



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

Christoph M. Wintersteiger  
[christoph@imandra.ai](mailto: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

The screenshot shows the IEEE Standards Association website. At the top right is the IEEE logo. Below it is a search bar with the placeholder "Search the IEEE SA Website...". To the left of the search bar is the IEEE SA Standards Association logo. On the far left is a vertical navigation menu with three horizontal bars. In the center, there is a dark card with white text. At the top of the card is the text "Active PAR" underlined in blue. Below it is "P3109". The main title of the card is "Standard for Arithmetic Formats for Machine Learning". At the bottom of the card is a blue button with the text "Express Interest in this Project".

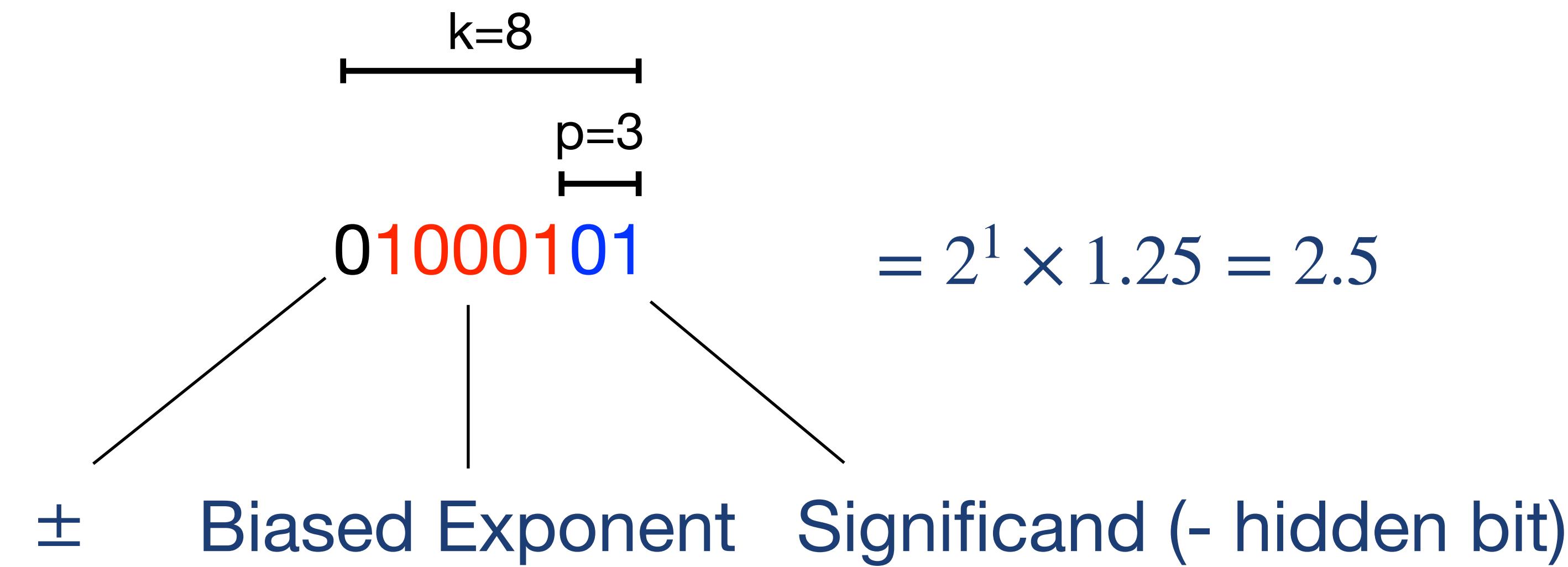
This standard defines a binary arithmetic and data format for machine learning-optimized domains. It also specifies the default handling of exceptions occurring in this arithmetic. This standard provides a consistent and flexible arithmetic framework optimized for Machine Learning Systems (MLSs) in hardware and/or software implementations to minimize the work required to make MLSs interoperable with each other as well as other dependent systems. This standard is aligned with the IEEE Std 754-2019 Standard for Floating-Point Arithmetic.



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

<https://github.com/P3109/Public>

# Anatomy of binary8p3se



(with NaN and optional  $\pm$  and  $\infty$ )

# Operation Specification Strategy

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

# Specification example

## 4.6.1 Decode

Decode P3109 value to extended real or NaN.

### Signature

$$\text{Decode}_f(x) \rightarrow X$$

### 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

$$\text{Decode}(2^{K_f-1}) \rightarrow \text{NaN}$$

$$\text{Decode}(2^{K_f-1} - 1) \rightarrow \infty$$

$$\text{Decode}(2^{K_f-1} < x < 2^{K_f}) \rightarrow -\text{Decode}_f(x - 2^{K_f-1})$$

$$\text{Decode}(0 \leq x < 2^{K_f-1} - 1) \rightarrow X$$

### where

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

$$T = x \bmod 2^{P_f-1}$$

$$E = x \div 2^{P_f-1} - b_f$$

## 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

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

### Parameters

$f_x$  : format of  $x$

$f_y$  : format of  $y$

$f_z$  : format of  $z$

$\pi$  : projection specification

### Operands

$x$  : P3109 value, format  $f_x$

$y$  : P3109 value, format  $f_y$

### Output

$z$  : P3109 value, format  $f_z$

### Behavior

$$\text{Add}(*, \text{NaN}) \rightarrow \text{NaN}$$

$$\text{Add}(\text{NaN}, *) \rightarrow \text{NaN}$$

$$\text{Add}(-\text{Inf}, \text{Inf}) \rightarrow \text{NaN}$$

$$\text{Add}(\text{Inf}, -\text{Inf}) \rightarrow \text{NaN}$$

$$\text{Add}(x, y) \rightarrow \text{Project}_{f_z, \pi}(X + Y) \quad \text{where } X = \text{Decode}_{f_x}(x), Y = \text{Decode}_{f_y}(y)$$

# Extended Reals

- Real numbers with infinities  $\mathbb{R} \cup \pm\infty$

- Undefined cases

- $+\infty + -\infty = ?$
- $\pm\infty \times 0 = ?$

```
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))
```

- Some choices
- Cannot depend on undefined operations

# 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
```

•• 3

✓ Re-check | Browse

```
theorem thm1 (y : int) = f y > y
...
Proved.
[task] (http://localhost:50098/po-task/b
id/task%3Apo%3AR4Fg9G9ALbZRJ0DJ5AZkiNQH
```

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

[open tasks in browser](#)

No quick fixes available

# 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 allow output formats to differ, a given implementation may supply any subset of the defined operations.

### Signature

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

### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_z$  : format of  $z$   
 $\pi$  : projection specification

### Operands

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

### Output

$z$  : P3109 value, format  $f_z$

### Behavior

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

# 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]

# 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>