Certificate Checking Static Analyses of Mixed-Precision Floating-Point Arithmetic

Raphaël Monat

Intern in the Max Planck Institute for Software Systems (Saarbrücken, Germany)
Under the supervision of Eva Darulová

Introduction

Designing floating-point programs is hard

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+2} = 111 - \dfrac{1130}{u_{n+1}} + \dfrac{3000}{u_{n+1} \cdot u_n} \end{cases}$$

Example taken from Muller et al., *Handbook of Floating-Point Arithmetic*

Designing floating-point programs is hard

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+2} = 111 - \dfrac{1130}{u_{n+1}} + \dfrac{3000}{u_{n+1} \cdot u_n} \end{cases}$$

$$\forall n \geq 2, u_n \geq 6$$

Example taken from Muller et al., *Handbook of Floating-Point Arithmetic*

Designing floating-point programs is hard

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+2} = 111 - \dfrac{1130}{u_{n+1}} + \dfrac{3000}{u_{n+1} \cdot u_n} \end{cases}$$

$$\forall n \geq 2, u_n \geq 6$$

$$\lim_{n \to +\infty} u_n = 6$$

Example taken from Muller et al., *Handbook of Floating-Point Arithmetic*

Designing floating-point programs is hard

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+2} = 111 - \dfrac{1130}{u_{n+1}} + \dfrac{3000}{u_{n+1} \cdot u_n} \end{cases}$$

$$\forall n \geq 2, u_n \geq 6$$

$$\lim_{n \to +\infty} u_n = 6$$

n	$\widetilde{u_n}$ (128-bits)
2	18.5
3	9.38
4	7.80
5	7.15
6–29	$\simeq 6$
30	**5.88**
31	3.90
32	-47.97
33	118.52
34–...	$\simeq 100$

Example taken from Muller et al., *Handbook of Floating-Point Arithmetic*

Designing floating-point programs is hard

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+2} = 111 - \dfrac{1130}{u_{n+1}} + \dfrac{3000}{u_{n+1} \cdot u_n} \end{cases}$$

$$\forall n \geq 2, u_n \geq 6$$

$$\lim_{n \to +\infty} u_n = 6$$

n	$\widetilde{u_n}$ (128-bits)
2	18.5
3	9.38
4	7.80
5	7.15
6–29	$\simeq 6$
30	5.88
31	3.90
32	-47.97
33	118.52
34–...	$\simeq$ **100**

Example taken from Muller et al., *Handbook of Floating-Point Arithmetic*

Designing floating-point programs is hard

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+2} = 111 - \dfrac{1130}{u_{n+1}} + \dfrac{3000}{u_{n+1} \cdot u_n} \end{cases}$$

n	$\widetilde{u_n}$ (128-bits)
2	18.5
3	9.38
4	7.80
5	7.15
6–29	$\simeq 6$
33	118.52
34–...	$\simeq 100$

$$\left| u_N - \widetilde{u_N} \right| = \mathbf{94}$$

$$\lim_{n \to +\infty} u_n = 6$$

Example taken from Muller et al., *Handbook of Floating-Point Arithmetic*

Designing floating-point programs is hard

- Roundoff errors ($|u_n - \widetilde{u_n}|$) $\Longrightarrow$ catastrophic results

Designing floating-point programs is hard

- Roundoff errors ($|u_n - \widetilde{u_n}|$) $\Longrightarrow$ catastrophic results
- Real algebraic equalities do not hold anymore

Designing floating-point programs is hard

- Roundoff errors ($|u_n - \widetilde{u_n}|$) $\Longrightarrow$ catastrophic results
- Real algebraic equalities do not hold anymore
- Performance - precision tradeoff

Designing floating-point programs is hard

- Roundoff errors ($|u_n - \widetilde{u_n}|$) $\Longrightarrow$ catastrophic results
- Real algebraic equalities do not hold anymore
- Performance - precision tradeoff

$\Longrightarrow$ automatic tools to help developers

Static analysis of floating-point computations

- Uses only the source code, no need to run the input program

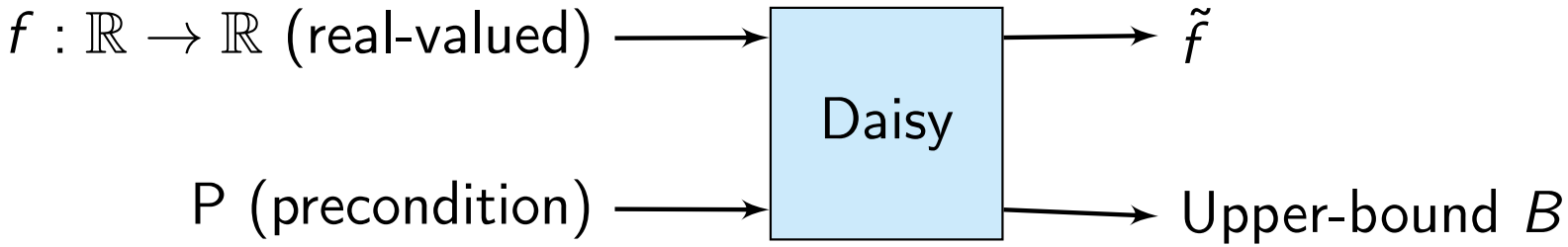
Static analysis of floating-point computations

- Uses only the source code, no need to run the input program
- Upper-bounds the roundoff error $|P(x) - P(\tilde{x})|$

Static analysis of floating-point computations

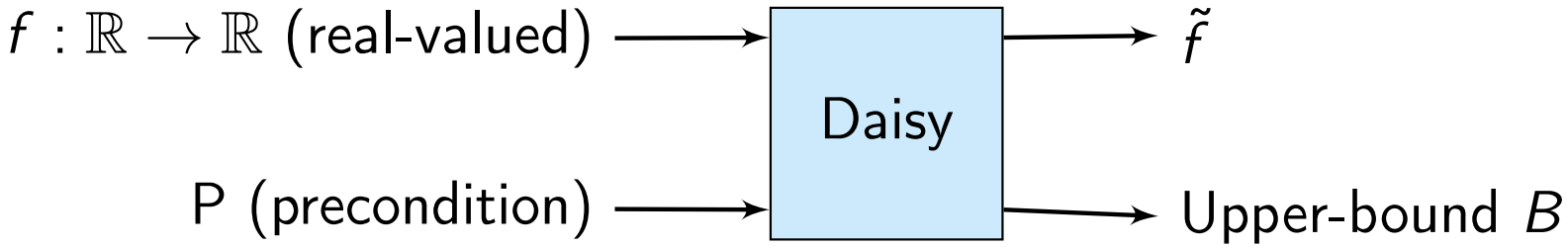
- Uses only the source code, no need to run the input program
- Upper-bounds the roundoff error $|P(x) - P(\tilde{x})|$
- Soundness property: computes over-approximations of roundoff errors

Daisy, a compiler of real programs



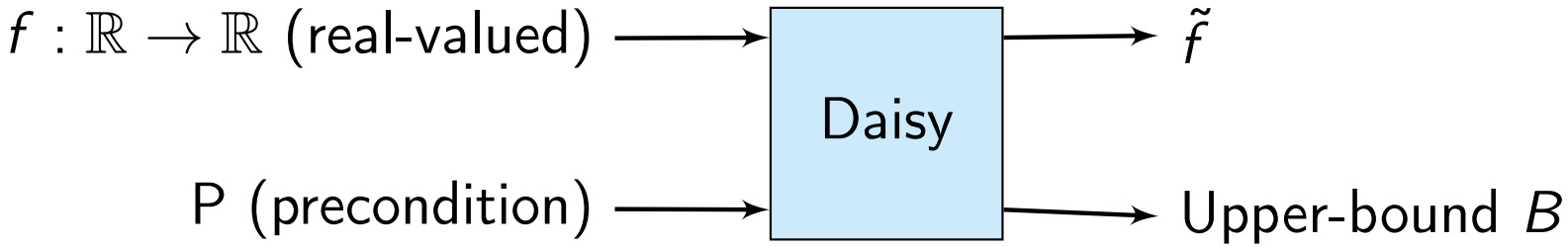
- Soundness: $B \geq \max_{x \in P} \left| f(x) - \tilde{f}(\tilde{x}) \right|$

Daisy, a compiler of real programs



- Soundness: $B \geq \max_{x \in P} \left| f(x) - \tilde{f}(\tilde{x}) \right|$
- Contains a static analysis phase

Daisy, a compiler of real programs



- Soundness: $B \geq \max_{x \in P} \left| f(x) - \tilde{f}(\tilde{x}) \right|$
- Contains a static analysis phase
- Correct optimizations of real-valued programs

Bugs in static analyzers

- The paper-proof of soundness may contain errors
- An automatic tool might contain implementation errors

Bugs in static analyzers

- The paper-proof of soundness may contain errors
- An automatic tool might contain implementation errors

$\Longrightarrow$ need more confidence in the results, use proof assistants

Bugs in static analyzers

- The paper-proof of soundness may contain errors
- An automatic tool might contain implementation errors

$\Longrightarrow$ need more confidence in the results, use proof assistants

Example: Verasco is a general static analyzer written in Coq.

