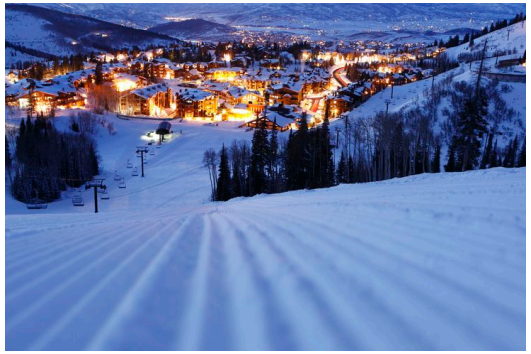
Symbolic AD with conditionals for error and instability analysis

Ganesh Gopalakrishnan
School of Computing, University of Utah, **Salt Lake City**, UT 84112



PL/FV group website: cpu.cs.utah.edu

THE UNIVERSITY OF UTAH®

NSF CCF 1817073, 1704715

Presenting work by Arnab Das (Ph.D. student)

with Ian Briggs (Ph.D. student), Mark Baranowski (Ph.D. student) and Sriram Krishnamoorthy (PNNL)

Why are Numerics Increasingly Important?

- ML folks love reduced precision and reduced energy
- **HPC folks love all of this + rigor ☺**

- Rigorously Reducing Precision is Rife with Unsolved Challenges:
  - How to does one tightly bound the # of bits lost?
  - For "large" Straight-line Code? ← "Satire" to appear in SC'20
  - With Conditionals? ← this talk
  - With Loops? ← Useful methods lacking

- Path forward:
  - Conquer each step
  - Try to stay "small" and "local"
  - Eventually try and evolve static analysis and ML-based methods
    - that calibrate off of the rigorous – this can buy scale
  - Bake it into compilers

Our Current Formal Analysis Capabilities of FP Code

[a]From *Sound Compilation of Reals* by E. Darulova and V. Kuncak
[b]Only the approximate function is shown

```
double jetEngine(double x1, double x2) {
  double t = 3 * x1 * x1 + 2 * x2 - x1
  double q = x1 * x1 + 1
  double p = t / q
  double s1 = 2 * x1 * p * (p - 3)
  double s2 = x1 * x1 * (4 * p - 6)
  double s3 = 3 * x1 * x1 * p
  double s4 = x1 * x1 * x1
  double s5 = 3 * p
  return x1 + ((s1 + s2) * q + s3 + s4 + x1 + s5)
}
```

- Two Inputs
- One Output
- About 30 operators
- Given input intervals of values for x1 and x2
- We can estimate the absolute error of the output value fairly tightly using many modern methods within acceptable times

