Verified Compilation and Optimization of Floating-Point Programs in CakeML

Heiko Becker, Robert Rabe, Eva Darulova, Magnus O. Myreen, Zachary Tatlock, Ramana Kumar, Yong Kiam Tan, Anthony Fox

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS
Technische Universität München
UPPSALA UNIVERSITET
CHALMERS UNIVERSITY OF TECHNOLOGY
PLSE
DeepMind
ARM
Carnegie Mellon University

Floating-Point Arithmetic in Unverified & Verified Compilers

Floating-Point Arithmetic in Unverified & Verified Compilers

COMPCERT

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof

Floating-Point Arithmetic in Unverified & Verified Compilers

LLVM

- IEEE-754 arithmetic
- fast-math optimizations
- no correctness guarantees

COMPCERT

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof

Floating-Point Arithmetic in Unverified & Verified Compilers

LLVM

- IEEE-754 arithmetic
- fast-math optimizations
- no correctness guarantees

CAKEML
A Verified Implementation of ML

- no floating-point support

COMPCERT

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof

Floating-Point Arithmetic in Unverified & Verified Compilers

GCC

LLVM

- IEEE-754 arithmetic
- fast-math optimizations
- no correctness guarantees

before our work

CAKEML
A Verified Implementation of ML

- no floating-point support

COMPCERT

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof

Floating-Point Arithmetic in Unverified & Verified Compilers

GCC

LLVM

- IEEE-754 arithmetic
- fast-math optimizations
- no correctness guarantees

in this talk

CAKEML
A Verified Implementation of ML

- **IEEE-754 arithmetic**
- **fast-math-style optimizations**
- **correctness & accuracy proofs**

COMPCERT

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof

Fast-Math-Style Optimizations in Compilers

LLVM

$$x * (x * (x * x)) \rightarrow (x * x) * (x * x)$$

Fast-Math-Style Optimizations in Compilers

LLVM

$$x * (x * (x * x)) \rightarrow (x * x) * (x * x)$$

changes bit-level result

Fast-Math-Style Optimizations in Compilers

GCC

LLVM

$$x * (x * (x * x)) \rightarrow (x * x) * (x * x)$$

changes bit-level result

CAKEML

A Verified Implementation of ML

preserve IEEE-754 floating-point arithmetic

Fast-Math-Style Optimizations in Compilers

LLVM

$x * (x * (x * x)) \rightarrow (x * x) * (x * x)$

changes bit-level result

preserve IEEE-754 floating-point arithmetic

requires bit-level accuracy

Fast-Math-Style Optimizations in Compilers

LLVM

CAKEML
A Verified Implementation of ML

$$x * (x * (x * x)) \rightarrow (x * x) * (x * x)$$

preserve IEEE-754 floating-point arithmetic

changes bit-level result

requires bit-level accuracy

Fast-Math-Style Optimizations in Compilers

$x * (x * (x * x$

...nt arithmetic

change

accuracy

CAV'19

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler

Heiko Becker[1], Eva Darulova[1], Magnus O. Myreen[2], and Zachary Tatlock[3]*

[1] MPI-SWS, Saarland Informatics Campus (SIC), {hbecker,eva}@mpi-sws.org
[2] Chalmers University of Technology, myreen@chalmers.se
[3] University of Washington, ztatlock@cs.washington.edu

Abstract. Verified compilers like CompCert and CakeML offer increasingly sophisticated optimizations. However, their deterministic source

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV'19]

verified floating-point optimizations

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV'19]

proof-of-concept optimizer

verified floating-point optimizations

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]

proof-of-concept optimizer

verified floating-point optimizations

fine-grained control

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV'19]

