




université
PARIS-SACLAY



Laboratoire
Méthodes
Formelles

A FLOATING-POINT NUMBERS THEORY FOR EVENT-B

 The LMF Lab Seminar

 Domaine Saint Paul, Saint-Remy-Lès-Chevreuse - June 13-14, 2024



Idir AIT SADOUNE 
idiraitsadoune@lmf.cnrs.fr 

OUTLINE

- The context of the work
- The motivating example
- The proposed approach
- Revisiting the motivating example
- Conclusion and future works

[Back to the begin](#) - [Back to the outline](#)

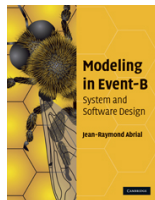
OUTLINE

- The context of the work
- The motivating example
- The proposed approach
- Revisiting the motivating example
- Conclusion and future works

[Back to the begin](#) - [Back to the outline](#)

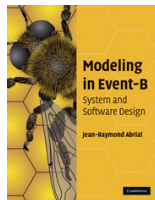
THE EVENT-B METHOD

- The **Event-B method** is an evolution of the **classical B method**.
 - modelling a system by a **set of events** instead of **operations**.



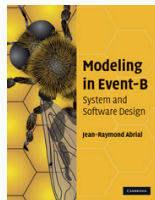
THE EVENT-B METHOD

- The **Event-B method** is an evolution of the **classical B method**.
 - modelling a system by a **set of events** instead of **operations**.
- The **Event-B method** is a formal method based on **first-order logic** and **set theory**.



THE EVENT-B METHOD

- The **Event-B method** is an evolution of the **classical B method**.
 - modelling a system by a **set of events** instead of **operations**.
- The **Event-B method** is a formal method based on **first-order logic** and **set theory**.
- The **Event-B method** is based on :
 - the notions of **pre-conditions** and **post-conditions** (**Hoare**),
 - the **weakest pre-condition** (**Dijkstra**),
 - and the **calculus of substitution** (**Abrial**).



USING EVENT-B METHOD

- The use of the **Event-B method** has continued to increase.
 - applied to various applications and domains.
 - railway, automotive, aeronautics, cybersecurity, nuclear-energy, ...

USING EVENT-B METHOD

- The use of the **Event-B method** has continued to increase.
 - applied to various applications and domains.
 - railway, automotive, aeronautics, cybersecurity, nuclear-energy, ...
- The **Event-B method** is adapted to analyse **discrete systems**.
 - offers the possibility of modelling **discrete behaviours**.

THE EVENT-B METHOD

CONTEXT ctx_1
EXTENDS ctx_2

MACHINE mch_1
REFINES mch_2
SEES ctx_i

END

END

THE EVENT-B METHOD

CONTEXT ctx_1
EXTENDS ctx_2

SETS s
CONSTANTS c
AXIOMS

$A(s, c)$

THEOREMS

$T(s, c)$

END

MACHINE mch_1
REFINES mch_2
SEES ctx_i

END

THE EVENT-B METHOD

CONTEXT ctx_1
EXTENDS ctx_2

SETS s
CONSTANTS c
AXIOMS
 $A(s, c)$
THEOREMS
 $T(s, c)$
END

MACHINE mch_1
REFINES mch_2
SEES ctx_i

VARIABLES v
INVARIANTS
 $I(s, c, v)$
THEOREMS
 $T(s, c, v)$
EVENTS
 $[events_list]$
END

event $\hat{=}$
 any x
 where
 $G(s, c, v, x)$
 then
 $BA(s, c, v, x, v')$
 end

THE EVENT-B METHOD

CONTEXT ctx_1
EXTENDS ctx_2

SETS s
CONSTANTS c
AXIOMS
 $A(s, c)$
THEOREMS
 $T(s, c)$
END

MACHINE mch_1
REFINES mch_2
SEES ctx_i

VARIABLES v
INVARIANTS
 $I(s, c, v)$
THEOREMS
 $T(s, c, v)$
EVENTS
 $[events_list]$
END

event $\hat{=}$
 any x
 where
 $G(s, c, v, x)$
 then
 $BA(s, c, v, x, v')$
 end

$$A(s, c) \vdash T(s, c)$$
$$A(s, c) \wedge I(s, c, v) \vdash T(s, c, v)$$

THE EVENT-B METHOD

CONTEXT ctx_1
EXTENDS ctx_2

SETS s
CONSTANTS c
AXIOMS
 $A(s, c)$
THEOREMS
 $T(s, c)$
END

MACHINE mch_1
REFINES mch_2
SEES ctx_i

VARIABLES v
INVARIANTS
 $I(s, c, v)$
THEOREMS
 $T(s, c, v)$
EVENTS
 $[events_list]$
END

