Programming Language and Compiler Support for Uncertainties

Eva Darulová
Viktor Kuncak
Writing accurate numerical software is hard because

- math is hard
- there are many unavoidable uncertainties
  - measurement errors
  - finite precision arithmetic
  - limited resources/truncation errors

Our goal: automate reasoning about uncertainties
def jetEngine(x1: ???, x2: ???): ??? = {

    val t = (3*x1*x1 + 2*x2 - x1)
    val t2 = x1*x1 + 1
    x1 + ((2*x1*(t/t2)*(t/t2 - 3) + x1*x1*(4*(t/t2)-6))*t2 +
            3*x1*x1*(t/t2) + x1*x1*x1 + x1 +
            3*((3*x1*x1 + 2*x2 -x1)/t2))
}

def jetEngine(x1: Real, x2: Real): Real = {
    require(-5 <= x1 && x1 <= 5 && -20 <= x2 && x2 <= 5)

    val t = (3*x1*x1 + 2*x2 - x1)
    val t2 = x1*x1 + 1
    x1 + ((2*x1*(t/t2)*(t/t2 - 3) + x1*x1*(4*(t/t2)-6))*t2 + 3*x1*x1*(t/t2) + x1*x1*x1 + x1 + 3*((3*x1*x1 + 2*x2 - x1)/t2))

} ensuring(res => res <= 5200 && res +/- 1e-7)
def jetEngine(x1: Double, x2: Double): Double = {
  require(-5 <= x1 && x1 <= 5 && -20 <= x2 && x2 <= 5)

  val t = (3*x1*x1 + 2*x2 - x1)
  val t2 = x1*x1 + 1
  x1 + ((2*x1*(t/t2)*(t/t2 - 3) + x1*x1*(4*(t/t2)-6))*t2 +
           3*x1*x1*(t/t2) + x1*x1*x1 + x1 +
           3*((3*x1*x1 + 2*x2 -x1)/t2))

  } ensuring(res => -1997.03679344963210 <= res &&
             res <= 5109.33737094863900 &&
             res +/- 1.6170398214864217e-08)
Precision vs Efficiency

Jet Engine

Run=me' abs.'Error
Why Program in Reals?

- separation of concerns
  mathematical vs implementation

- verification over reals

- compiler optimisations

- specification of ideal behaviour
  quantifying deviations
def jetEngine(x1: Real, x2: Real): Real = {
  require(-5 <= x1 && x1 <= 5 && -20 <= x2 && x2 <= 5)
  ...
}

Real compiler

fixed-point program

def jetEngine(x1: Long, x2: Long): Long = {
  val _tmp1 = ((1610612736l * x1) >> 30)
  val _tmp2 = ((_tmp1 * x1) >> 31)
  ...
}

floating-point program

def jetEngine(x1: Double, x2: Double): Double = {
  val t = (3*x1*x1 + 2*x2 - x1)
  ...
}

refined postcondition
Roundoff Errors

track real computation +/- uncertainties

real computation

roundoff error
Roundoff Errors

track real computation +/- uncertainties

affine arithmetic to maintain linear correlations

\[ x = x_0 + \sum_{i=1}^{n} x_i \epsilon_i , \quad \epsilon_i \in [-1, 1] \]
Nonlinear Arithmetic

```python
def jetEngine(x1: Real, x2: Real): Real = {
    require(-5 <= x1 && x1 <= 5 && -20 <= x2 && x2 <= 5)
    val t = (3*x1*x1 + 2*x2 - x1)
    val t2 = x1*x1 + 1
    x1 + ((2*x1*(t/t2)*(t/t2 - 3) + x1*x1*(4*(t/t2)-6))*t2 +
        3*x1*x1*(t/t2) + x1*x1*x1 + x1 +
        3*((3*x1*x1 + 2*x2 -x1)/t2))
}
```

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

- includes any correlations between variables

- gain precision even if solver fails at some step
Conditionals

```python
def approx(x: Real): Real = {
    require(0 <= x && x <= 4 && x +/- 1e-8)

    if (x <= 2) {
        10.0268 - 14.4554*x + 4.92857*x*x
    } else {
        -4.125 + 4.125*x - 0.75*x*x
    }
}
```

- discontinuity error: max \(|xin\mathbb{R}| - \text{else}[\mathbb{F}]|

- evaluated over ``critical inputs`` ~ 2 ± 1e-8
Iterative Algorithms
Iterative Algorithms

tan x_1 - k*(2*sin x_1 + sin x_2) = 0

tan x_2 - 2*k*(sin x_1 + sin x_2) = 0
Iterative Algorithms

tan x_1 - k*(2*sin x_1 + sin x_2) = 0

tan x_2 - 2*k*(sin x_1 + sin x_2) = 0

graph showing:
- Residual: \( \varepsilon \approx 0.025 \)
- True error: \( \tau \approx 0.055 \)
Iterative Algorithms

\[ \tan x_1 - k^* (2^* \sin x_1 + \sin x_2) = 0 \]
\[ \tan x_2 - 2^* k^* (\sin x_1 + \sin x_2) = 0 \]

- runtime library to certify (black-box) computed results
- relies on theorems from validated computation of roots

\[ \epsilon \approx 0.025 \]
\[ \tau \approx 0.055 \]
Conclusion

Programmers

should not have to worry about low-level details of implementing real computation

only need to be aware of magnitude of errors

Reasoning can be automated to handle straight-line code, conditionals and self-stabilising iterative computations

benchmarks: physics, chemistry, biology and control systems