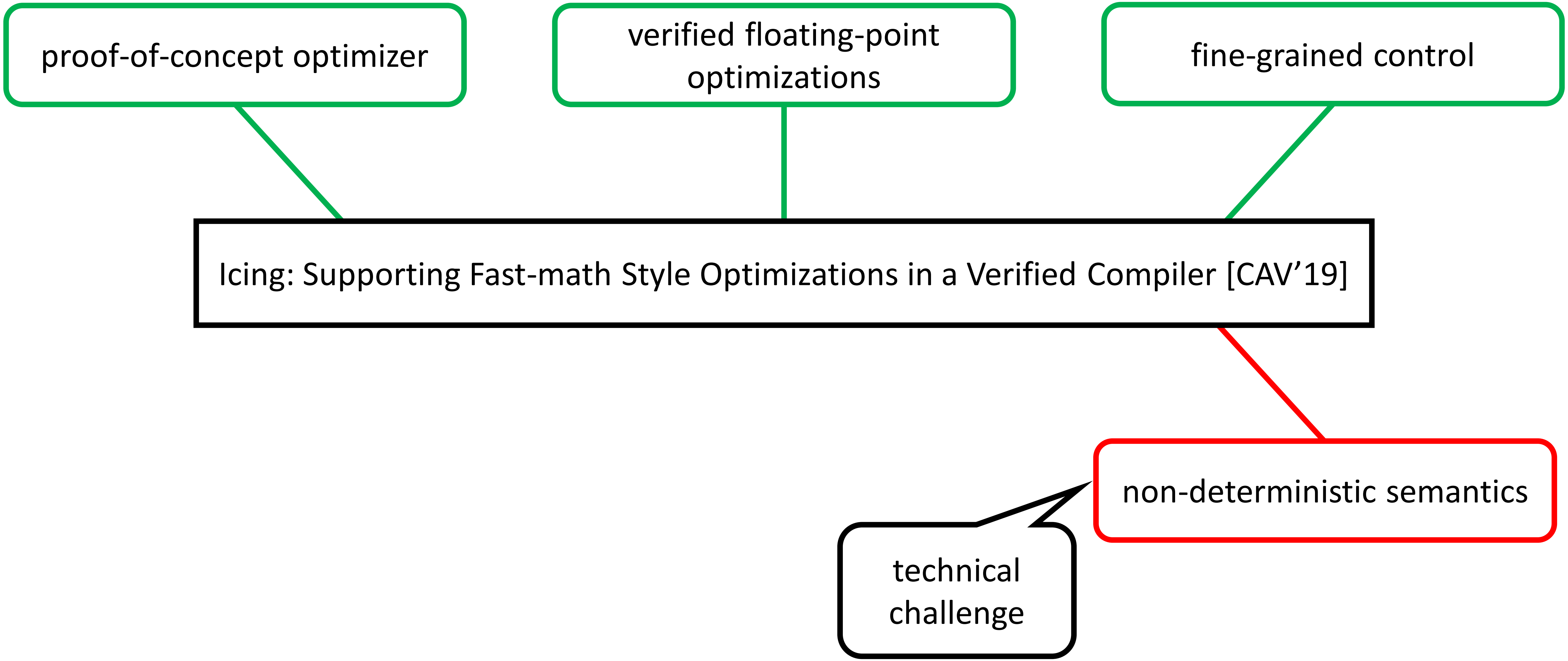
proof-of-concept optimizer

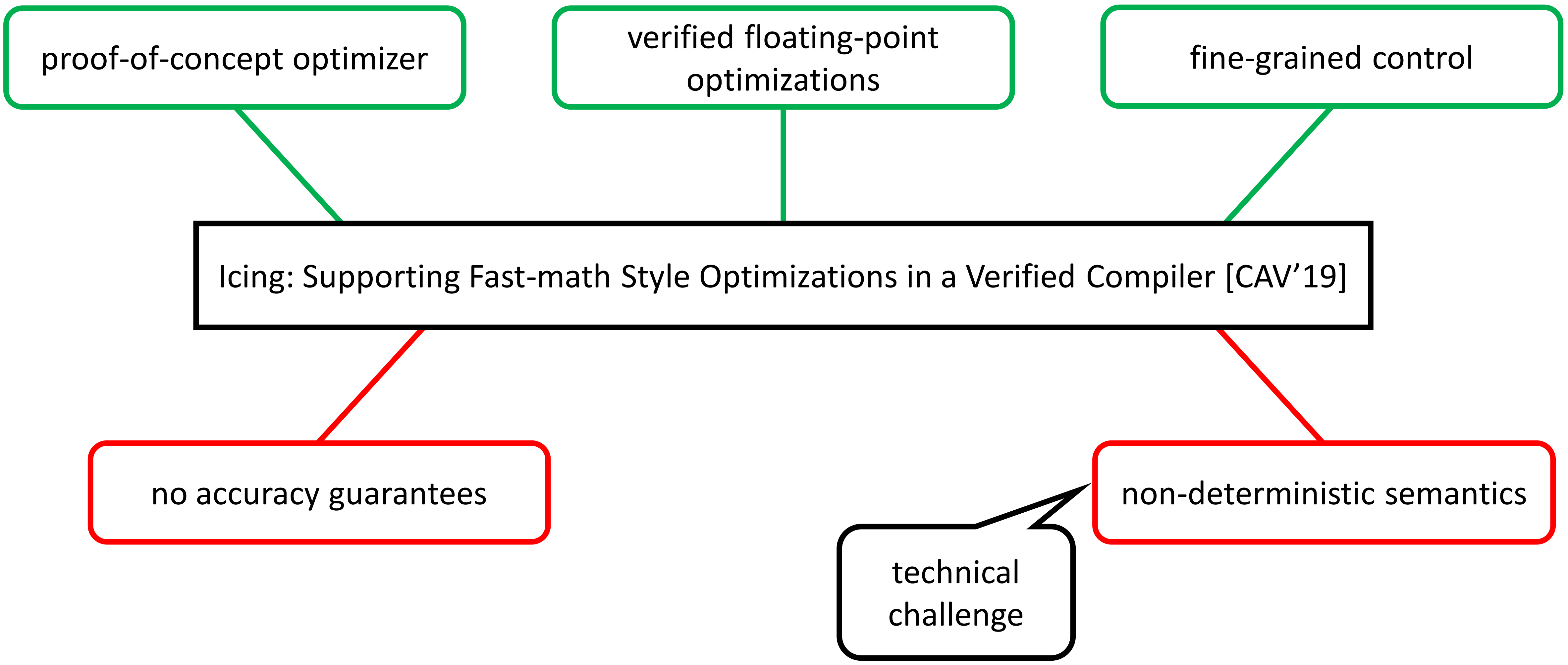
verified floating-point optimizations

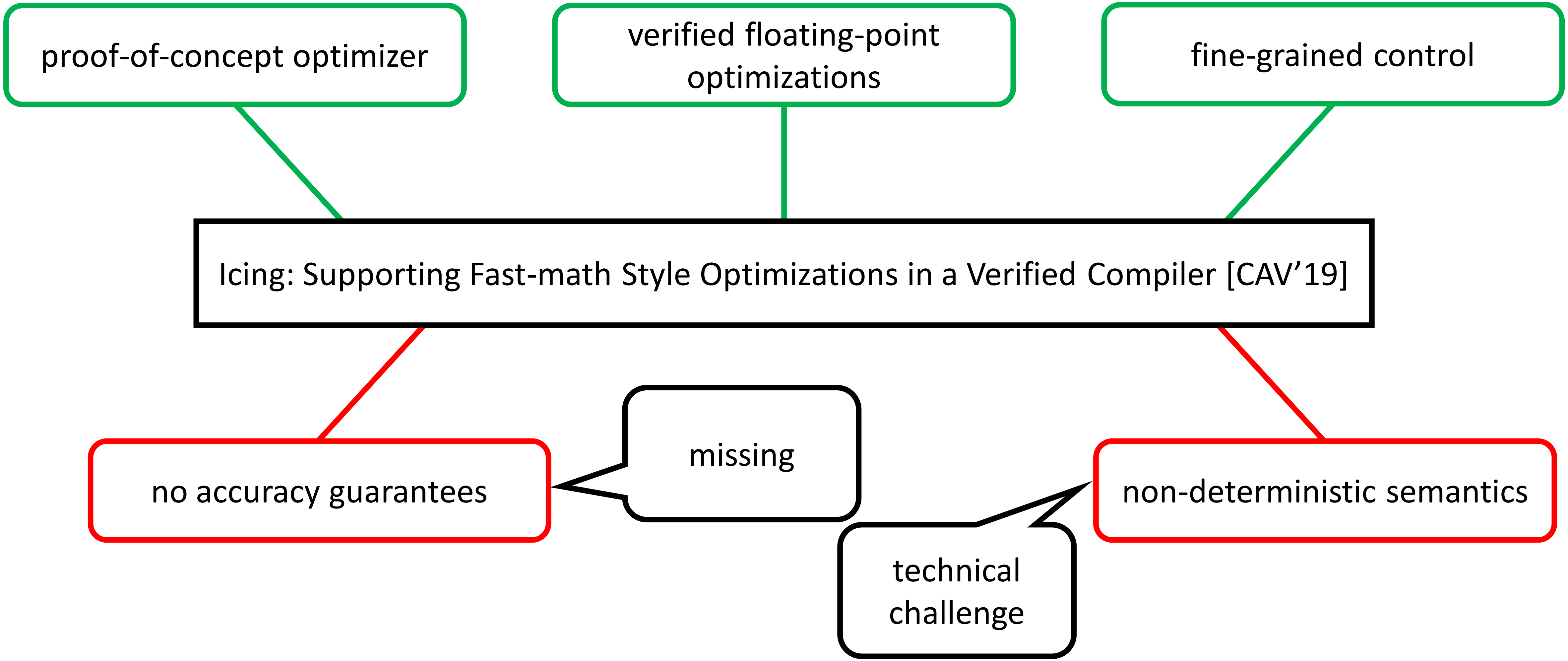
fine-grained control

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV'19]

non-deterministic semantics


proof-of-concept optimizer
verified floating-point optimizations
fine-grained control
Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV'19]
non-deterministic semantics
technical challenge


proof-of-concept optimizer
verified floating-point optimizations
fine-grained control
Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV'19]
no accuracy guarantees
non-deterministic semantics
technical challenge


proof-of-concept optimizer
verified floating-point optimizations
fine-grained control
Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV'19]
no accuracy guarantees
missing
technical challenge
non-deterministic semantics

Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
          1.0 ≤  y ≤100.0)
                 *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```