event $\hat{=}$
 any x
 where
 $G(s, c, v, x)$
 then
 $BA(s, c, v, x, v')$
 end

$$A(s, c) \vdash T(s, c)$$
$$A(s, c) \wedge I(s, c, v) \vdash T(s, c, v)$$
$$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \wedge BA(s, c, v, x, v') \vdash I(s, c, v')$$

THE EVENT-B METHOD

CONTEXT ctx_1
EXTENDS ctx_2

SETS s
CONSTANTS c
AXIOMS
 $A(s, c)$
THEOREMS
 $T(s, c)$
END

MACHINE mch_1
REFINES mch_2
SEES ctx_i

VARIABLES v
INVARIANTS
 $I(s, c, v)$
THEOREMS
 $T(s, c, v)$
EVENTS
 $[events_list]$
END

event $\hat{=}$
 any x
 where
 $G(s, c, v, x)$
 then
 $BA(s, c, v, x, v')$
 end

$A(s, c) \vdash T(s, c)$
 $A(s, c) \wedge I(s, c, v) \vdash T(s, c, v)$
 $A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \wedge BA(s, c, v, x, v') \vdash I(s, c, v')$
 $A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \vdash \exists v'. BA(s, c, v, x, v')$
...

THE THEORY PLUGIN

- **Theory Plug-in** provides capabilities to **extend the Event-B mathematical language** and **the Rodin proving infrastructure**.

THE THEORY PLUGIN

- **Theory Plug-in** provides capabilities to **extend the Event-B mathematical language** and **the Rodin proving infrastructure**.
- An **Event-B theory** can contain :
 - new datatype definitions,
 - new polymorphic operator definitions,
 - axiomatic definitions,
 - theorems,
 - associated rewrite and inference rules.

THE EVENT-B METHOD

THEORY thy_1

IMPORT thy_2

DATATYPES

DT_1, \dots, DT_n

OPERATORS

OP_{11}, \dots, OP_{1n}

AXIOMATIC DEFINITIONS

operators

OP_{21}, \dots, OP_{2n}

axioms

A

THEOREMS

T

PROOF RULES

PR

END

CONTEXT ctx_1

EXTENDS ctx_2

SETS s

CONSTANTS c

AXIOMS

$A(s, c)$

THEOREMS

$T(s, c)$

END

MACHINE mch_1

REFINES mch_2

SEES ctx_i

VARIABLES v

INVARIANTS

$I(s, c, v)$

THEOREMS

$T(s, c, v)$

EVENTS

$[events_list]$

END

OUTLINE

- The context of the work
- The motivating example
- The proposed approach
- Revisiting the motivating example
- Conclusion and future works

[Back to the begin](#) - [Back to the outline](#)

A SIMPLE EXAMPLE

- System that continuously calculates **a moving object's speed**.

A SIMPLE EXAMPLE

- System that continuously calculates **a moving object's speed**.
- Analysing **two functional properties** :

A SIMPLE EXAMPLE

- System that continuously calculates **a moving object's speed**.
- Analysing **two functional properties** :
 - **PROP-1** : **the speed of the moving object** is equal to the *travaled_distance* divided by the *measured_time* ($v = d/t$).

A SIMPLE EXAMPLE

- System that continuously calculates **a moving object's speed**.
- Analysing **two functional properties** :
 - **PROP-1** : **the speed of the moving object** is equal to the *travaled_distance* divided by the *measured_time* ($v = d/t$).
 - **PROP-2** : when the *travaled_distance* is **strictly positive**, the *speed* of the moving object must also be **strictly positive**.
 - **the object moves** when its *speed* is different from zero.

A SIMPLE EXAMPLE

- System that continuously calculates **a moving object's speed**.
- Analysing **two functional properties** :
 - **PROP-1** : **the speed of the moving object** is equal to the *traveled_distance* divided by the *measured_time* ($v = d/t$).
 - **PROP-2** : when the *traveled_distance* is **strictly positive**, the *speed* of the moving object must also be **strictly positive**.
 - **the object moves** when its *speed* is different from zero.