Proof assistants

- Can define objects (sets, relations, functions) and theorems

Proof assistants

- Can define objects (sets, relations, functions) and theorems
- Proof checker (no skipped case, ...) using a small kernel (verified by hand)

Proof assistants

- Can define objects (sets, relations, functions) and theorems
- Proof checker (no skipped case, ...) using a small kernel (verified by hand)
- High confidence in the proved results

Proof assistants

- Can define objects (sets, relations, functions) and theorems
- Proof checker (no skipped case, ...) using a small kernel (verified by hand)
- High confidence in the proved results

I used both Coq and HOL4.

Mixed-precision

```
def sine(a: Double) =
  val t: Double = - a * a * a / 6
  val u: Double = a * a * a * a * a / 120
  a + t + u
```

$5.11 \cdot 10^{-16}$

Mixed-precision

```
def sine(a: Double) =
  val t: Double = − a * a * a / 6
  val u: Double = a * a * a * a * a / 120
  a + t + u
```

$5.11 \cdot 10^{-16}$

```
def sine(a: DoubleDouble) =
  val t: DoubleDouble = − a * a * a / 6
  val u: DoubleDouble = a * a * a * a * a / 120
  a + t + u
```

$4.26 \cdot 10^{-34}$

but slow

Mixed-precision

```
def sine(a: Double) =
  val t: Double = − a * a * a / 6
  val u: Double = a * a * a * a * a / 120
  a + t + u
```

$5.11 \cdot 10^{-16}$

```
def sine(a: DoubleDouble) =
  val t: DoubleDouble = − a * a * a / 6
  val u: DoubleDouble = a * a * a * a * a / 120
  a + t + u
```

$4.26 \cdot 10^{-34}$
but slow

```
def sine(a: Double) =
  val t: DoubleDouble = − a * a * a / 6
  val u: Double = a * a * a * a * a / 120
  a + t + u
```

$2.51 \cdot 10^{-16}$

Contribution

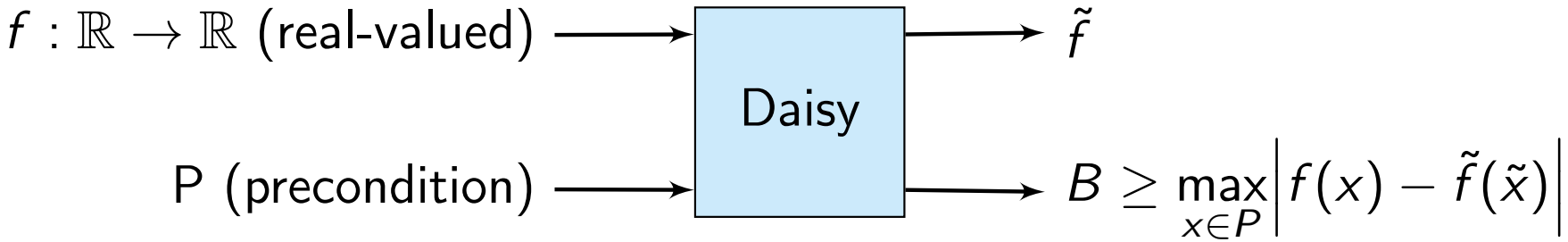
Before, Daisy supported:

- formal validation of uniform-precision analyses
- mixed-precision analyses

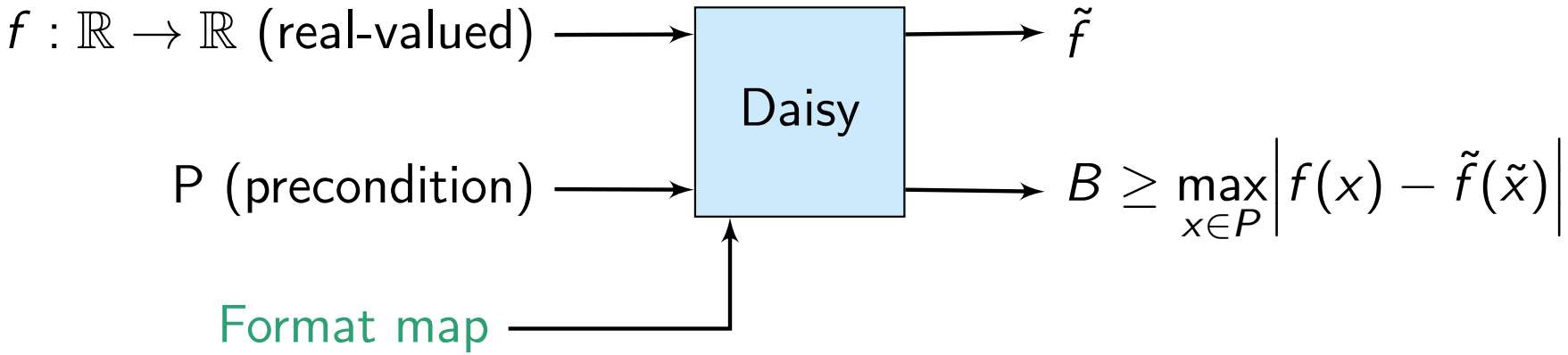
Contribution: I extended the formal validation to mixed-precision.

Daisy

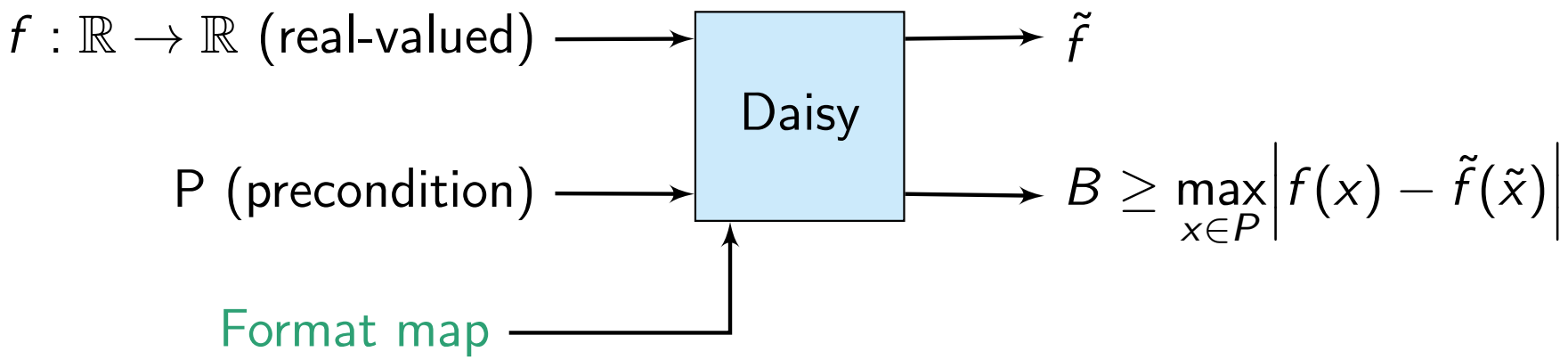
Extending Daisy to Mixed-Precision



Extending Daisy to Mixed-Precision



Extending Daisy to Mixed-Precision



Technical details:

- Written in Scala
- Supports variable bindings (no control flow)
- Analysis using arbitrary-precision rationals

How Daisy works – example

```
def sine(a:Real) = {
  require(-1 <= a && a <= 1)
  val three = - a*a*a / 6
  val five = a*a*a*a*a / 120
  a + three + five
}
```

How Daisy works – example

```
def sine(a:Real) = {
  require(-1 <= a && a <= 1)
  val three = - a*a*a / 6
  val five = a*a*a*a*a / 120
  a + three + five
}

sine = {
     a:Double
     three:DoubleDouble
     five:Double
}
```

How Daisy works – example

```
def sine(a:Real) = {
  require(-1 <= a && a <= 1)
  val three = - a*a*a / 6
  val five = a*a*a*a*a / 120
  a + three + five
}

sine = {
     a:Double
     three:DoubleDouble
     five:Double
}
```

```
object Sine {
  /*@pre : ((-1. <= a) && (a <= 1.))*/
  def sine(a: Double): DoubleDouble = {
    val three: DoubleDouble = -a * a * a / 6
    val five: Double = a * a * a * a * a / 120
    ((a + three) + five)
  }
}
```

How Daisy works – example

```
def sine(a:Real) = {
  require(-1 <= a && a <= 1)
  val three = - a*a*a / 6
  val five = a*a*a*a*a / 120
  a + three + five
}

sine = {
     a:Double
     three:DoubleDouble
     five:Double
}
```

```
object Sine {
  /*@pre : ((-1. <= a) && (a <= 1.))*/
  def sine(a: Double): DoubleDouble = {
    val three: DoubleDouble = -a * a * a / 6
    val five: Double = a * a * a * a * a / 120
    ((a + three) + five)
  }
}
```

```
abs-error: 2.507253663945146e-16,
real range: [-1.175, 1.175]
```

How Daisy works – Analysis

Two-step analysis:

- Real range analysis `real range: [-1.175, 1.175]`
- Roundoff error analysis `abs-error: 2.51e-16`

Range analysis using Interval Arithmetic

Interval arithmetic

$$[a; b] + [c; d] = [a + c; b + d]$$

$$[a; b] \times [c; d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$$

Range analysis using Interval Arithmetic

Interval arithmetic

$$[a; b] + [c; d] = [a + c; b + d]$$
$$[a; b] \times [c; d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$$

Soundness of interval arithmetic

For $\circ \in \{+, -, \times, /\}$, $\circ$ should **over-approximate** $\circ$, i.e:

$$x \in [a; b] \wedge y[c; d] \implies x \circ y \in [a; b] \circ [c; d]$$

Floating-point numbers (Normalized and normal)

$$1.m_1 m_2 \ldots m_{p-1} \cdot \pm 2^e \qquad e_{min} \leq e \leq e_{max}$$

Floating-point arithmetic

Floating-point numbers (Normalized and normal)

$$1.m_1 m_2 \ldots m_{p-1} \cdot \pm 2^e \qquad e_{min} \leq e \leq e_{max}$$

Rounding

Round-to-nearest, ties to even: $RN(x)$.