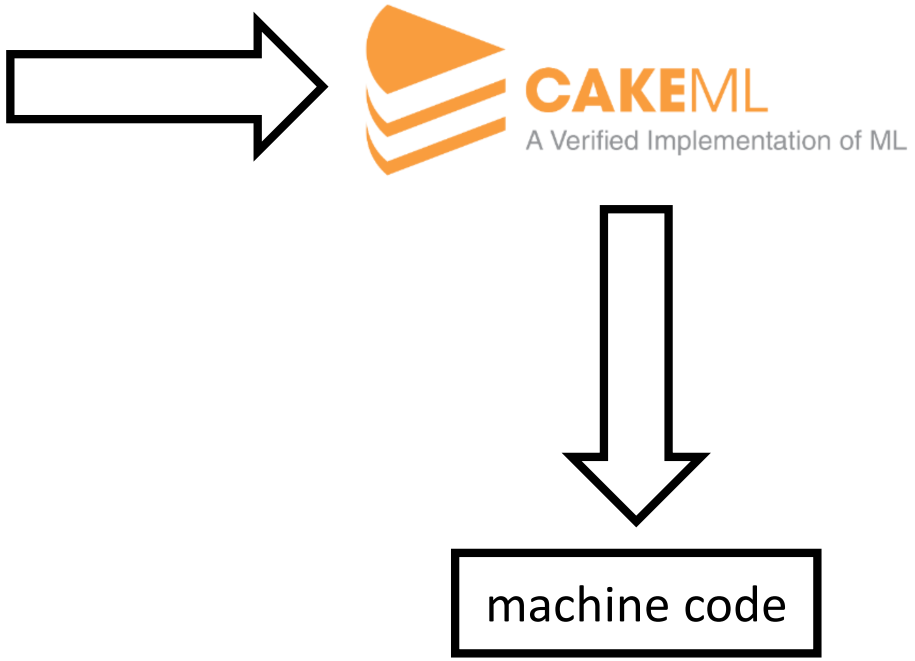
Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
           1.0 ≤  y ≤100.0)
                  *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```

CAKEML
A Verified Implementation of ML

Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
           1.0 ≤  y ≤100.0)
                 *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```



Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
           1.0 ≤ y ≤100.0)
                  *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```

CAKEML
A Verified Implementation of ML

IEEE-754

machine code

Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
          1.0 ≤  y ≤100.0)
                 *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```

CAKEML
A Verified Implementation of ML

machine code

Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
           1.0 ≤ y ≤ 100.0)
output error: 2^-5 *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```

program designed for unavoidable input and output errors

CAKEML
A Verified Implementation of ML

machine code

Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
           1.0 ≤  y ≤100.0)
output error: 2^-5 *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```

program designed for unavoidable
input and output errors

CAKEML
A Verified Implementation of ML

roundoff error ≤
output error

machine code

Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
           1.0 ≤  y ≤100.0)
output error: 2^-5 *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```

program designed for unavoidable
input and output errors

CAKEML
A Verified Implementation of ML

roundoff error ≤
output error

machine code | machine code | machine code

Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧
           1.0 ≤  y ≤100.0)
output error: 2^-5 *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```

program designed for unavoidable
input and output errors

CAKEML
A Verified Implementation of ML

roundoff error ≤
output error

machine code

machine code

machine code

error refinement
any optimized implementation
below output error is fine

Contributions

RealCake:

- extends CakeML with **relaxed non-deterministic floating-point semantics**
- optimizes with a **fast-math optimizer**
- **soundly proves roundoff errors** of floating-point kernels with automated tools
- proves **error refinement**

Contributions

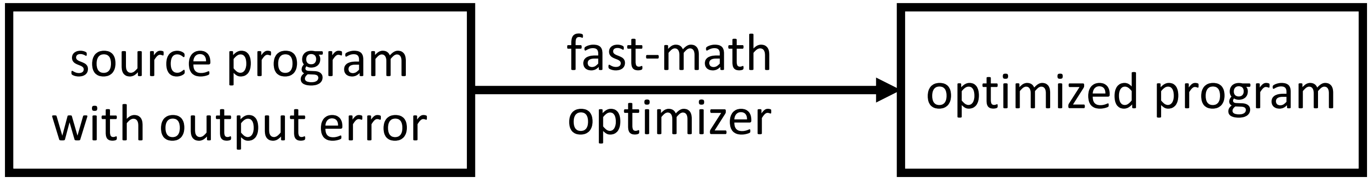
RealCake:

- extends CakeML with **relaxed non-deterministic floating-point semantics**
- optimizes with a **fast-math optimizer**
- **soundly proves roundoff errors** of floating-point kernels with automated tools
- proves **error refinement**