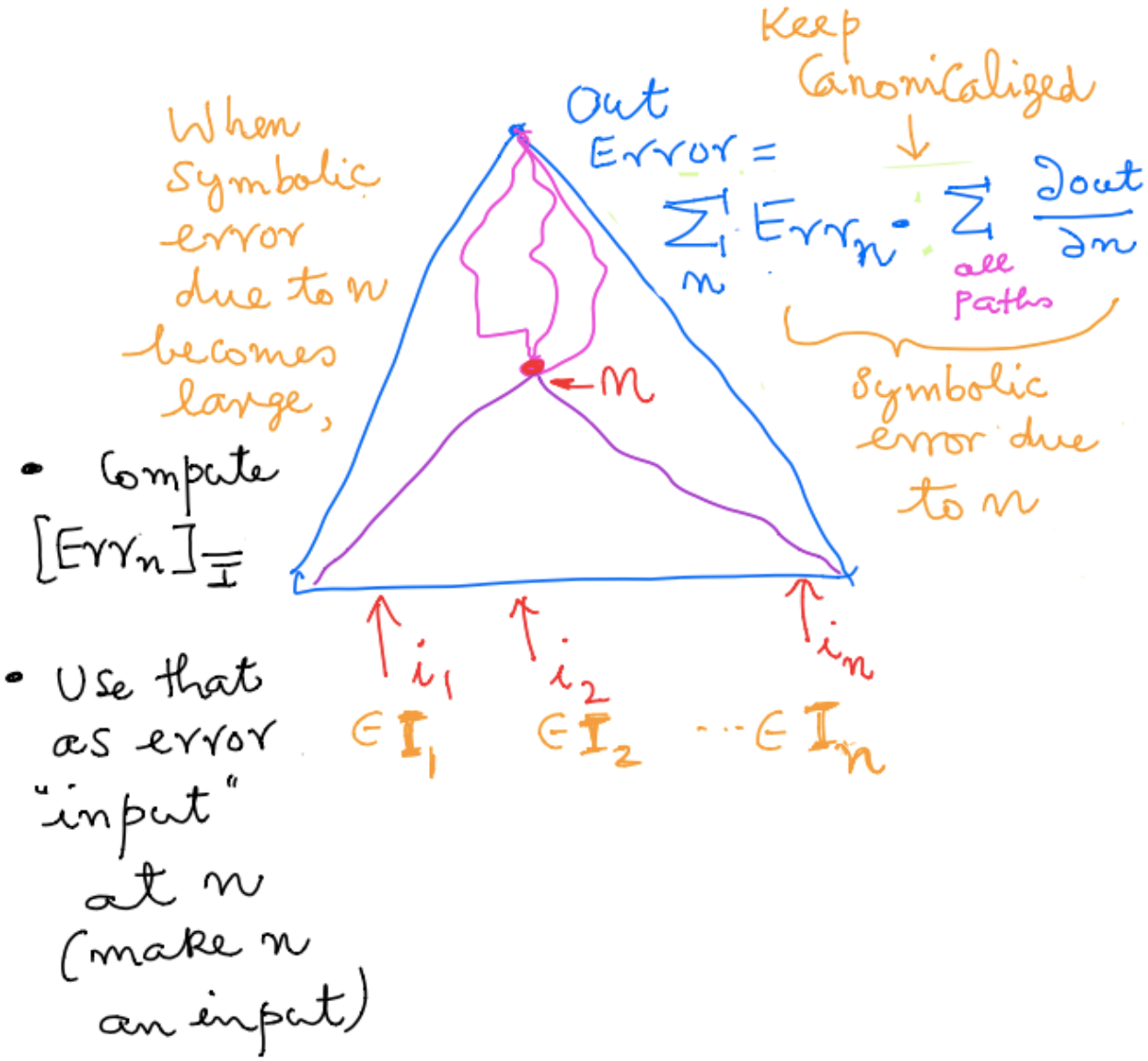
Recent Advances for Straightline Code

- Most tools up until 2020 gave up on expressions of about 100 operators
- **In 2020, we introduced a method (to appear in SC'20) that can scale up to 4 million operators**
  - **The Satire tool realizes our method (see** https://arxiv.org/abs/2004.11960 for a precursor to Satire that went upto 1M operators)
  - **Satire is based on**
    - **Symbolic Reverse-Mode AD**
    - **Expression Canonicalization to help Global Optimizers**
    - **An Info-Theory-Guided Conservative Abstraction Method**