Floating-point arithmetic

Floating-point numbers (Normalized and normal)

$$1.m_1 m_2 \ldots m_{p-1} \cdot \pm 2^e \qquad e_{min} \leq e \leq e_{max}$$

Rounding

Round-to-nearest, ties to even: $\tilde{x}$.

Floating-point arithmetic

Floating-point numbers (Normalized and normal)

$$1.m_1 m_2 \ldots m_{p-1} \cdot \pm 2^e \qquad e_{min} \leq e \leq e_{max}$$

Rounding

Round-to-nearest, ties to even: $\tilde{x}$.

$1 + \delta$ **abstraction**

$$RN(x) = x \cdot (1 + \delta) \qquad |\delta| \leq 2^{-p}$$

Floating-point arithmetic

Floating-point numbers (Normalized and normal)

$$1.m_1 m_2 \ldots m_{p-1} \cdot \pm 2^e \qquad e_{min} \leq e \leq e_{max}$$

Rounding

Round-to-nearest, ties to even: $\tilde{x}$.

$1 + \delta$ **abstraction**

$$\tilde{x} = x \cdot (1 + \delta) \qquad |\delta| \leq 2^{-p}$$

Floating-point arithmetic

Floating-point numbers (Normalized and normal)

$$1.m_1 m_2 \ldots m_{p-1} \cdot \pm 2^e \qquad e_{min} \leq e \leq e_{max}$$

Rounding

Round-to-nearest, ties to even: $\tilde{x}$.

$1 + \delta$ **abstraction**

$$\tilde{x} = x \cdot (1 + \delta) \qquad |\delta| \leq 2^{-p}$$

So: $x \circ_{FP} y = (x \circ y) \cdot (1 + \delta)$

Floating-point arithmetic – Mixed-precision

Format	Precision (p)	e_{min}	e_{max}
32-bits	24	-126	+127
64-bits	53	-1022	+1023
128-bits	113	-16382	+16382

Floating-point arithmetic – Mixed-precision

Format	Precision (p)	e_{min}	e_{max}
32-bits	24	-126	+127
64-bits	53	-1022	+1023
128-bits	113	-16382	+16382

Downcast operator

```
val x: Float = (y: Double). toFloat
```

Floating-point arithmetic – Mixed-precision

Format	Precision (p)	e_{min}	e_{max}
32-bits	24	-126	+127
64-bits	53	-1022	+1023
128-bits	113	-16382	+16382

`Downcast` operator

```
val x: Float = (y: Double). toFloat
```

No need for `Upcast`

Floating-point arithmetic – Mixed-precision

Format	Precision (p)	e_{min}	e_{max}
32-bits	24	-126	+127
64-bits	53	-1022	+1023
128-bits	113	-16382	+16382

`Downcast` operator

```
val x: Float = (y: Double). toFloat
```

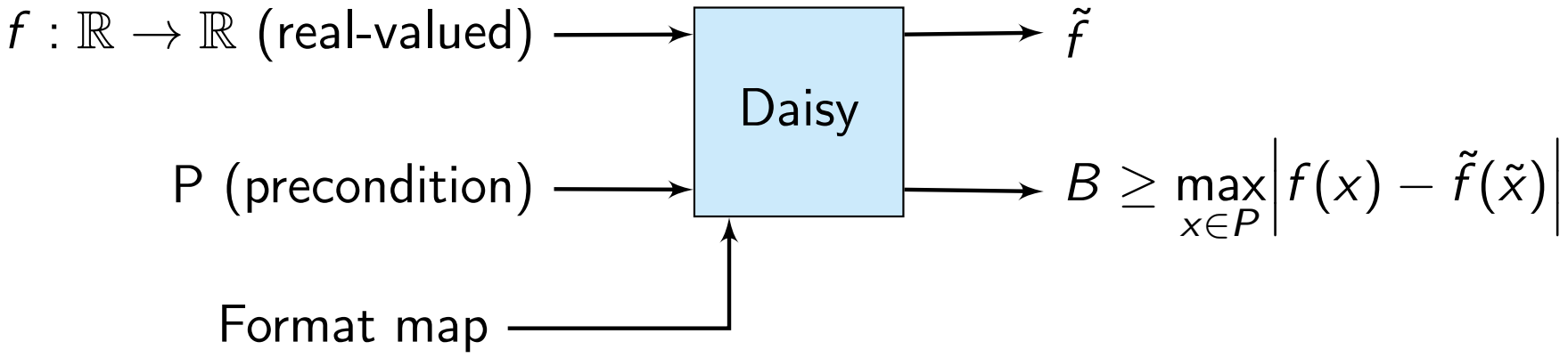
No need for `Upcast`

$1 + \delta$ abstraction for `Downcast`

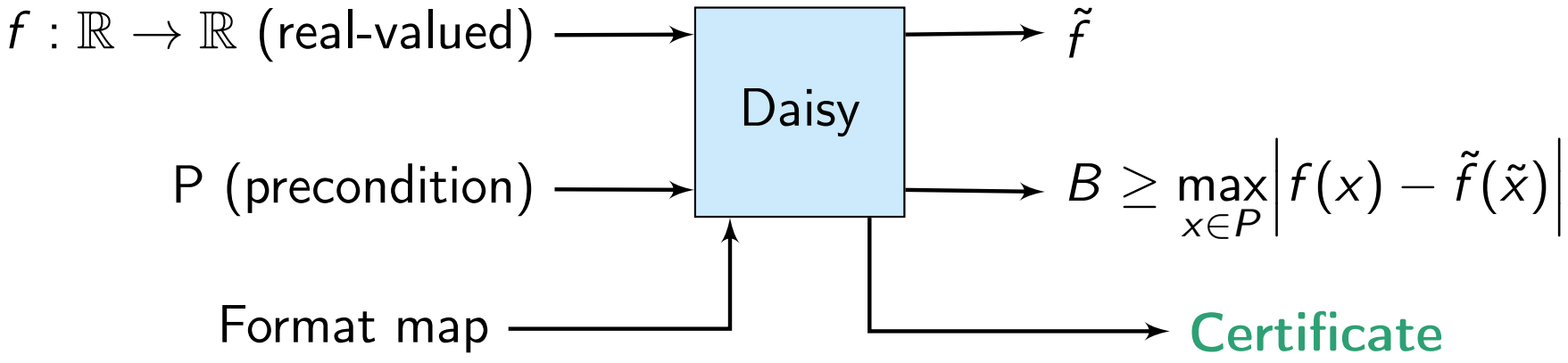
$$\texttt{Downcast}(32, x_{64}) = x_{64} \cdot (1 + \delta) \qquad |\delta| \leq 2^{-p_{32}} = 2^{-24}$$