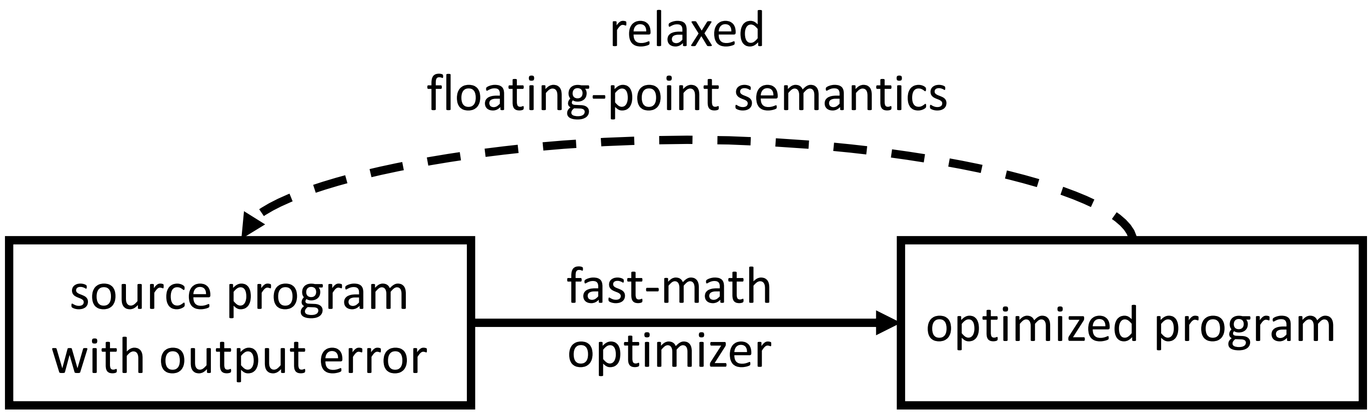
The RealCake Compiler Zoomed In

source program
with output error

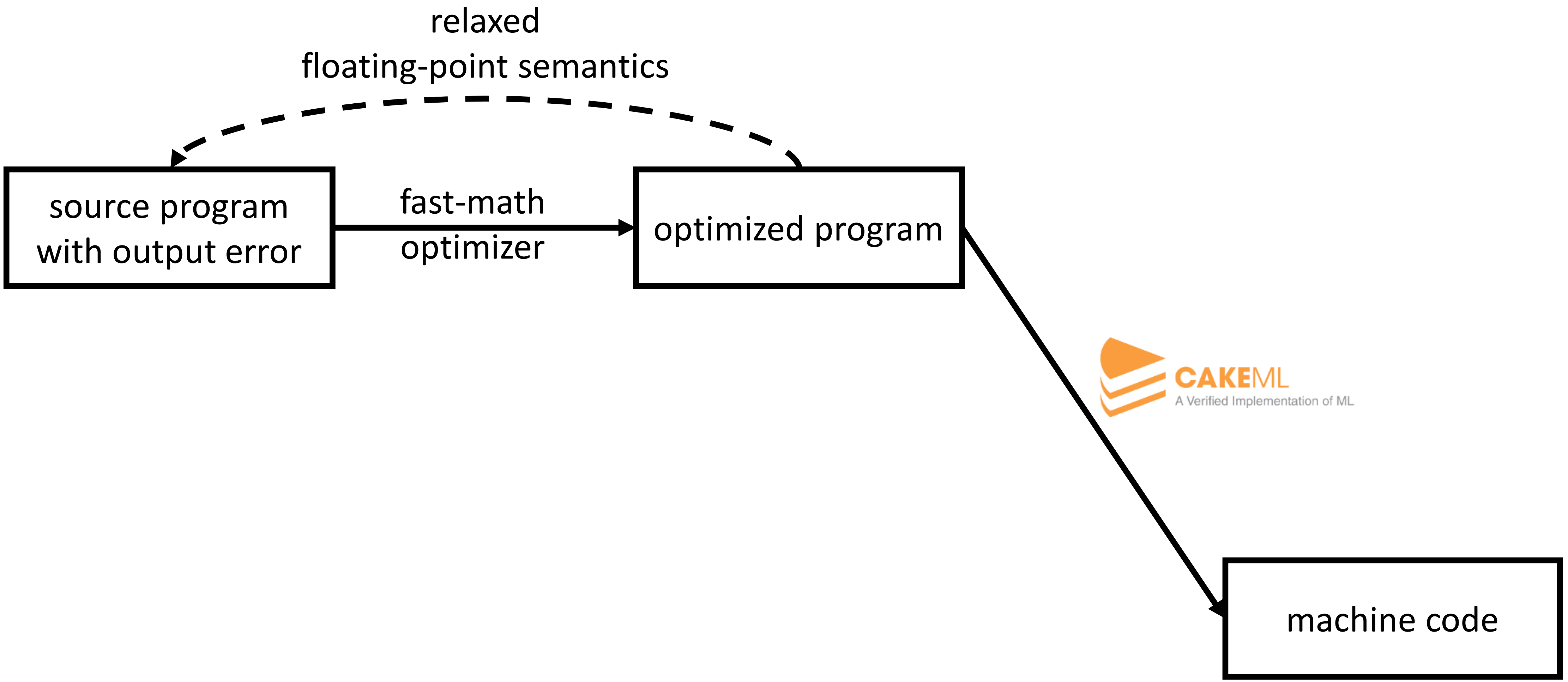
The RealCake Compiler Zoomed In



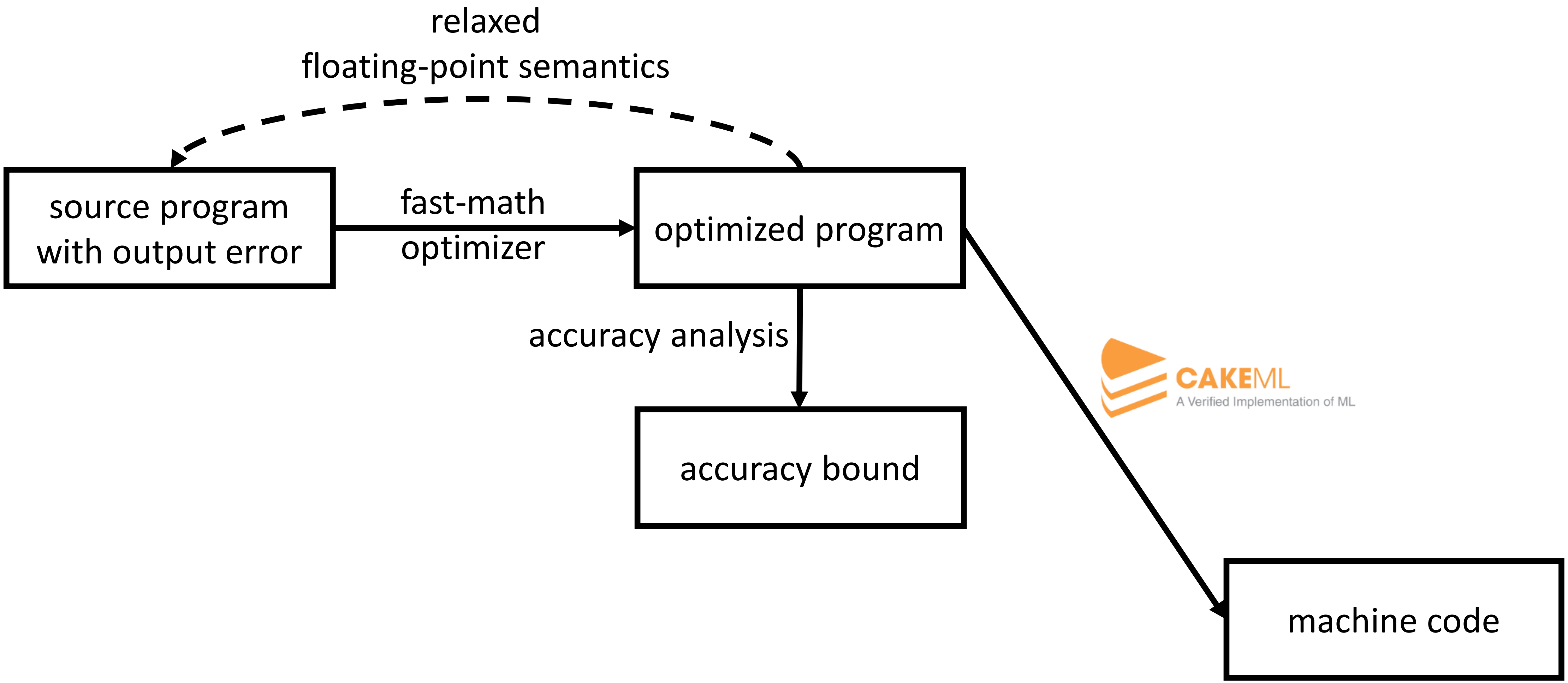
The RealCake Compiler Zoomed In



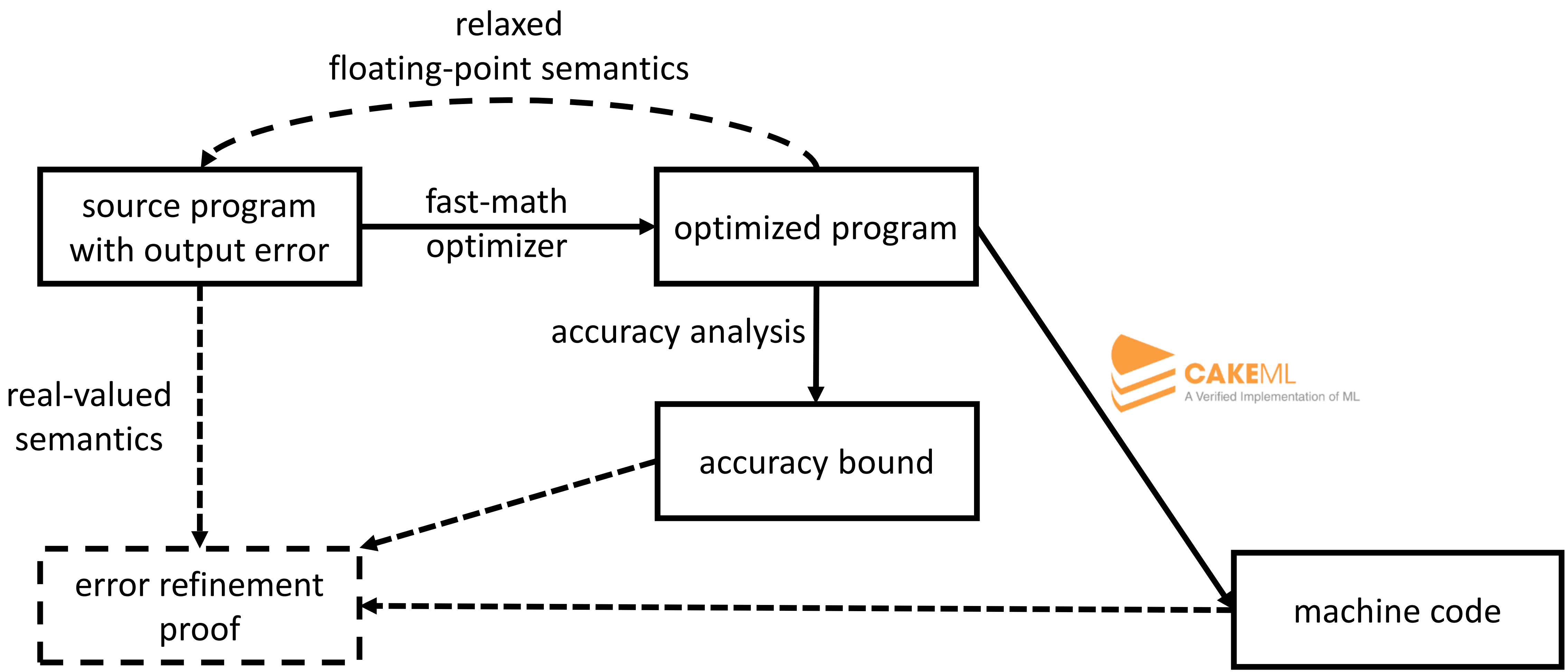
The RealCake Compiler Zoomed In



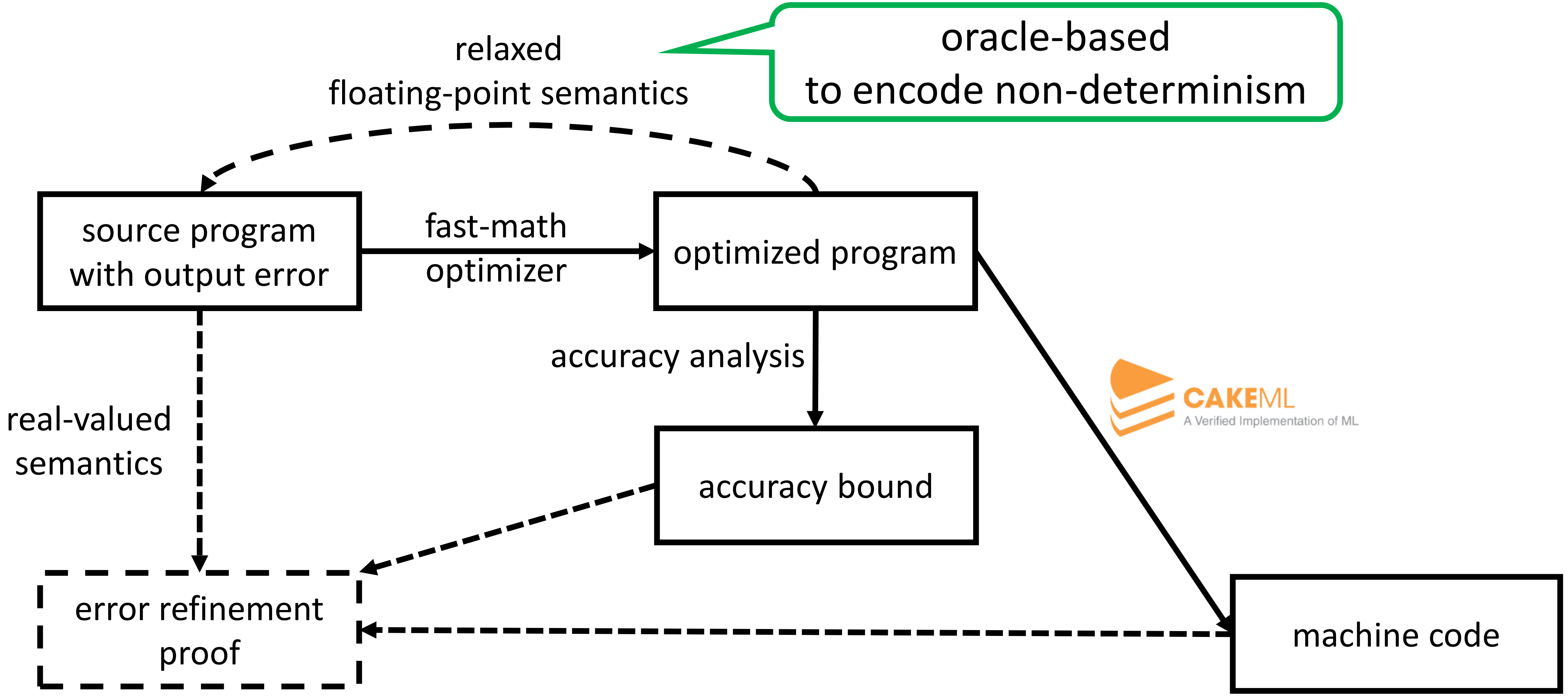
The RealCake Compiler Zoomed In



The RealCake Compiler Zoomed In



The RealCake Compiler Zoomed In



Floating-Point Programs in CakeML

```
fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
    val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
    val d = (x1 * x1) + 1.0
    val s = t / d
    val s2 = t2 / d
  in
    x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) +
    ((x1 * x1) * ((4.0 * s) - 6.0))) * d) +
    (((3.0 * x1) * x1) * s)) +
    ((x1 * x1) * x1)) + x1) + (3.0 * s2))
end)
```

