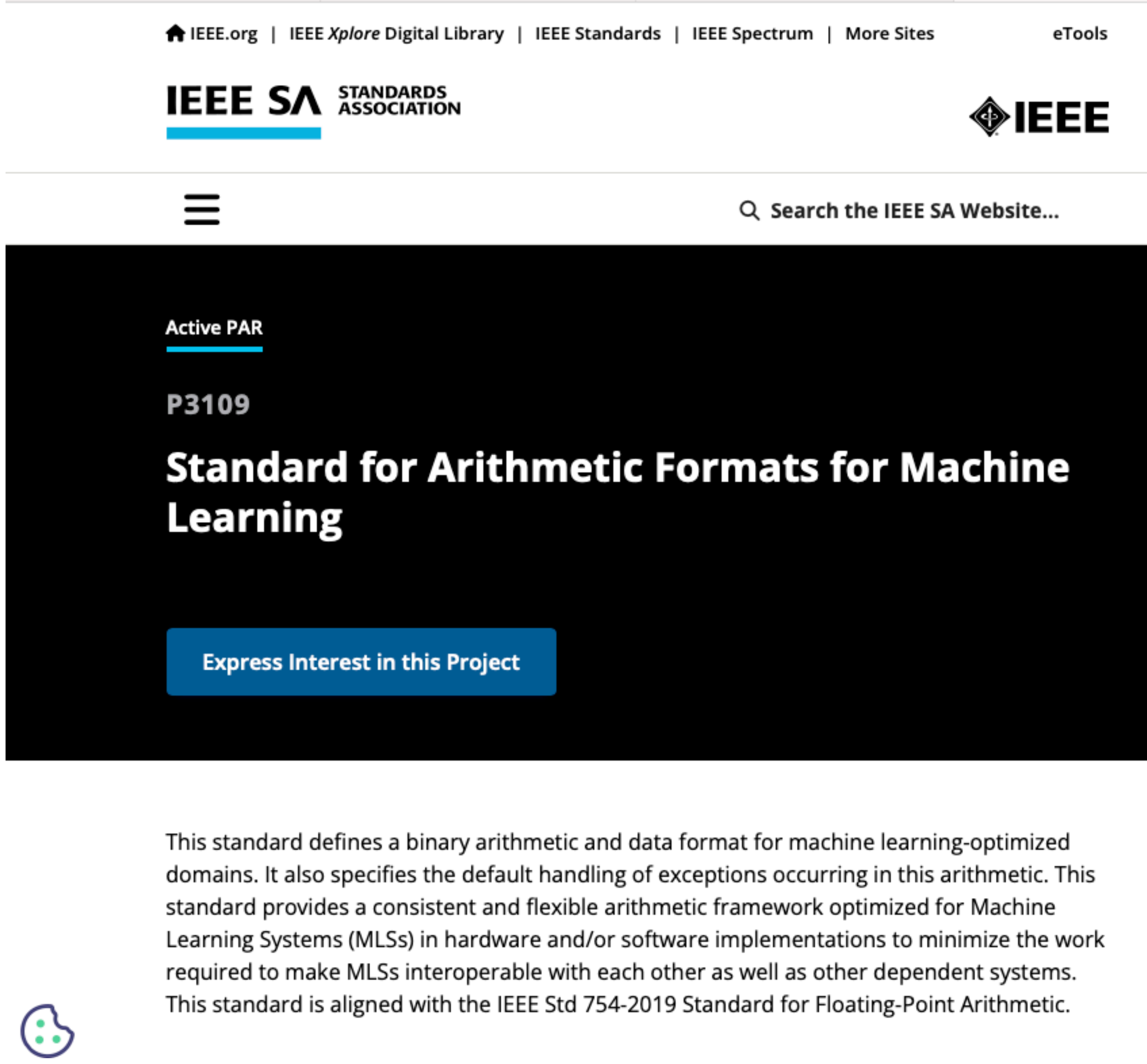
Formal Verification of the IEEE P3109 Standard for Binary Floating-point Formats for Machine Learning

Christoph M. Wintersteiger
christoph@imandra.ai

What is IEEE P3109?