Formal verification of Daisy

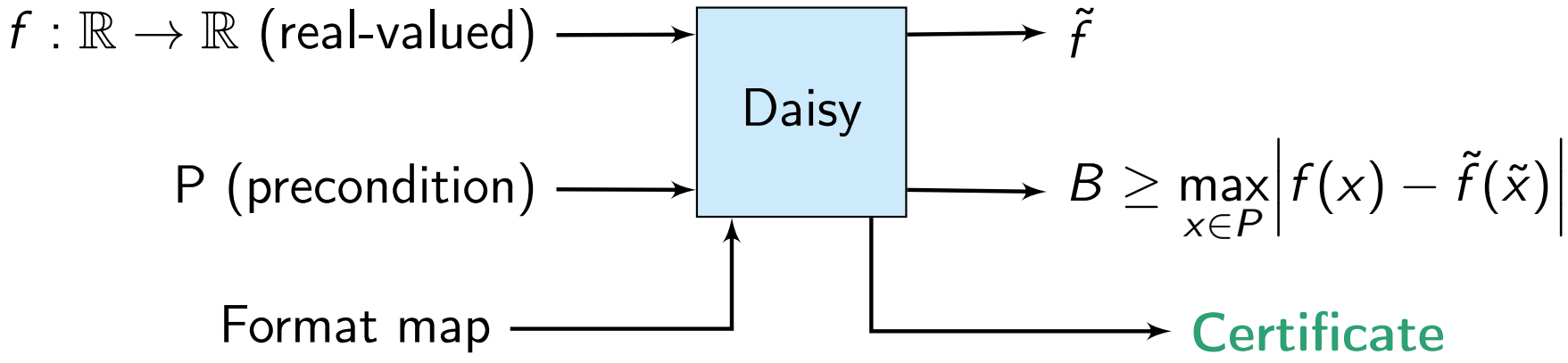
Overview of certificate checking



Overview of certificate checking



Overview of certificate checking

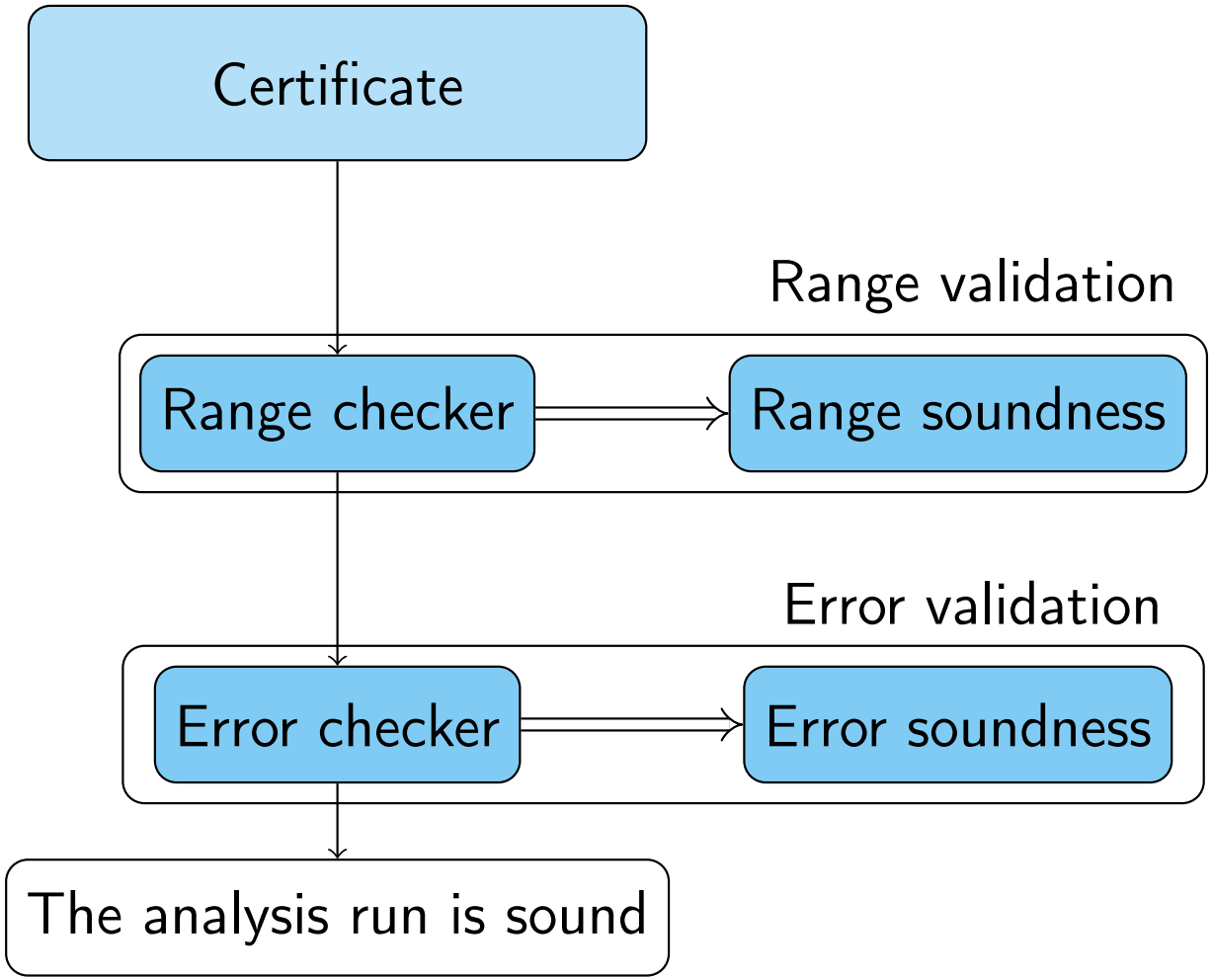


Certificate checker:

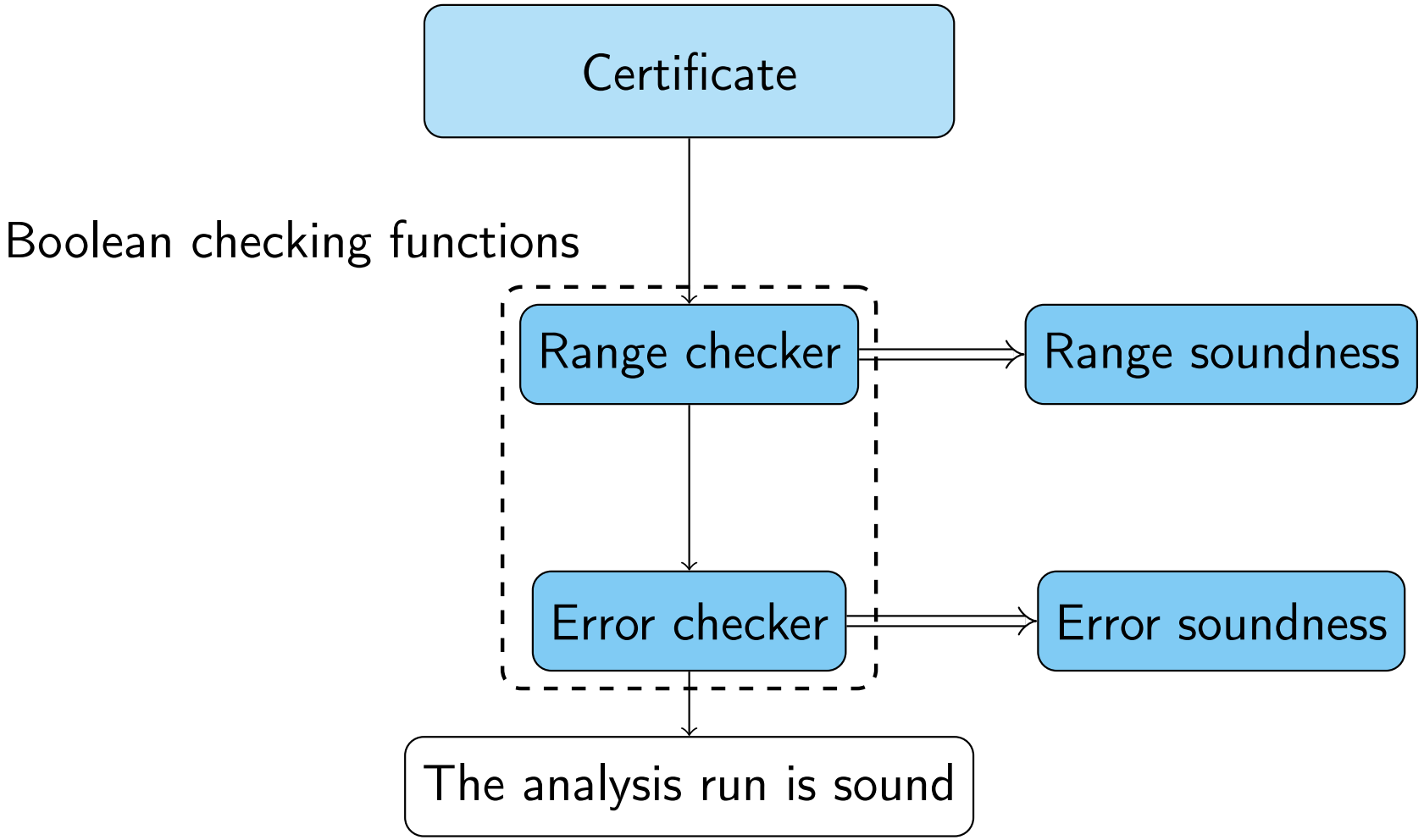
- computes its own roundoff error C
- checks that $B \geq C$

Formal soundness proof: $C \geq \max_{x \in P} \left| f(x) - \tilde{f}(\tilde{x}) \right|$