Floating-Point Programs in CakeML

```
fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
    val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
    val d = (x1 * x1) + 1.0
    val s = t / d
    val s2 = t2 / d
  in
    x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) +
    ((x1 * x1) * ((4.0 * s) - 6.0))) * d) +
    (((3.0 * x1) * x1) * s)) +
    ((x1 * x1) * x1)) + x1) + (3.0 * s2))
end)
```

double
operations

Floating-Point Programs in CakeML

```
fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
    val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
    val d = (x1 * x1) + 1.0
    val s = t / d
    val s2 = t2 / d
  in
    x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) +
    ((x1 * x1) * ((4.0 * s) - 6.0))) * d) +
    (((3.0 * x1) * x1) * s)) +
    ((x1 * x1) * x1)) + x1) + (3.0 * s2))
end)
```

optimization annotation

double operations

Floating-Point Programs in CakeML

```
(* output error: 2^-5,
  precondition P: 0.0 ≤ x1 ≤ 5.0 ∧ -20.0 ≤ x2 ≤ 5.0 *)
fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
    val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
    val d = (x1 * x1) + 1.0
    val s = t / d
    val s2 = t2 / d
  in
    x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) +
    ((x1 * x1) * ((4.0 * s) - 6.0))) * d) +
    (((3.0 * x1) * x1) * s)) +
    ((x1 * x1) * x1)) + x1) + (3.0 * s2))
end)
```

optimization annotation

double operations

Floating-Point Programs in CakeML

```
(* output error: 2^-5,
  precondition P: 0.0 ≤ x1 ≤ 5.0 ∧ -20.0 ≤ x2 ≤ 5.0 *)
fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
    val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
    val d = (x1 * x1) + 1.0
    val s = t / d
    val s2 = t2 / d
  in
    x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) +
    ((x1 * x1) * ((4.0 * s) - 6.0))) * d) +
    (((3.0 * x1) * x1) * s)) +
    ((x1 * x1) * x1)) + x1) + (3.0 * s2))
end)
```

tolerable noise

double operations

optimization annotation

Floating-Point Programs in CakeML

```
(* output error: 2^-5,
  precondition P: 0.0 ≤ x1 ≤ 5.0 ∧ -20.0 ≤ x2 ≤ 5.0 *)
fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
    val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
    val d = (x1 * x1) + 1.0
    val s = t / d
    val s2 = t2 / d
  in
    x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) +
    ((x1 * x1) * ((4.0 * s) - 6.0))) * d) +
    (((3.0 * x1) * x1) * s)) +
    ((x1 * x1) * x1)) + x1) + (3.0 * s2))
end)
```

input constraints

optimization annotation

double operations

tolerable noise

Optimized Floating-Point Programs

```
(* guaranteed error bound: 2^-5,
precondition P: 0.0 ≤ x1 ≤ 5.0 ∧ −20.0 ≤ x2 ≤ 5.0 *)
fun jetEngine(x1:double, x2:double):double =
  noopt: (let
    val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
    val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
    val d = fma(x1, x1, 1.0)
    val s = t / d
    val s2 = t2 / d
  in
    x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s,
      x1 * fma(4.0, s, -6.0)),
        fma(x1 * x1, ((s + s) + s) + x1,
          x1 + ((s2 + s2) + s2)))
  end)
```

Optimized Floating-Point Programs

```
(* guaranteed error bound: 2^-5,
precondition P: 0.0 ≤ x1 ≤ 5.0 ∧ -20.0 ≤ x2 ≤ 5.0 *)
fun jetEngine(x1:double, x2:double):double =
  noopt: (let
    val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
    val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
    val d = fma(x1, x1, 1.0)
    val s = t / d
    val s2 = t2 / d
  in
    x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s,
      x1 * fma(4.0, s, -6.0)),
        fma(x1 * x1, ((s + s) + s) + x1,
          x1 + ((s2 + s2) + s2)))
  end)
```

verified by accuracy analysis

Optimized Floating-Point Programs

```
(* guaranteed error bound: 2^-5,
precondition P: 0.0 ≤ x1 ≤ 5.0 ∧ −20.0 ≤ x2 ≤ 5.0 *)
fun jetEngine(x1:double, x2:double):double =
  noopt: (let
    val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
    val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
    val d = fma(x1, x1, 1.0)
    val s = t / d
    val s2 = t2 / d
  in
    x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s,
      x1 * fma(4.0, s, -6.0)),
        fma(x1 * x1, ((s + s) + s) + x1,
          x1 + ((s2 + s2) + s2)))
  end)
```

verified by accuracy analysis

faster, locally more accurate

Optimized Floating-Point Programs

```
(* guaranteed error bound: 2^-5,
precondition P: 0.0 ≤ x1 ≤ 5.0 ∧ −20.0 ≤ x2 ≤ 5.0 *)
fun jetEngine(x1:double, x2:double):double =
  noopt: (let
    val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
    val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
    val d = fma(x1, x1, 1.0)
    val s = t / d
    val s2 = t2 / d
  in
    x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s,
      x1 * fma(4.0, s, -6.0)),
        fma(x1 * x1, ((s + s) + s) + x1,
          x1 + ((s2 + s2) + s2)))
  end)
```

verified by accuracy analysis

faster, locally more accurate

disallow further optimization

Optimized Floating-Point Programs

machine code from IEEE-754 preserving compilation in CakeML

disallow further optimization

```
(* guaranteed error bound: 2^-5,
precondition P: 0.0 ≤ x1 ≤ 5.0 ∧ −20.0 ≤ x2 ≤ 5.0 *)
fun jetEngine(x1:double, x2:double):double =
 noopt: (let
   val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
   val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
   val d = fma(x1, x1, 1.0)
   val s = t / d
   val s2 = t2 / d
 in
   x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s,
     x1 * fma(4.0, s, -6.0)),
       fma(x1 * x1, ((s + s) + s) + x1,
         x1 + ((s2 + s2) + s2)))
 end)
```

verified by accuracy analysis

faster, locally more accurate

The Final Specification Theorem

$$
\begin{array}{l}
\texttt{jetEngineInputsInPrecond}\ (s_1, s_2)\ (w_1, w_2)\ \wedge\ \texttt{environmentOk}\ ([\mathit{jetEngine};\ s_1;\ s_2], \mathit{fs})\ \Rightarrow \\
\exists\, w\ r. \\
\quad \texttt{CakeMLevaluatesAndPrints}\ (\mathit{jetEngineCode}, s_1, s_2, \mathit{fs})\ (\texttt{toString}\ w)\ \wedge \\
\quad \texttt{initialFPcodeReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ w\ \wedge \\
\quad \texttt{realSemanticsReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ r\ \wedge\ \texttt{abs}\ (\texttt{fpToReal}\ w\ -\ r)\ \leq\ 2^{-5}
\end{array}
$$

The Final Specification Theorem

inputs in
specified
constraints

$$\texttt{jetEngineInputsInPrecond}\ (s_1, s_2)\ (w_1, w_2)\ \wedge\ \texttt{environmentOk}\ ([\mathit{jetEngine};\ s_1;\ s_2], \mathit{fs})\ \Rightarrow$$
$$\exists\, w\ r.$$
$$\quad \texttt{CakeMLevaluatesAndPrints}\ (\mathit{jetEngineCode}, s_1, s_2, \mathit{fs})\ (\texttt{toString}\ w)\ \wedge$$
$$\quad \texttt{initialFPcodeReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ w\ \wedge$$
$$\quad \texttt{realSemanticsReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ r\ \wedge\ \texttt{abs}\ (\texttt{fpToReal}\ w\ -\ r)\ \leq\ 2^{-5}$$

The Final Specification Theorem

inputs in specified constraints

the program is run with the correct inputs

$$\texttt{jetEngineInputsInPrecond}\ (s_1, s_2)\ (w_1, w_2)\ \wedge\ \texttt{environmentOk}\ ([\mathit{jetEngine};\ s_1;\ s_2], \mathit{fs})\ \Rightarrow$$
$$\exists\, w\ r.$$
$$\quad \texttt{CakeMLevaluatesAndPrints}\ (\mathit{jetEngineCode}, s_1, s_2, \mathit{fs})\ (\texttt{toString}\ w)\ \wedge$$
$$\quad \texttt{initialFPcodeReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ w\ \wedge$$
$$\quad \texttt{realSemanticsReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ r\ \wedge\ \texttt{abs}\ (\texttt{fpToReal}\ w\ -\ r)\ \leq\ 2^{-5}$$

