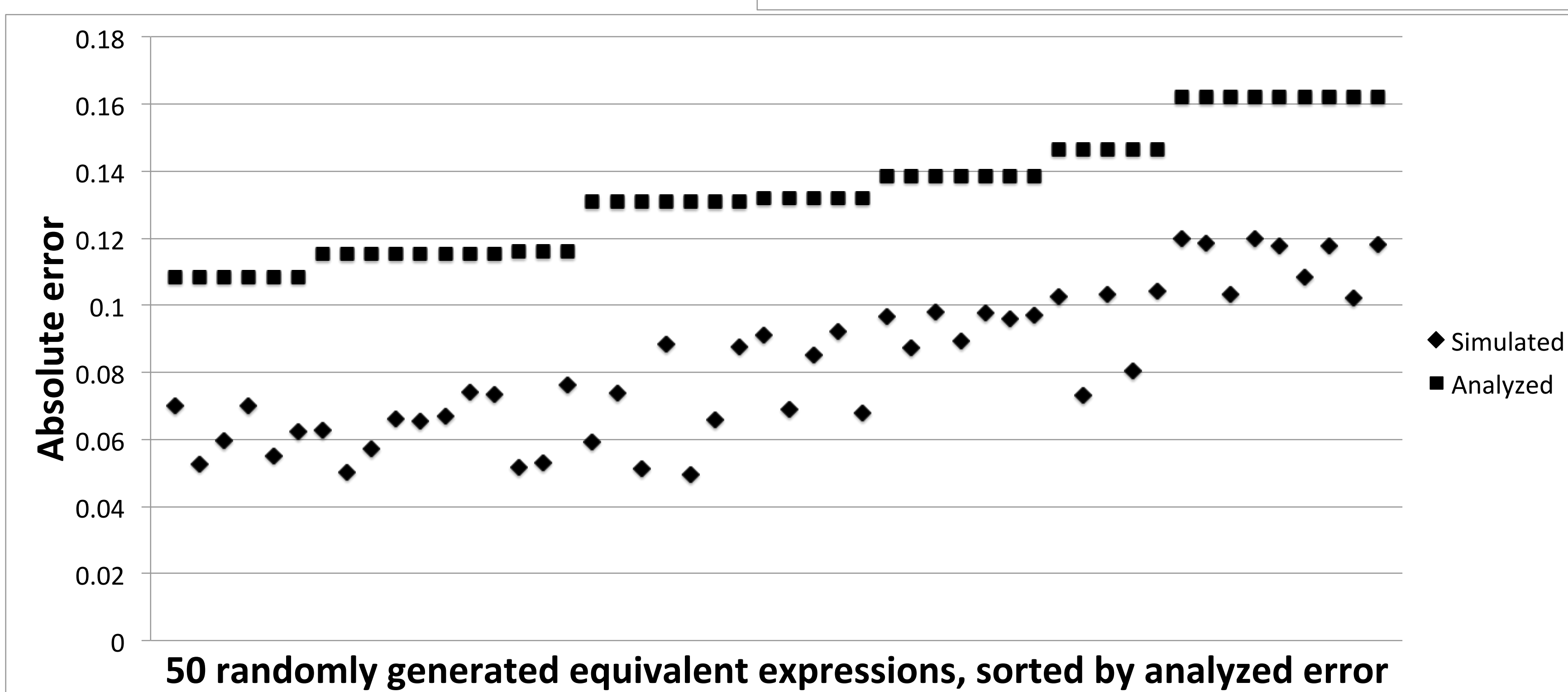
[E. Darulova, V. Kuncak, R. Majumdar and I. Saha, EMSOFT'13]

- enumerating all expressions is infeasible
- small local error can produce large global error

- genetic programming as search strategy:
  - genetic algorithm over AST (instead of strings)
  - 30 generations with a population of 30
  - mutation: associativity, distributivity, etc.
  - crossover: labeling
  - fitness function: static analysis, AA-based