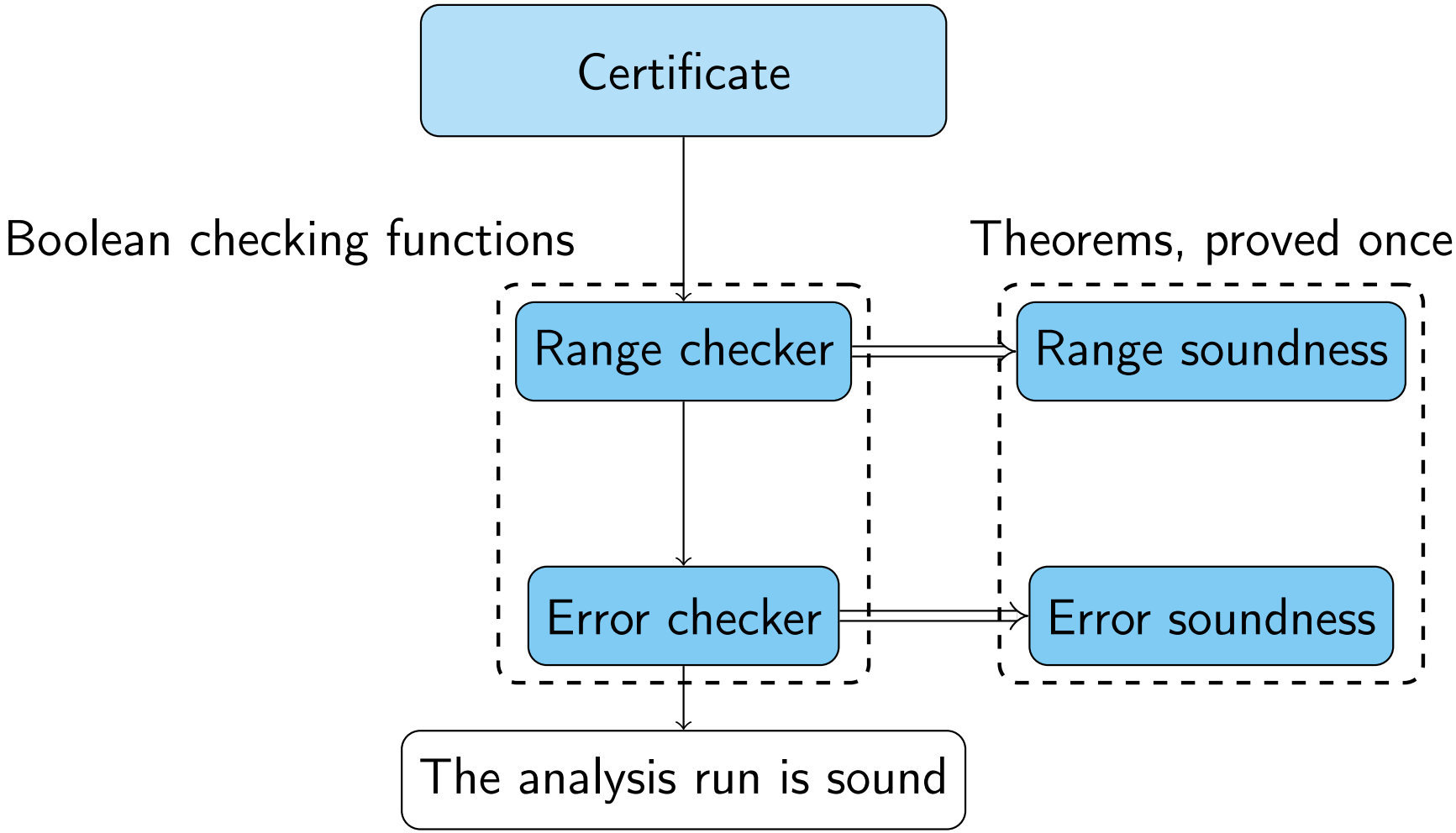
Overview of certificate checking – uniform precision



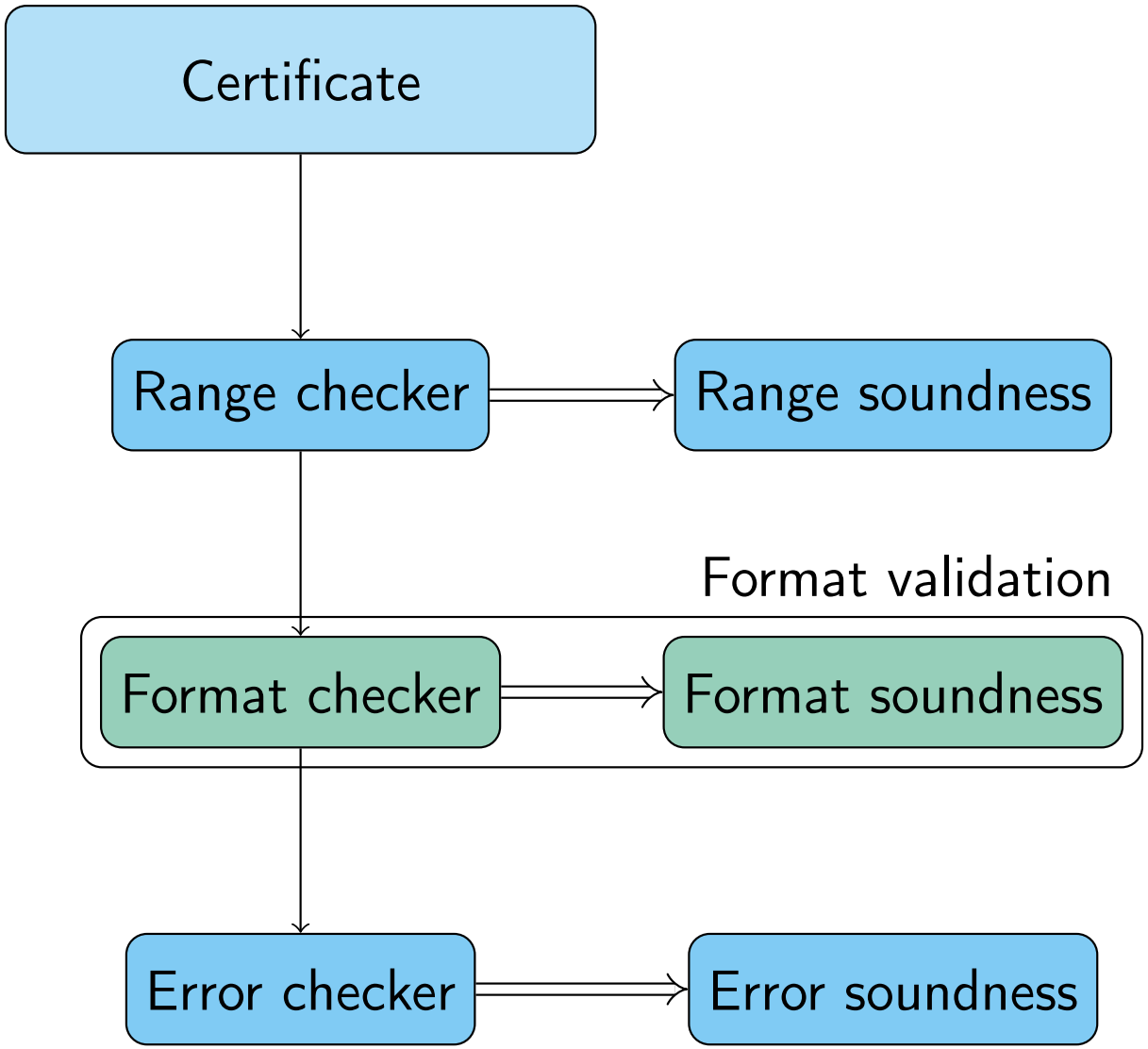
Overview of certificate checking – uniform precision