SATIRE = Symbolic Abstraction-guided TechnIque for Rigorous estimation of Error

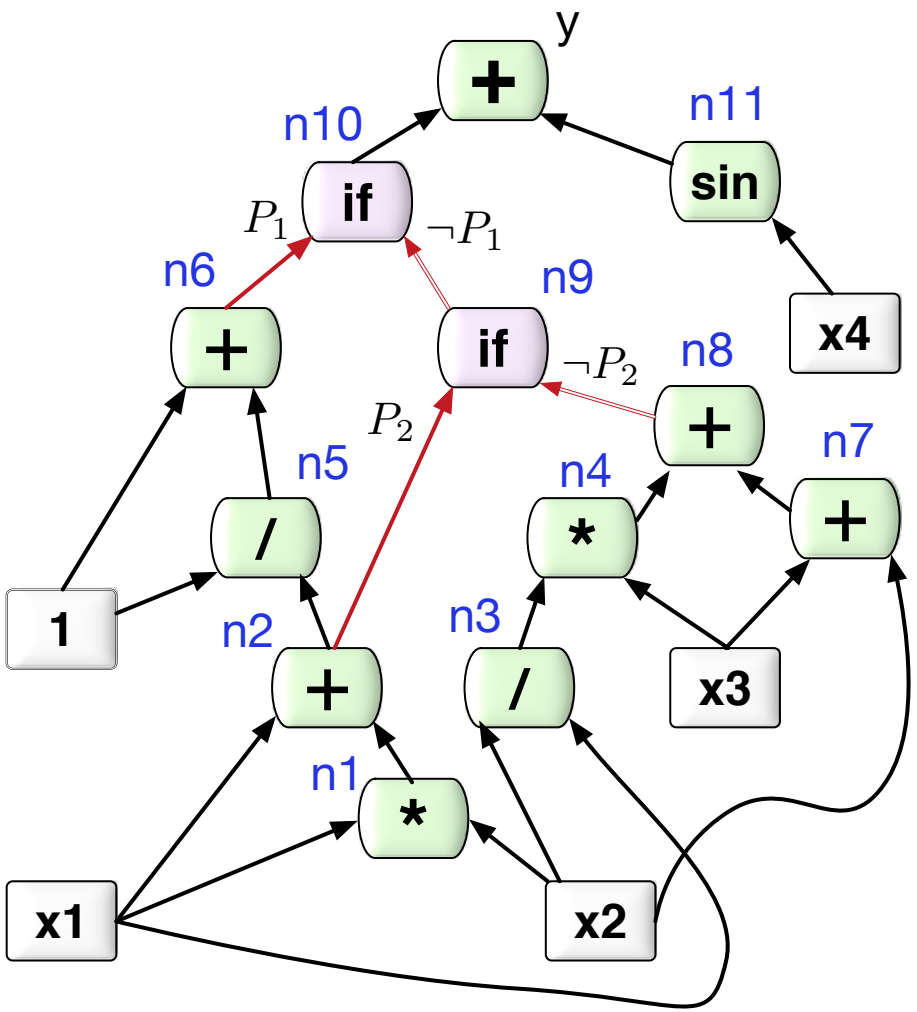
Satire in One Slide

- Symbolic reverse-mode A/D
  - Derivative-strength of Out wrt n
- Keep expressions canonicalized
- Multiply forward error at n
- Compute incrementally
- Abstract when Err_n becomes large



Trouble due to Conditionals

```
1   INPUTS { x1 fl64 : (0.01, 1.0);
2            x2 fl64 : (0.01, 1.0);
3            x3 fl64 : (0.01, 1.0);
4            x4 fl64 : (0.01, 1.0); }
5   OUTPUTS { y ; }
6   REQUIRES { RSC1 : (x1*x3 < x2) || (x2 >= x4) ; }
7   EXPRS { h rnd64 = (x2/x1) + x3 ;
8           g rnd64 = x1 + x1*x2 ; //<–LOC0
9           if ( x1–x2 < 0.4 ) then //P1
10            g rnd64 = 1 + 1/g ; // <–LOC1
11          else
12            if (( x3*x3 > 0.25 ) && (x4*h <= x1*x1)) //P2
13              then g rnd64 = h + x2*x3 ; // <–LOC2
14            endif
15          endif
16          // <–LOC3
17          y rnd64 = g + sin(x4) ;
18  }
```