- A new standard for floating-point numbers
  - for machine-learning applications
  - consistent and flexible framework
  - small bit-width (2-15 bits)
  - arithmetic & interchange format
  - defines profiles



https://standards.ieee.org/ieee/3109/11165/

https://github.com/P3109/Public

Anatomy of binary8p3se

k=8

p=3

01000101 $= 2^1 \times 1.25 = 2.5$

± Biased Exponent Significand (- hidden bit)

(with NaN and optional ± and ∞)

Operation Specification Strategy

- From floating-point (bit pattern) $\mathbb{F}$
- Decode into extended reals (or NaN) $\mathbb{F} \to \overline{\mathbb{R}}$
- Apply operation on extended reals (full precision) $\ldots \times \overline{\mathbb{R}} \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$
- Round and saturate $\ldots \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$
- Encode result $\overline{\mathbb{R}} \to \mathbb{F}$

Specification example

4.6.1 Decode

Decode P3109 value to extended real or NaN.

Signature

$\mathsf{Decode}_f(x) \to X$

Parameters

f : format, width K_f, precision P_f, exponent bias b_f

Operands

x : P3109 value, in format f

Output

X : extended real value or NaN

Behavior

$$\begin{aligned}
&\mathsf{Decode}(2^{\mathsf{K}_f-1}) \to \mathsf{NaN}\\
&\mathsf{Decode}(2^{\mathsf{K}_f-1}-1) \to \infty\\
&\mathsf{Decode}(2^{\mathsf{K}_f-1} < x < 2^{\mathsf{K}_f}) \to -\mathsf{Decode}_f(x - 2^{\mathsf{K}_f-1})\\
&\mathsf{Decode}(0 \le x < 2^{\mathsf{K}_f-1}-1) \to X
\end{aligned}$$

where

$$X = \begin{cases} (0 + T \times 2^{1-\mathsf{P}_f}) \times 2^{E+1} & \text{if } E = -b_f \\ (1 + T \times 2^{1-\mathsf{P}_f}) \times 2^{E} & \text{Otherwise} \end{cases}$$

$$T = x \bmod 2^{\mathsf{P}_f-1}$$

$$E = x \div 2^{\mathsf{P}_f-1} - b_f$$

4.8.3 Binary arithmetic operations

These operations take two P3109 values as input, and return a P3109 value. The definitions allow for all input and output formats to differ, a given implementation may supply any subset of the defined operations.

Signature

$\mathsf{Operation}_{f_x,f_y,f_z,\pi}(x,y) \to z$

Parameters

f_x : format of x
f_y : format of y
f_z : format of z
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Output

z : P3109 value, format f_z

Behavior

$$\begin{aligned}
&\mathsf{Add}(*, \mathsf{NaN}) \to \mathsf{NaN}\\
&\mathsf{Add}(\mathsf{NaN}, *) \to \mathsf{NaN}\\
&\mathsf{Add}(\text{-}\mathsf{Inf}, \mathsf{Inf}) \to \mathsf{NaN}\\
&\mathsf{Add}(\mathsf{Inf}, \text{-}\mathsf{Inf}) \to \mathsf{NaN}\\
&\mathsf{Add}(x,y) \to \mathsf{Project}_{f_z,\pi}(X+Y) \quad \textbf{where} \quad X = \mathsf{Decode}_{f_x}(x), Y = \mathsf{Decode}_{f_y}(y)
\end{aligned}$$

IMANDRA

Extended Reals

- Real numbers with infinities $\mathbb{R} \cup \pm\infty$
- Undefined cases
  - $+\infty \; + -\infty = ?$
  - $\pm\infty \; \times \; 0 = ?$
- Some choices
- Cannot depend on undefined operations

```
let add (x : t) (y : t) : (t, string) Result.t =
  match x, y with
  | NINF, PINF | PINF, NINF -> Error "undefined"
  | NINF, NINF | R _, NINF | NINF, R _ -> Ok NINF
  | PINF, PINF | R _, PINF | PINF, R _ -> Ok PINF
  | R x, R y -> Ok (R (x +. y))
```