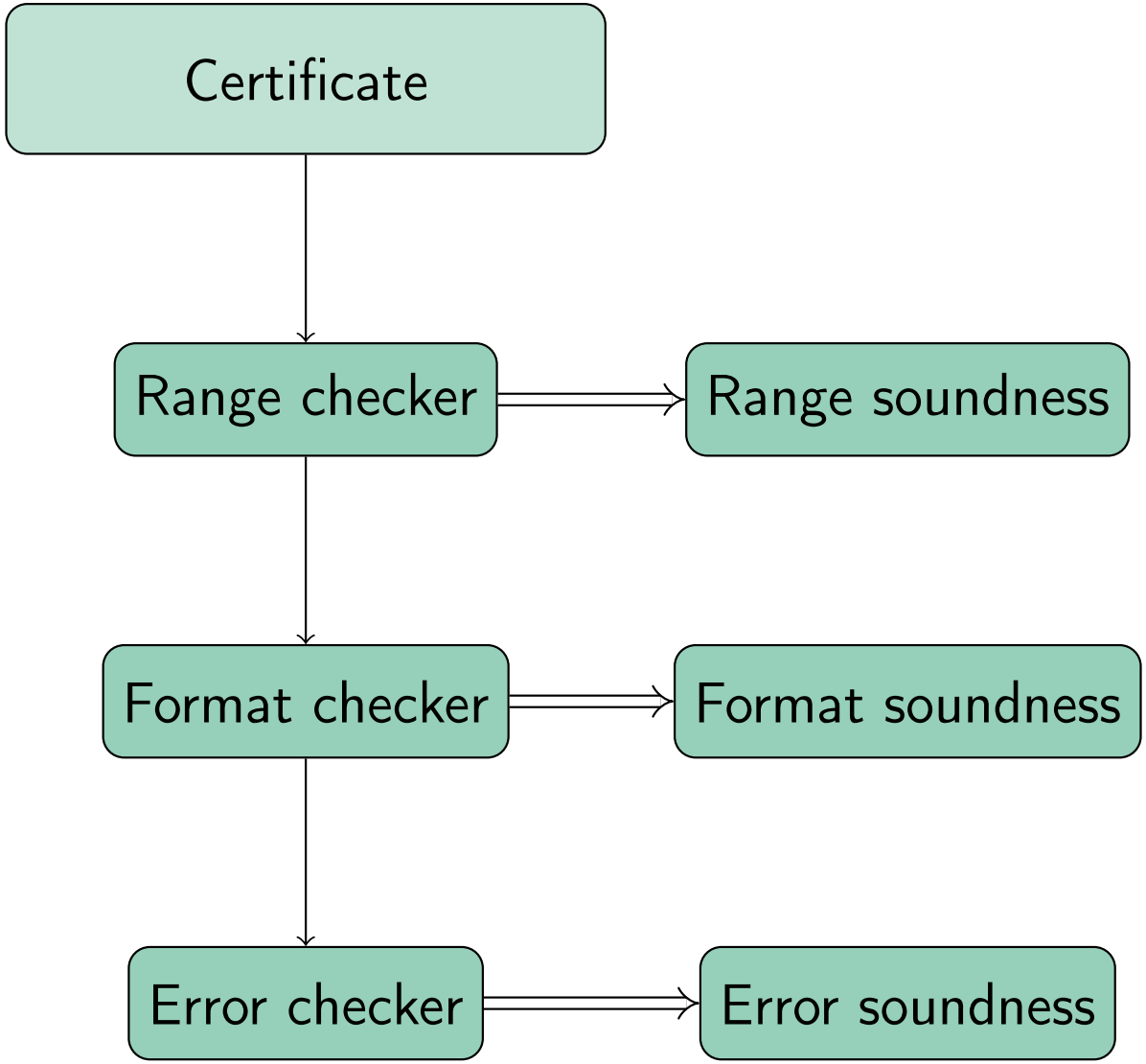
Overview of certificate checking – uniform precision



Adding support for mixed-precision



Adding support for mixed-precision



Certificate

What is encoded in a certificate of a function f?

- Definition of the expressions of f
- Precondition of f
- Analysis result (range and error) for each expression of f
- Format map for the variables of f

Certificate

```
Definition ExpVara0 :exp Q := Var Q 0.
Definition UMinExpVara0 :exp Q := Unop Neg ExpVara0.
Definition MultUMinExpVara0ExpVara0 :exp Q := Binop Mult UMinExpVara0 ExpVara0.

Definition defVars_sine :( nat → option mType) := fun n ⇒
  if n =? 0 then Some M64 else if n =? 3 then Some M128
  else if n =? 4 then Some M64 else None.

Definition thePrecondition_sine : precond := fun (n: nat) ⇒
if n =? 0 then ( (−1)/(1), (1)/(1)) else (0/1,0/1).

Definition absenv_sine :analysisResult :=
fun (e: exp Q) ⇒
if (expEqBool e ExpVara0) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e UMinExpVara0 ) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e MultUMinExpVara0ExpVara0 ) then
      (((−1)/(1), (1)/(1)),(24338891524382007210896477965516 9)/
           (730750818665451459101842416358141509827966271488))
```

Certificate

Expressions

```
Definition ExpVara0 :exp Q := Var Q 0.
Definition UMinExpVara0 :exp Q := Unop Neg ExpVara0.
Definition MultUMinExpVara0ExpVara0 :exp Q := Binop Mult UMinExpVara0 ExpVara0.

Definition defVars_sine :( nat → option mType) := fun n ⇒
  if n =? 0 then Some M64 else if n =? 3 then Some M128
  else if n =? 4 then Some M64 else None.

Definition thePrecondition_sine : precond := fun (n: nat) ⇒
if n =? 0 then ( (−1)/(1), (1)/(1)) else (0/1,0/1).

Definition absenv_sine :analysisResult :=
fun (e: exp Q) ⇒
if (expEqBool e ExpVara0) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e UMinExpVara0 ) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e MultUMinExpVara0ExpVara0 ) then
      (((−1)/(1), (1)/(1)),(24338891524382007210896477965516 9)/
           (73075081866545145910184241635814150982796627 1488))
```

Certificate

Expressions

```
Definition ExpVara0 : exp Q := Var Q 0.
Definition UMinExpVara0 : exp Q := Unop Neg ExpVara0.
Definition MultUMinExpVara0ExpVara0 : exp Q := Binop Mult UMinExpVara0 ExpVara0.
```

Format map

```
Definition defVars_sine :( nat → option mType) := fun n ⇒
  if n =? 0 then Some M64 else if n =? 3 then Some M128
  else if n =? 4 then Some M64 else None.
```

```
Definition thePrecondition_sine : precond := fun (n: nat) ⇒
if n =? 0 then ( (−1)/(1), (1)/(1)) else (0/1,0/1).
```

```
Definition absenv_sine : analysisResult :=
fun (e: exp Q) ⇒
if (expEqBool e ExpVara0) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e UMinExpVara0 ) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e MultUMinExpVara0ExpVara0 ) then
      (((−1)/(1), (1)/(1)),(24338891524382007210896477965516 9)/
            (730750818665451459101842416358141509827966271488))
```

Certificate

Expressions

```
Definition ExpVara0 : exp Q := Var Q 0.
Definition UMinExpVara0 : exp Q := Unop Neg ExpVara0.
Definition MultUMinExpVara0ExpVara0 : exp Q := Binop Mult UMinExpVara0 ExpVara0.
```

Format map

```
Definition defVars_sine :( nat → option mType) := fun n ⇒
  if n =? 0 then Some M64 else if n =? 3 then Some M128
  else if n =? 4 then Some M64 else None.
```

Precondition

```
Definition thePrecondition_sine : precond := fun (n: nat) ⇒
if n =? 0 then ( (−1)/(1), (1)/(1)) else (0/1,0/1).
```

```
Definition absenv_sine : analysisResult :=
fun (e: exp Q) ⇒
if (expEqBool e ExpVara0) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e UMinExpVara0) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e MultUMinExpVara0ExpVara0 ) then
      (((−1)/(1), (1)/(1)),(24338891524382007210896477965516 9)/
           (730750818665451459101842416358141509827966271488))
```

Certificate

Expressions

```
Definition ExpVara0 : exp Q := Var Q 0.
Definition UMinExpVara0 : exp Q := Unop Neg ExpVara0.
Definition MultUMinExpVara0ExpVara0 : exp Q := Binop Mult UMinExpVara0 ExpVara0.
```

Format map

```
Definition defVars_sine :( nat → option mType) := fun n ⇒
  if n =? 0 then Some M64 else if n =? 3 then Some M128
  else if n =? 4 then Some M64 else None.
```

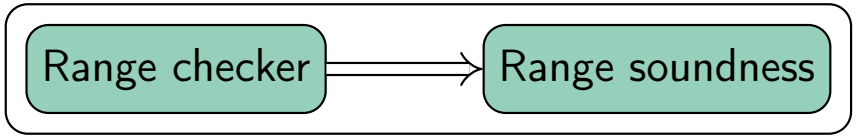
Precondition

```
Definition thePrecondition_sine : precond := fun (n: nat) ⇒
if n =? 0 then ( (−1)/(1), (1)/(1)) else (0/1,0/1).
```

Analysis result

```
Definition absenv_sine : analysisResult :=
fun (e: exp Q) ⇒
if (expEqBool e ExpVara0) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e UMinExpVara0) then (((−1)/(1), (1)/(1)), (1)/(9007199254740992))
else if (expEqBool e MultUMinExpVara0ExpVara0 ) then
      (((−1)/(1), (1)/(1)),(24338891524382007210896477965516 9)/
            (730750818665451459101842416358141509827966271488))
```

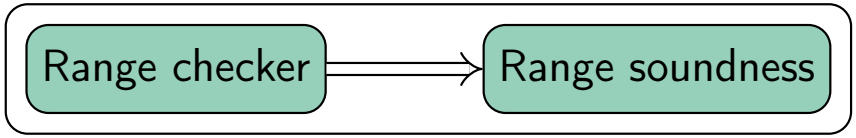
Range validation



Extension: case of `Downcast`

Check that the range of `Downcast m e` is equal to the range of `e`.

Range validation



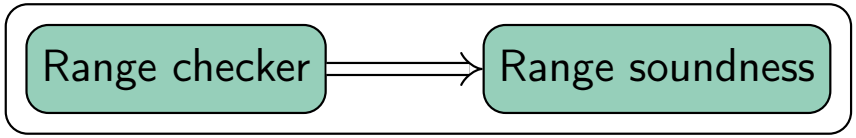
Extension: case of `Downcast`

Check that the range of `Downcast m e` is equal to the range of `e`.