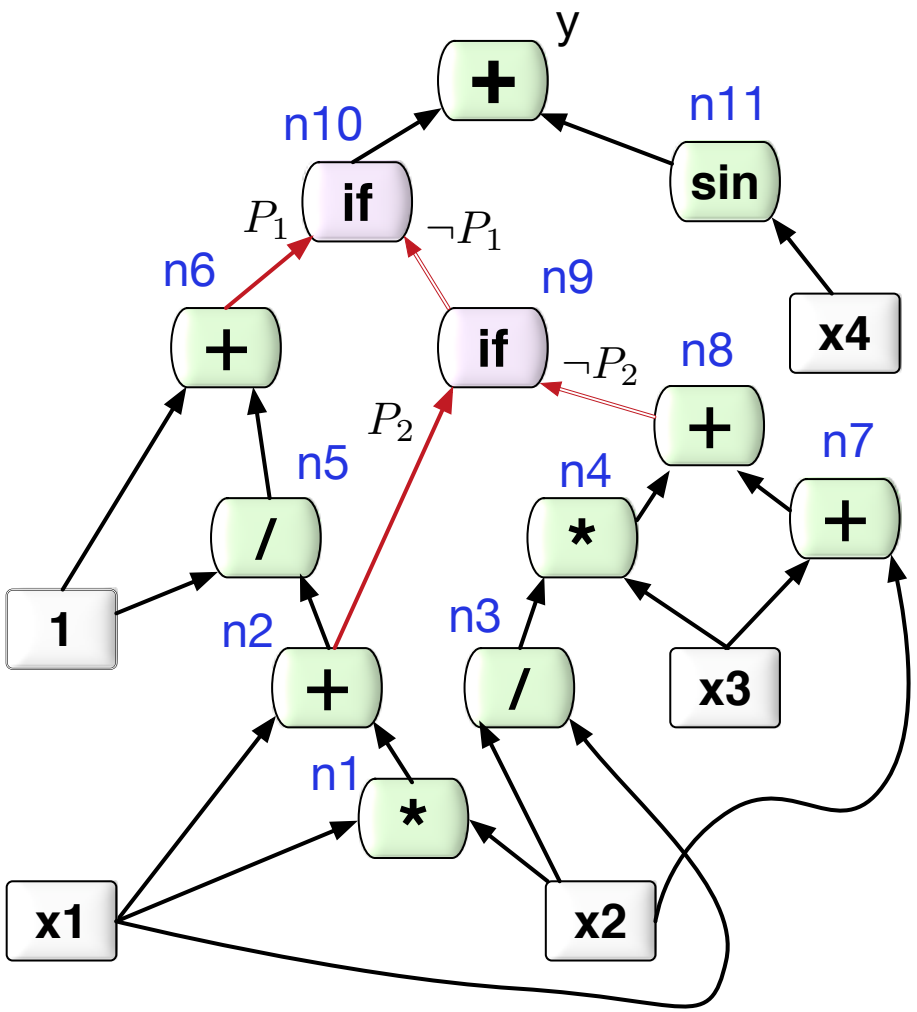


- Absolute error depends on the path taken
- The path taken depends on the error

- In addition, the “unstable conditionals” can suddenly switch, causing an “amplified output value jump”

Challenges

```
1   INPUTS { x1 fl64 : (0.01, 1.0);
2            x2 fl64 : (0.01, 1.0);
3            x3 fl64 : (0.01, 1.0);
4            x4 fl64 : (0.01, 1.0); }
5   OUTPUTS { y ; }
6   REQUIRES { RSC1 : (x1*x3 < x2) || (x2 >= x4) ; }
7   EXPRS { h rnd64 = (x2/x1) + x3 ;
8           g rnd64 = x1 + x1*x2 ; //<–LOC0
9           if ( x1–x2 < 0.4 ) then //P1
10            g rnd64 = 1 + 1/g ; // <–LOC1
11          else
12            if (( x3*x3 > 0.25 ) && (x4*h <= x1*x1)) //P2
13              then g rnd64 = h + x2*x3 ; // <–LOC2
14            endif
15          endif
16          // <–LOC3
17          y rnd64 = g + sin(x4) ;
18  }
```



- (How) can we obtain the worst-case absolute error across all paths?
- (How) can we obtain the worst case instability jump over all conditionals?
- Can we do this for practically sized problems?

Results

- Preliminary results are encouraging
  - We will release our conditional benchmarks into http://fpbench.org/
  - They include many examples found in books like this + others…

Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) Hardcover – December 22, 2004

- The examples far exceed in size those handled by the only other tool (PRECISA)

- Loops anyone? ☺

Concluding Remarks

- We have a method based on extending symbolic reverse-mode AD to handle conditionals
- Error analysis requires **predicated path-derivative summation**
  - Also a cond-widening technique to infer instability errors exactly
- It also requires extending the underlying global optimizer to handle conditional global optimization

- **Progress in this area directly aids synthesis and resynthesis:**
  - Safely precision-tune code
  - Do operator selection
  - Re-analyze to keep error and instability jump under acceptable limits