ImandraX

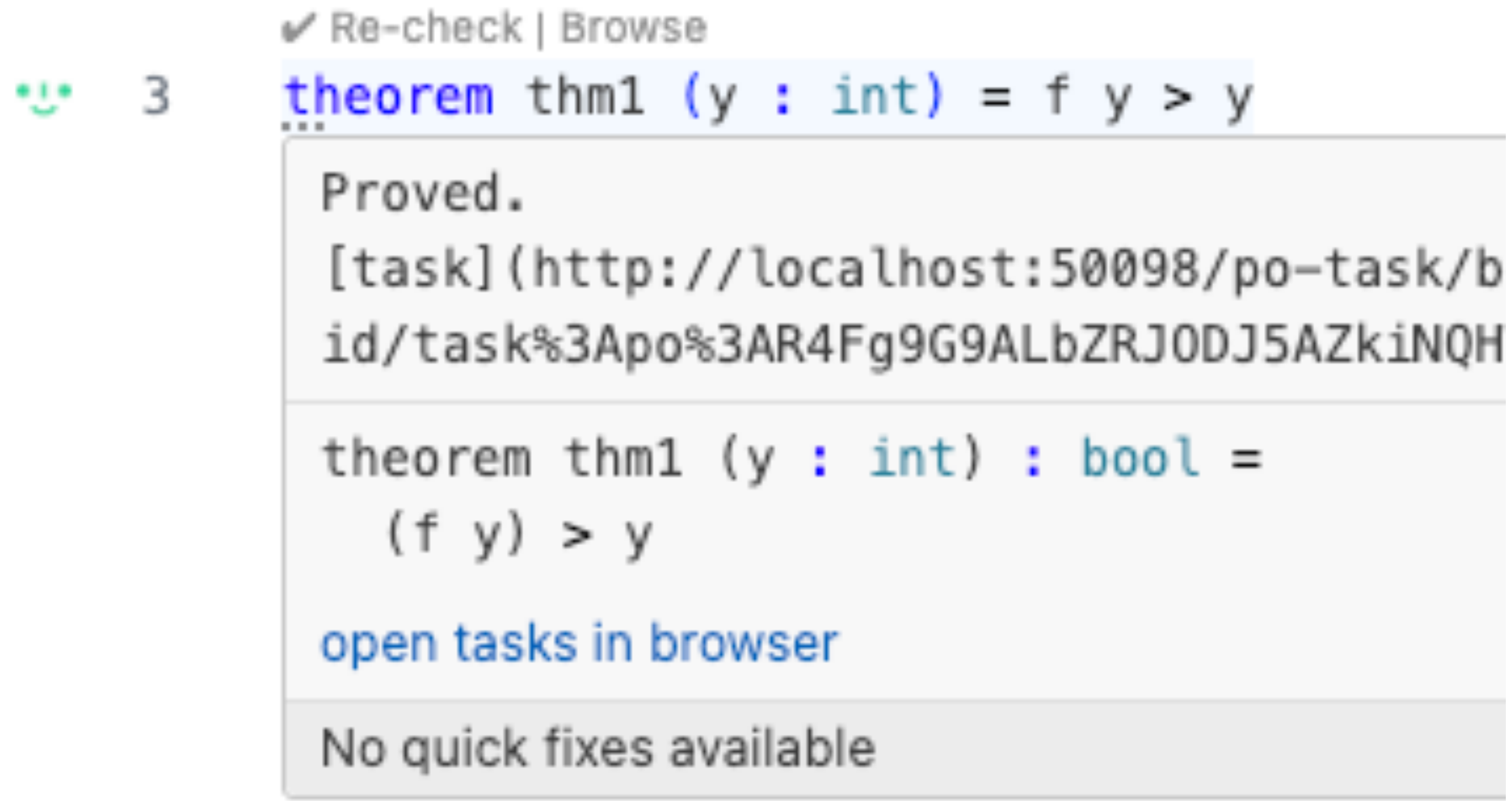
- Theorem prover & program verifier
- IML (based on OCaml)
- Functional, strongly typed
- Total functions
- No
  - Exceptions
  - Assertions
  - Automatic conversions
  - Operator overloading
  - Quantifiers

```
(f x y) = f(x, y)
```

ImandraX Hello World

```
let f (x : int) : int = x + 1

theorem thm1 (y : int) = f y > y
```



IMANDRA

Formal Specification Example

```
let add
  (f_x : Format.t) (f_y : Format.t)
  (f_z : Format.t) (pi : Projection.t)
  (x : Float.t) (y : Float.t) : (t, string) Result.t =
    let open NaNOrExReal in
    match (x, y) with
    | _, y when y = nan f_y -> Ok (nan f_z)
    | x, _ when x = nan f_x -> Ok (nan f_z)
    | x, y when x = ninf f_x && y = pinf f_y -> Ok (nan f_z)
    | x, y when x = pinf f_x && y = ninf f_y -> Ok (nan f_z)
    | x, y ->
      let x = decode f_x x in
      let y = decode f_y y in
      (match x, y with
      | Ok (XR x), Ok (XR y)  ->
        let open ExReal.Infix in
        (match (x +. y) with
        | Ok z -> project f_z pi z
        | Error e -> Error e)
      | _, Error e
      | Error e, _ -> Error e)
```