Soundness theorem

```
Theorem validIntervalbounds_sound  e absenv P E:
  validIntervalbounds  e absenv P = true →
  eval_exp E (toREval e) vR M0 →
  vR ∈  (fst (absenv e))
```

Range validation



Extension: case of `Downcast`

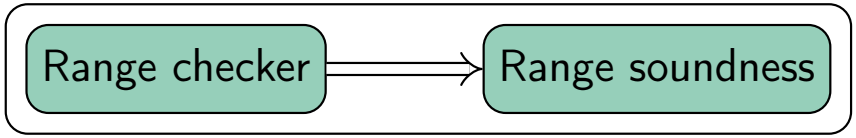
Check that the range of `Downcast m e` is equal to the range of `e`.

Soundness theorem

```
Theorem validIntervalbounds_sound  e absenv P E:
  validIntervalbounds  e absenv P = true →
  eval_exp E (toREval e) vR M0 →
  vR ∈  (fst (absenv e))
```

Range check ok

Range validation



Extension: case of `Downcast`

Check that the range of `Downcast m e` is equal to the range of `e`.

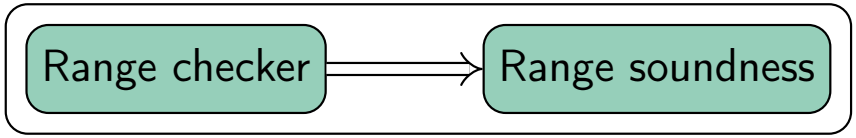
Soundness theorem

```
Theorem validIntervalbounds_sound  e absenv P E:
  validIntervalbounds  e absenv P = true →
  eval_exp E (toREval e) vR M0 →
  vR ∈  (fst (absenv e))
```

Range check ok

The real value of e is vR

Range validation



Extension: case of `Downcast`

Check that the range of `Downcast m e` is equal to the range of `e`.

Soundness theorem

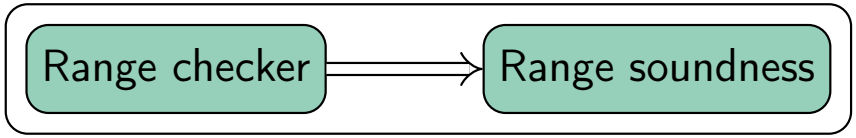
```
Theorem validIntervalbounds_sound  e absenv P E:
  validIntervalbounds  e absenv P = true →
  eval_exp E (toREval e) vR M0 →
  vR ∈ (fst (absenv e))
```

Range check ok

The real value of e is vR

Range computed by Daisy

Range validation



Extension: case of `Downcast`

Check that the range of `Downcast m e` is equal to the range of `e`.

Soundness theorem

```
Theorem validIntervalbounds_sound  e absenv P E:
  validIntervalbounds  e absenv P = true →
  eval_exp E (toREval e) vR M0 →
  vR ∈ (fst (absenv e))
```

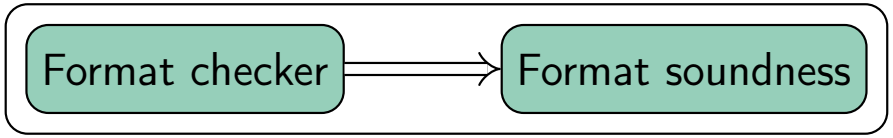
Range check ok

The real value of e is vR

Range computed by Daisy

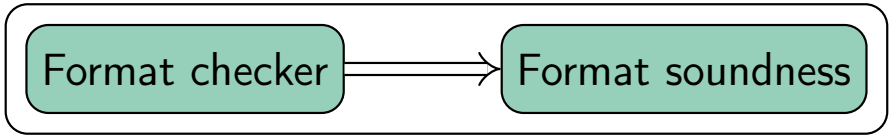
$\Longrightarrow$ The real range analysis is sound for e.

Format validation



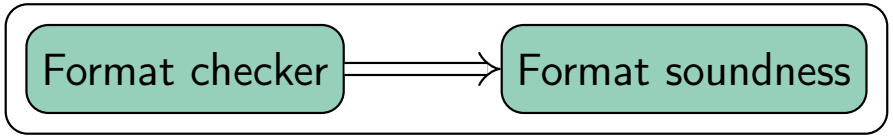
- Completely new validation phase

Format validation



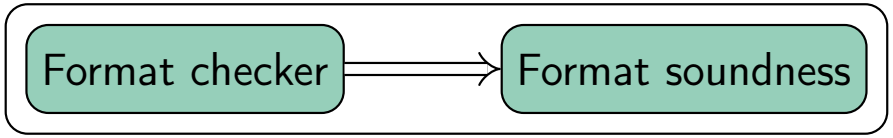
- Completely new validation phase
- Checks that the precision in the expressions are coherent:

Format validation



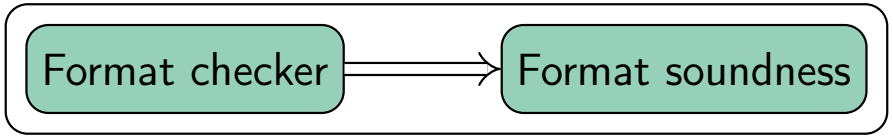
- Completely new validation phase
- Checks that the precision in the expressions are coherent:
  - `(x:Double) + (y:Float)` should be `Double`

Format validation



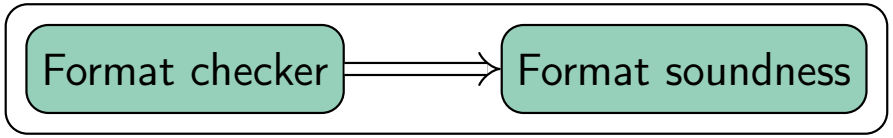
- Completely new validation phase
- Checks that the precision in the expressions are coherent:
  - `(x:Double) + (y:Float)` should be `Double`
  - $\mathit{Downcast}(64, x_{32})$ is incorrect

Format validation



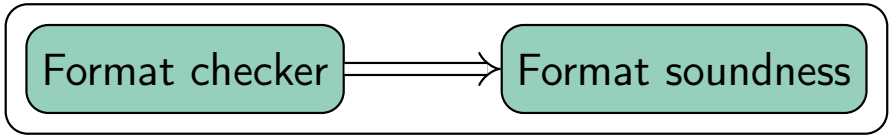
- Completely new validation phase
- Checks that the precision in the expressions are coherent:
  - `(x:Double) + (y:Float)` should be `Double`
  - $\mathit{Downcast}(64, x_{32})$ is incorrect
- Difficult to design: I wanted to have a stronger property

Format validation




```
Theorem formatSoundnessExp  e defVars E v m Gamma:
  formatCheck e defVars Gamma = true →
  eval_exp E defVars e v m →
  Gamma e = Some m.
```

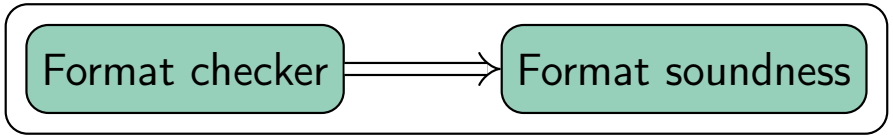
Format validation




```
Theorem formatSoundnessExp  e defVars E v m Gamma:
  formatCheck e defVars Gamma = true →
  eval_exp E defVars e v m →
  Gamma e = Some m.
```

Format check of Γ ok

Format validation

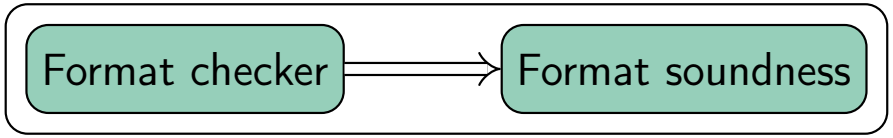



```
Theorem formatSoundnessExp  e defVars E v m Gamma:
  formatCheck e defVars Gamma = true →
  eval_exp E defVars e v m →
  Gamma e = Some m.
```

Format check of Γ ok

In Scala, e has type m

Format validation



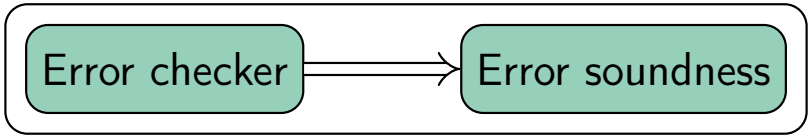

```
Theorem formatSoundnessExp  e defVars E v m Gamma:
  formatCheck e defVars Gamma = true →
  eval_exp E defVars e v m →
  Gamma e = Some m.
```

Format check of Γ ok

In Scala, e has type m

$\Longrightarrow$ The types used in the analysis are correct with respect to the Scala semantics.