The Final Specification Theorem

inputs in specified constraints

program returns double word w

the program is run with the correct inputs

$$
\begin{array}{l}
\texttt{jetEngineInputsInPrecond}\ (s_1, s_2)\ (w_1, w_2)\ \wedge\ \texttt{environmentOk}\ ([\mathit{jetEngine};\ s_1;\ s_2], \mathit{fs})\ \Rightarrow \\
\exists\, w\ r. \\
\quad \texttt{CakeMLevaluatesAndPrints}\ (\mathit{jetEngineCode}, s_1, s_2, \mathit{fs})\ (\texttt{toString}\ w)\ \wedge \\
\quad \texttt{initialFPcodeReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ w\ \wedge \\
\quad \texttt{realSemanticsReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ r\ \wedge\ \texttt{abs}\ (\texttt{fpToReal}\ w\ -\ r)\ \leq\ 2^{-5}
\end{array}
$$

The Final Specification Theorem

inputs in specified constraints

program returns double word w

the program is run with the correct inputs

$$
\begin{array}{l}
\texttt{jetEngineInputsInPrecond}\ (s_1, s_2)\ (w_1, w_2)\ \land\ \texttt{environmentOk}\ ([\mathit{jetEngine};\ s_1;\ s_2], \mathit{fs})\ \Rightarrow \\
\exists\, w\ r. \\
\quad \texttt{CakeMLevaluatesAndPrints}\ (\mathit{jetEngineCode}, s_1, s_2, \mathit{fs})\ (\texttt{toString}\ w)\ \land \\
\quad \texttt{initialFPcodeReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ w\ \land \\
\quad \texttt{realSemanticsReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ r\ \land\ \texttt{abs}\ (\texttt{fpToReal}\ w\ -\ r)\ \leq\ 2^{-5}
\end{array}
$$

w also result of nondeter ministic semantics

The Final Specification Theorem

inputs in specified constraints

program returns double word w

the program is run with the correct inputs

$$
\begin{array}{l}
\texttt{jetEngineInputsInPrecond}\ (s_1, s_2)\ (w_1, w_2)\ \wedge\ \texttt{environmentOk}\ ([\mathit{jetEngine};\ s_1;\ s_2], \mathit{fs})\ \Rightarrow \\
\exists\, w\ r. \\
\quad \texttt{CakeMLevaluatesAndPrints}\ (\mathit{jetEngineCode}, s_1, s_2, \mathit{fs})\ (\texttt{toString}\ w)\ \wedge \\
\quad \texttt{initialFPcodeReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ w\ \wedge \\
\quad \texttt{realSemanticsReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ r\ \wedge\ \texttt{abs}\ (\texttt{fpToReal}\ w\ -\ r)\ \leq\ 2^{-5}
\end{array}
$$

w also result of nondeter ministic semantics

real-number semantics returns r

The Final Specification Theorem

inputs in specified constraints

program returns double word w

the program is run with the correct inputs

$$
\begin{array}{l}
\texttt{jetEngineInputsInPrecond}\ (s_1, s_2)\ (w_1, w_2)\ \wedge\ \texttt{environmentOk}\ ([\mathit{jetEngine};\ s_1;\ s_2], \mathit{fs})\ \Rightarrow \\
\exists\, w\ r. \\
\quad \texttt{CakeMLevaluatesAndPrints}\ (\mathit{jetEngineCode}, s_1, s_2, \mathit{fs})\ (\texttt{toString}\ w)\ \wedge \\
\quad \texttt{initialFPcodeReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ w\ \wedge \\
\quad \texttt{realSemanticsReturns}\ \mathit{jetEngineUnopt}\ (w_1, w_2)\ r\ \wedge\ \texttt{abs}\ (\texttt{fpToReal}\ w\ -\ r)\ \leq\ 2^{-5}
\end{array}
$$

w also result of nondeterministic semantics

real-number semantics returns r

output error sound

The Final Specification Theorem

inputs in specified constraints

program returns double word w

the program is run with the correct inputs

`jetEngineInputsInPrec` ... $tEngine;\ s_1;\ s_2], fs) \Rightarrow$

$\exists\, w\ r.$

`CakeMLevaluatesAndP` ... $w) \wedge$

`initialFPcodeReturn` ...

`realSemanticsReturn` ... `Real` $w - r) \leq 2^{-5}$

RealCake is the first verified compiler that proves end-to-end accuracy bounds for **compiled** fast-math optimized programs

w also result of nondeter ministic semantics

real-number semantics returns r

output error sound

RealCake’s Four-Phase-Optimizer

canonical form

RealCake's Four-Phase-Optimizer

canonical form

$$x - y \rightarrow x + ((-1) \times y)$$
$$((x + y) + z \rightarrow x + (y + z)$$

move constants to right-hand sides

RealCake’s Four-Phase-Optimizer

canonical form → undistribute

$$x - y \rightarrow x + ((-1) \times y)$$
$$((x + y) + z \rightarrow x + (y + z)$$

move constants to right-hand sides

RealCake's Four-Phase-Optimizer

canonical form → undistribute

$(x \times y) + (x \times z) \rightarrow x \times (y + z)$

$x - y \rightarrow x + ((-1) \times y)$

$((x + y) + z \rightarrow x + (y + z)$

move constants to right-hand sides

RealCake's Four-Phase-Optimizer

canonical form → undistribute → peephole optimizer

$$(x \times y) + (x \times z) \rightarrow x \times (y + z)$$

$$x - y \rightarrow x + ((-1) \times y)$$

$$((x + y) + z \rightarrow x + (y + z)$$

move constants to right-hand sides

RealCake's Four-Phase-Optimizer

canonical form → undistribute → peephole optimizer

$x - y \to x + ((-1) \times y)$
$((x + y) + z \to x + (y + z)$
move constants to right-hand sides

$(x \times y) + (x \times z) \to x \times (y + z)$

$x \times 0 \to 0$
$x \times 1 \to x$
$x \times -1 \to -x$
$x \times 2 \to x + x$
$x \times 3 \to x + (x + x)$
$x * y + z \to \mathrm{fma}(x, y, z)$ …

RealCake's Four-Phase-Optimizer

canonical form → undistribute → peephole optimizer → balance trees

$$x - y \rightarrow x + ((-1) \times y)$$
$$((x + y) + z \rightarrow x + (y + z)$$
move constants to right-hand sides

$$(x \times y) + (x \times z) \rightarrow x \times (y + z)$$

$$x \times 0 \rightarrow 0$$
$$x \times 1 \rightarrow x$$
$$x \times -1 \rightarrow -x$$
$$x \times 2 \rightarrow x + x$$
$$x \times 3 \rightarrow x + (x + x)$$
$$x * y + z \rightarrow \mathrm{fma}(x, y, z) \ldots$$