Static Analysis as Fitness Function



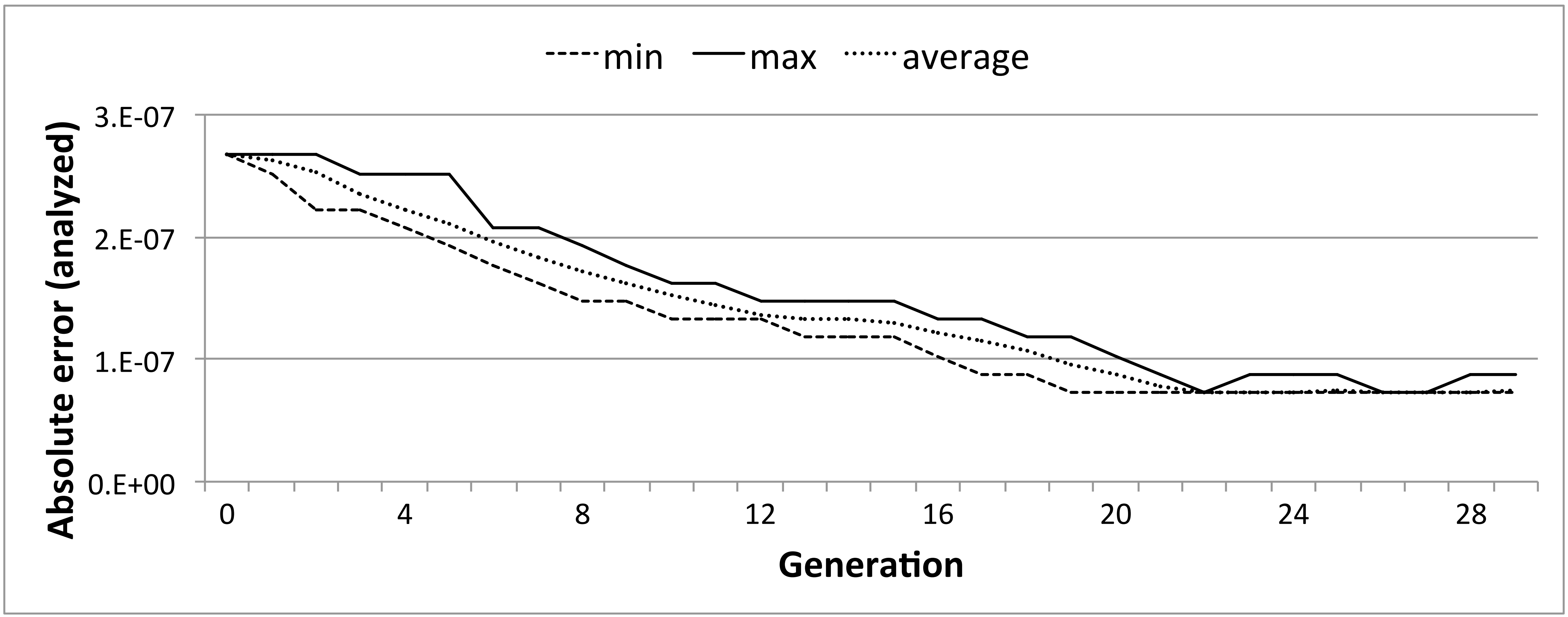


Guarantees

- no optimality
  - incomplete search
  - static, over-approximating analysis may not discriminate correctly