4.8.3 Binary arithmetic operations

These operations take two P3109 values as input, and return a P3109 value. The definitions all output formats to differ, a given implementation may supply any subset of the defined operations

Signature

$\mathsf{Operation}_{f_x,f_y,f_z,\pi}(x, y) \to z$

Parameters

f_x : format of x
f_y : format of y
f_z : format of z
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Output

z : P3109 value, format f_z

Behavior

$\mathsf{Add}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\mathsf{Add}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{Add}(\text{-}\mathsf{Inf}, \mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{Add}(\mathsf{Inf}, \text{-}\mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{Add}(x, y) \to \mathsf{Project}_{f_z,\pi}(X + Y) \quad \textbf{where} \quad X = \mathsf{Decode}_{f_x}(x), Y = \mathsf{Decode}_{f_y}(y)$

IMANDRA

Formal Specification Example

```
let add
  (f_x : Format.t) (f_y : Format.t)
  (f_z : Format.t) (pi : Projection.t)
  (x : Float.t) (y : Float.t) : (t, string) Result.t =
    let open NaNOrExReal in
    match (x, y) with
    | _, y when y = nan f_y -> Ok (nan f_z)
    | x, _ when x = nan f_x -> Ok (nan f_z)
    | x, y when x = ninf f_x && y = pinf f_y -> Ok (nan f_z)
    | x, y when x = pinf f_x && y = ninf f_y -> Ok (nan f_z)
    | x, y ->
      let x = decode f_x x in
      let y = decode f_y y in
      (match x, y with
      | Ok (XR x), Ok (XR y)  ->
        let open ExReal.Infix in
        (match (x +. y) with
        | Ok z -> project f_z pi z
        | Error e -> Error e)
      | _, Error e
      | Error e, _ -> Error e)
```

```
theorem add_ok
  (f_x : Format.t) (f_y : Format.t)
  (f_z : Format.t) (pi : Projection.t)
  (x : Float.t) (y : Float.t) =
 Result.is_ok (Float.add f_x f_y f_z pi x y)
  ...
  [@@by auto]
```

IMANDRA

Format ranges

- **decode** must always produce numbers within the range of the format
- Speeds up many other proofs
- Combination of
  - Ground evaluation
  - Induction

```
let is_within (kp : Format.kpt) (x : Float.t) : bool =
  let open Float.NaNOrExReal in
  let f = { kp = kp; s = Signed; d = Extended } in
  let _, _, _, fmax, _, _ = Format.get_format_parameters f in
  match Float.decode f x with
  | Ok NaN
  | Ok (XR NINF)
  | Ok (XR PINF) -> true
  | Ok (XR r) -> R Real.(- fmax) <=. r && r <=. R fmax
  | _ -> false

let rec forall_in_range (p : int -> bool) (l : int) (u : int) =
  if l > u then
    true
  else
    p l && forall_in_range p (l + 1) u
  [@@measure Ordinal.of_int (u - l)]

theorem forall_elim (p: int -> bool) (l : int) (u : int) (x : int) =
  forall_in_range p l u ==> (l <= x && x <= u ==> p x)

theorem wfr_above (kp : Format.kpt) (x : Float.t) =
  let f = { kp = kp; s = Signed; d = Extended } in
  let k, _, _, _, _, _ = Format.get_format_parameters f in
  (x > ipow2 k) ==> is_within kp x [@trigger]

theorem wfr_15p1_mid = forall_in_range (is_within B15P1) 0 (ipow2 15)
 [@@by ground_eval] [@@timeout 60]

... [%use forall_elim (is_within B15P1) 0 (ipow2 15) x]
```

Some Numbers

- 3k lines of code
- ~23 minutes

Property	#
Extended reals safety	28
Intermediate lemmas	34
Format ranges	121
Termination	83
Test cases	91
Total proof obligations	357

Future work

- Ongoing
  - Additional formats
  - Block formats
  - Precision of elementary functions
  - Verified example implementation

Conclusion

- P3109 is a new floating-point standard
- Formal verification employed as a design tool
- Ensures formal properties of the specification
- Validates design choices
- Reference implementation
- Test & verification reference for implementers
- https://github.com/imandra-ai/ieee-P3109