RealCake's Four-Phase-Optimizer

canonical form → undistribute → peephole optimizer → balance trees

canonical form:

$$x - y \to x + ((-1) \times y)$$
$$((x + y) + z \to x + (y + z)$$

move constants to right-hand sides

undistribute:

$$(x \times y) + (x \times z) \to x \times (y + z)$$

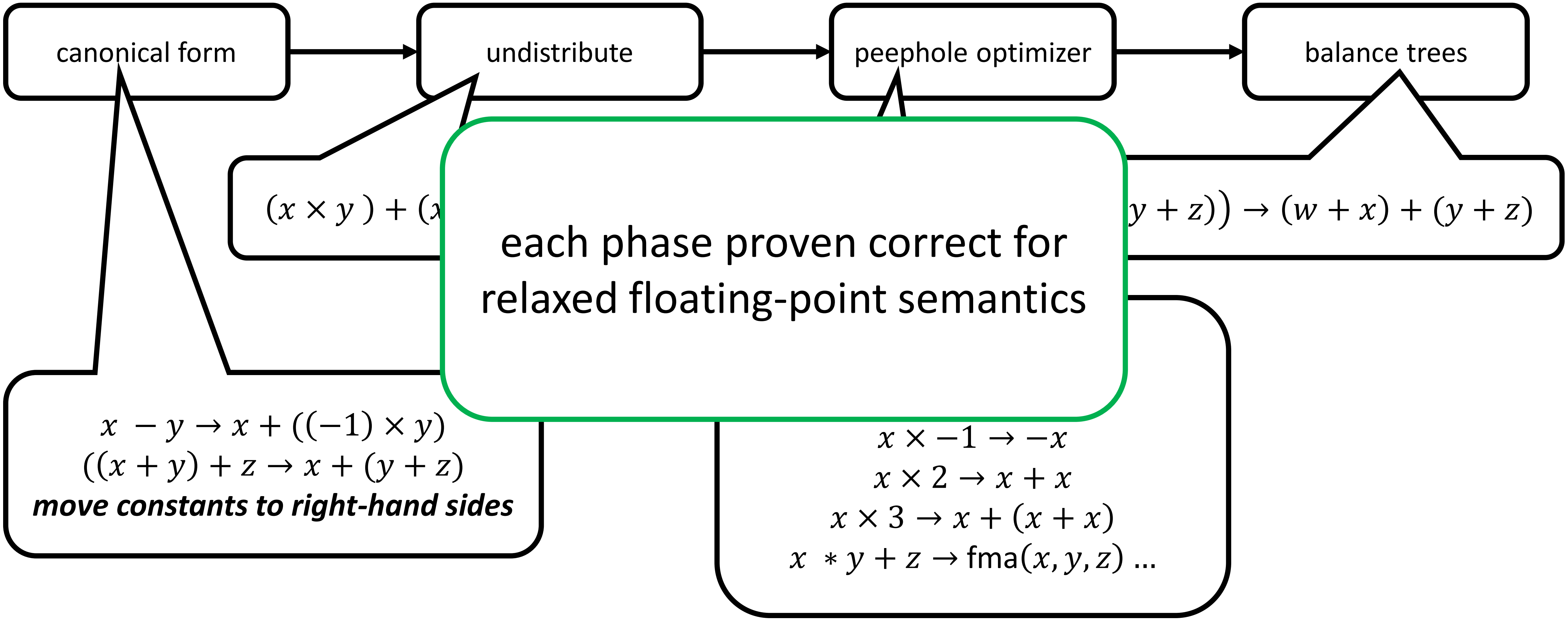
peephole optimizer:

$$x \times 0 \to 0$$
$$x \times 1 \to x$$
$$x \times -1 \to -x$$
$$x \times 2 \to x + x$$
$$x \times 3 \to x + (x + x)$$
$$x * y + z \to \mathrm{fma}(x, y, z) \ldots$$

balance trees:

$$w + \big(x + (y + z)\big) \to (w + x) + (y + z)$$

RealCake's Four-Phase-Optimizer



Experimental Results – Roundoff Errors

Name	
cartToPol	
delta	
doppler2	
pid	
sine_newton	
sqroot	
turbine1	

Experimental Results – Roundoff Errors

Name	Original
cartToPol	2.815e-09
delta	1.970e-13
doppler2	6.534e-13
pid	7.621e-15
sine_newton	7.495e-15
sqroot	1.115e-15
turbine1	1.588e-13

Experimental Results – Roundoff Errors

Name	Original	fast-math
cartToPol	2.815e-09	2.463e-09
delta	1.970e-13	2.940e-12
doppler2	6.534e-13	1.639e-12
pid	7.621e-15	7.727e-15
sine_newton	7.495e-15	6.27e-15
sqroot	1.115e-15	1.059e-15
turbine1	1.588e-13	1.541e-13

Experimental Results – Roundoff Errors

Name	Original	fast-math	Improvement
cartToPol	2.815e-09	2.463e-09	13%
delta	1.970e-13	2.940e-12	-198%
doppler2	6.534e-13	1.639e-12	50%
pid	7.621e-15	7.727e-15	-1%
sine_newton	7.495e-15	6.27e-15	16%
sqroot	1.115e-15	1.059e-15	5%
turbine1	1.588e-13	1.541e-13	3%

Experimental Results – Roundoff Errors

Name	Original	fast-math	Improvement
cartToPol	2.815e-09	2.463e-09	13%
delta	1.970e-13	2.940e-12	-198%
doppler2	6.534e-13	1.639e-12	50%
pid	7.621e-15	7.727e-15	-1%
sine_newton	7.495e-15	6.27e-15	16%
sqroot	1.115e-15	1.059e-15	5%
turbine1	1.588e-13	1.541e-13	3%

trade accuracy for performance

Experimental Results – Performance

Name
cartToPol
delta
doppler2
pid
sine_newton
sqroot
turbine1

Experimental Results – Performance

Name	Original
cartToPol	2.05
delta	13.49
doppler2	36.00
pid	104.11
sine_newton	126.34
sqroot	87.06
turbine1	121.02

Experimental Results – Performance

Name	Original	fast-math
cartToPol	2.05	9%
delta	13.49	16%
doppler2	36.00	6%
pid	104.11	0%
sine_newton	126.34	0%
sqroot	87.06	5%
turbine1	121.02	0%

Experimental Results – Performance

Name	Original	Csts	fast-math
cartToPol	2.05	1%	9%
delta	13.49	1%	16%
doppler2	36.00	91%	6%
pid	104.11	96%	0%
sine_newton	126.34	92%	0%
sqroot	87.06	95%	5%
turbine1	121.02	96%	0%

Conclusion

RealCake:

- proves **error refinement for CakeML programs**
- extends CakeML with **oracle-based floating-point semantics**
- optimizes with **fast-math-style optimizations**

Conclusion

RealCake:

- proves **error refinement for CakeML programs**
- extends CakeML with **oracle-based floating-point semantics**
- optimizes with **fast-math-style optimizations**
- is **integrated into official CakeML** codebase: https://code.cakeml.org

Conclusion

RealCake:

- proves **error refinement for CakeML programs**
- extends CakeML with **oracle-based floating-point semantics**
- optimizes with **fast-math-style optimizations**
- is **integrated into official CakeML** codebase: https://code.cakeml.org

in the paper:

- verified constant lifting optimization
- heuristic to avoid slow-downs
- integration into CakeML toolchain
- implementation of real-numbered and IEEE-754 semantics