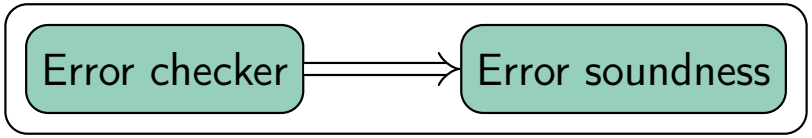
Error validation



Case of `Downcast`

Example: `val y: Float = (x: Double). toFloat`

Error validation

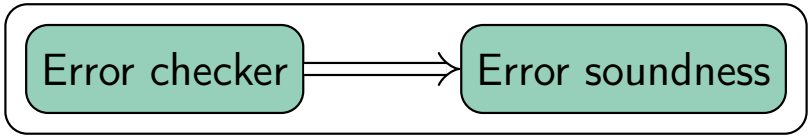


Case of `Downcast`

Example: `val y: Float = (x: Double). toFloat` $\rightsquigarrow$ `Downcast 32` $\tilde{x}_{64}$.

$$|x - (\texttt{Downcast 32 } \tilde{x}_{64})| \leq |x - \tilde{x}_{64}| +$$

Error validation

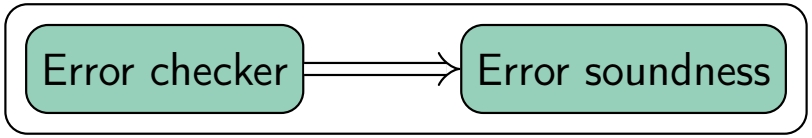


Case of `Downcast`

Example: `val y: Float = (x: Double). toFloat` $\rightsquigarrow$ `Downcast 32` $\tilde{x}_{64}$.

$$|x - (\texttt{Downcast 32 } \tilde{x}_{64})| \leq |x - \tilde{x}_{64}| + |\tilde{x}_{64} - (\texttt{Downcast 32 } \tilde{x}_{64})|$$

Error validation

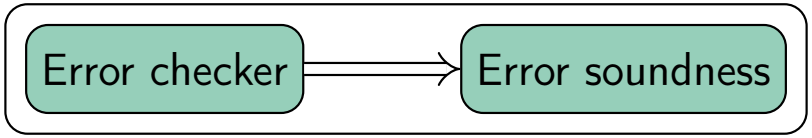


Case of `Downcast`

Example: `val y: Float = (x: Double). toFloat` $\rightsquigarrow$ `Downcast 32` $\tilde{x}_{64}$.

$$\begin{aligned} |x - (\texttt{Downcast 32 } \tilde{x}_{64})| &\leq |x - \tilde{x}_{64}| + |\tilde{x}_{64} - (\texttt{Downcast 32 } \tilde{x}_{64})| \\ &\leq |x - \tilde{x}_{64}| + |\tilde{x}_{64}| \cdot 2^{-24} \end{aligned}$$

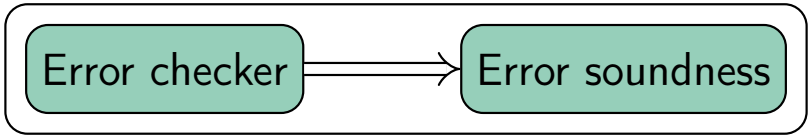
Error validation




```
Theorem validErrorbound_sound  e absenv P E1 E2 m Gamma defVars:
  approxEnv E1 defVars absenv E2 →
  validIntervalbounds  e absenv P = true →
  formatCheck e defVars Gamma = true →
  validErrorbound  e absenv Gamma = true →
  eval_exp E1 (toREval e) vR M0 →
  eval_exp E2 e vF m →
  |vR − vF| <= snd (absenv e).
```

Range check ok

Error validation

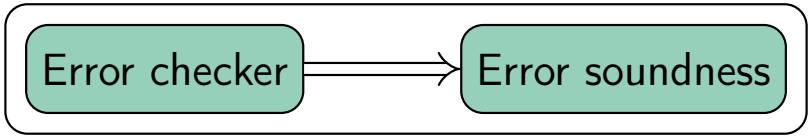



```
Theorem validErrorbound_sound  e absenv P E1 E2 m Gamma defVars:
  approxEnv E1 defVars absenv E2 →
  validIntervalbounds  e absenv P = true →
  formatCheck e defVars Gamma = true →
  validErrorbound  e absenv Gamma = true →
  eval_exp E1 (toREval e) vR M0 →
  eval_exp E2 e vF m →
  |vR − vF| <= snd (absenv e).
```

Range check ok

Format check ok

Error validation



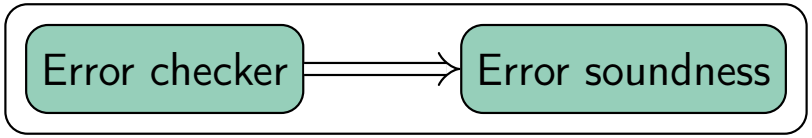

```
Theorem validErrorbound_sound  e absenv P E1 E2 m Gamma defVars:
  approxEnv E1 defVars absenv E2 →
  validIntervalbounds  e absenv P = true →
  formatCheck e defVars Gamma = true →
  validErrorbound  e absenv Gamma = true →
  eval_exp E1 (toREval e) vR M0 →
  eval_exp E2 e vF m →
  |vR − vF| <= snd (absenv e).
```

Range check ok

Format check ok

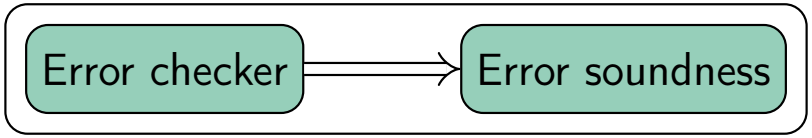
Error check ok

Error validation



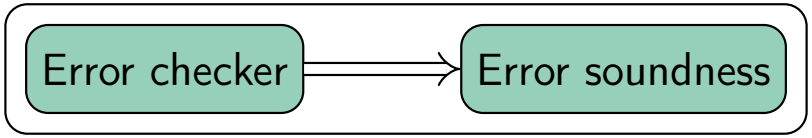

```
Theorem validErrorbound_sound  e absenv P E1 E2 m Gamma defVars:
  approxEnv E1 defVars absenv E2 →
  validIntervalbounds  e absenv P = true →          Range check ok
  formatCheck e defVars Gamma = true →              Format check ok
  validErrorbound  e absenv Gamma = true →          Error check ok
  eval_exp E1 (toREval e) vR M0 →                   Using reals, e evaluates to vR
  eval_exp E2 e vF m →
  |vR − vF| <= snd (absenv e).
```

Error validation




```
Theorem validErrorbound_sound  e absenv P E1 E2 m Gamma defVars:
  approxEnv E1 defVars absenv E2 →
  validIntervalbounds  e absenv P = true →          Range check ok
  formatCheck e defVars Gamma = true →              Format check ok
  validErrorbound  e absenv Gamma = true →          Error check ok
  eval_exp E1 (toREval e) vR M0 →                   Using reals, e evaluates to vR
  eval_exp E2 e vF m →                              In Scala, e evaluates to vF having type m
  |vR - vF| <= snd (absenv e).
```

Error validation




```
Theorem validErrorbound_sound  e absenv P E1 E2 m Gamma defVars:
  approxEnv E1 defVars absenv E2 →
  validIntervalbounds  e absenv P = true →
  formatCheck e defVars Gamma = true →
  validErrorbound  e absenv Gamma = true →
  eval_exp E1 (toREval e) vR M0 →
  eval_exp E2 e vF m →
  |vR − vF| <= snd (absenv e).
```

Range check ok

Format check ok

Error check ok

Using reals, e evaluates to vR

In Scala, e evaluates to vF having type m

$\Longrightarrow$ The result of the analysis is an upper-bound on the roundoff error, so the run roundoff error analysis is sound.

Related Work

Related Work

Tool	Fluctuat[1]	Daisy	FPTaylor[2]	Gappa[3]
Formal verification	✗	✓	✓	✓
Mixed-precision	✓	✓	✗	✗
Control-flow	✓	✗	✗	✗
Variable-bindings	✓	✓	✗	✓
Automation	Full	Full	Full	Partial
Provers	N/A	Coq, HOL4	HOL-Light	Coq
Domain	Custom	IA	N/A	IA

[1] Goubault and Putot. "Static Analysis of Numerical Algorithms". *SAS 2006*.

[2] Solovyev et al. "Rigorous Estimation of Floating-Point Round-off Errors with Symbolic Taylor Expansions". *FM 2015*.

[3] Boldo, Filliâtre, and Melquiond. "Combining Coq and Gappa for Certifying Floating-Point Programs". *ICM, CICM 2009*.

Conclusion

Conclusion

I extended the formal validation of Daisy to mixed-precision.

Daisy is the first formally verified tool supporting mixed-precision.

Conclusion

I extended the formal validation of Daisy to mixed-precision.
Daisy is the first formally verified tool supporting mixed-precision.

Also:

- Formally implemented a part of Affine Arithmetic
- Helped create an automata demo for the "Girl's Day"
- Designed and conducted a practical session on Interproc, for a Master's course on Static Analysis
- Regular meetings within the PLV group

Code statistics

- Daisy: 11,000 lines of Scala code
- Uniform-precision development: 5,000 (Coq), 6,000 (HOL4)
- Mixed-precision development: 6,500 (Coq), 7,100 (HOL4)

Girl's Day

Bahnhof Scheidt

Halberg

Flughafen Saarbrücken

Hauptbahnhof
Saarbrücken

Völklingen

Burbach Mitte

Universität des Saarlandes

Schloss Saarbrücken

Staatstheater