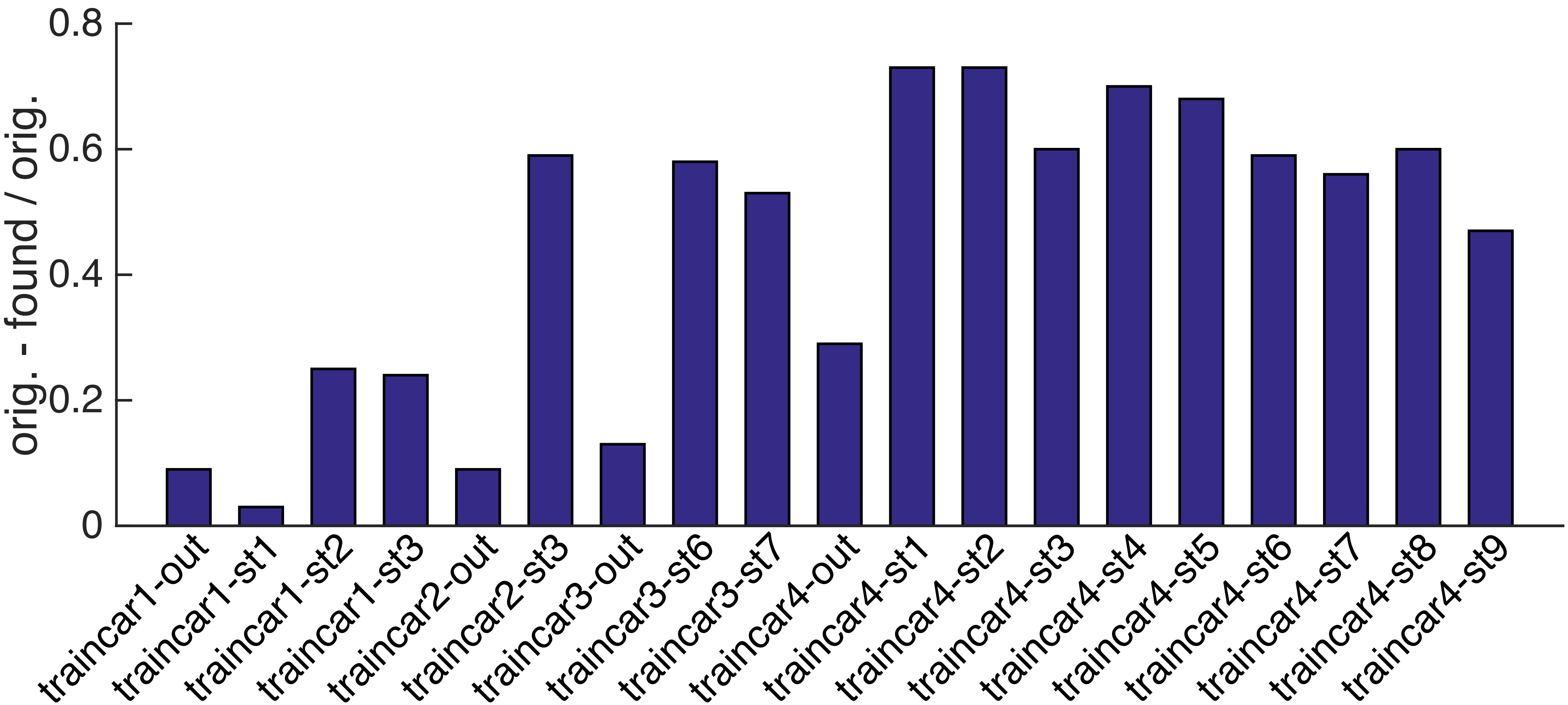
- find expression with *provable* smallest bound

Evolution of Errors



selected Experiments

improvement over automatically generated initial version



The ‘Competition’

[Martel’09, Ioualalen&Martel’12]

- abstract domain to represent under-representation of possible rewritings
- local search
- optimise bit-length

[Eldib&Wang’13]

- SMT-base synthesis with skeleton
- linear expressions only
- local search

Herbie [PLDI’15]

- testing-based heuristic search
- average error
- ‘regime’ inference

Conclusion

Quantifying numerical uncertainties is hard.

Combination of techniques + advances in SMT

$\downarrow$

automated & accurate results

- nonlinearity, branches and loops
- non-associativity

Thank you!