Objectives → showing some **modelling and validation problems** :

- analysing **physical phenomena**.
 - expressions that come from **the physics laws**.
- using **integer** variables to handle **small values**.

THE EVENT-B MODEL

- System that continuously calculates **a moving object's speed**.
- Analysing **two functional properties** :
 - **PROP-1** : **the speed of the moving object** is equal to the *traveled_distance* divided by the *measured_time* ($v = d/t$).
 - **PROP-2** : when the *traveled_distance* is **strictly positive**, the *speed* of the moving object must also be **strictly positive**.
 - **the object moves** when its *speed* is different from zero.

```
MACHINE mch_integer_version
```

```
...
```

```
INVARIANTS
```

```
@inv1: traveled_distance ∈ N
```

```
@inv2: measured_time ∈ N1
```

```
@inv3: speed ∈ N
```

```
@inv4: starting_position ∈ N
```

```
@inv5: starting_time ∈ N
```


THE EVENT-B MODEL

- System that continuously calculates **a moving object's speed**.
- Analysing **two functional properties** :
 - **PROP-1** : **the speed of the moving object** is equal to the *traveled_distance* divided by the *measured_time* ($v = d/t$).
 - **PROP-2** : when the *traveled_distance* is **strictly positive**, the *speed* of the moving object must also be **strictly positive**.
 - **the object moves** when its *speed* is different from zero.

```
MACHINE mch_integer_version
```

```
...
```

```
INVARIANTS
```

```
@inv1: traveled_distance ∈ ℕ
```

```
@inv2: measured_time ∈ ℕ1
```

```
@inv3: speed ∈ ℕ
```

```
@inv4: starting_position ∈ ℕ
```

```
@inv5: starting_time ∈ ℕ
```

```
@inv6: speed = traveled_distance ÷ measured_time //PROP-1
```

```
@inv7: traveled_distance > 0 ⇒ speed > 0 //PROP-2
```

THE EVENT-B MODEL

```
MACHINE mch_integer_version
...
EVENTS
...
get_starting_point  $\hat{=}$ 
  any p t
  where
    @grd1:  $p \in \mathbb{N}_1$ 
    @grd2:  $t \in \mathbb{N}_1$ 
  then
    @act1: starting_position := p
    @act2: starting_time := t
  end
...
END
```

THE EVENT-B MODEL

```
MACHINE mch_integer_version
...
EVENTS
...
get_speed  $\hat{=}$ 
  any p t
  where
    @grd1:  $p \in \mathbb{N}_1 \wedge p > \text{starting\_position}$ 
    @grd2:  $t \in \mathbb{N}_1 \wedge t > \text{starting\_time}$ 
  then
    @act1:  $\text{traveled\_distance} := p - \text{starting\_position}$ 
    @act2:  $\text{measured\_time} := t - \text{starting\_time}$ 
    @act3:  $\text{speed} := (p - \text{starting\_position}) \div (t - \text{starting\_time})$ 
  end
END
```

GENERATED AND PROVEN POS

- All POs are green **except** the one maintaining the **@inv7** invariant by the *get_speed* event.

- ✓ mch_integer_version
 - > Variables
 - > Invariants
 - > Events
 - ✓ Proof Obligations
 - ✓ inv6/WD
 - ✓ INITIALISATION/inv1/INV
 - ✓ INITIALISATION/inv2/INV
 - ✓ INITIALISATION/inv3/INV
 - ✓ INITIALISATION/inv4/INV
 - ✓ INITIALISATION/inv5/INV
 - ✓ INITIALISATION/inv6/INV
 - ✓ INITIALISATION/inv7/INV
 - ✓ get_starting_point/inv4/INV
 - ✓ get_starting_point/inv5/INV
 - ✓ get_speed/inv1/INV
 - ✓ get_speed/inv2/INV
 - ✓ get_speed/inv3/INV
 - ✓ get_speed/inv6/INV
 - ✗ get_speed/inv7/INV
 - ✓ get_speed/act3/WD

GENERATED AND PROVEN POS

- All POs are green **except** the one maintaining the **@inv7** invariant by the *get_speed* event.
- This invariant formalises the **PROP 2** property.
 - **the object moves** (*traveled_distance* $\neq 0$) when *speed* $\neq 0$.

- ✓ mch_integer_version
 - > Variables
 - > Invariants
 - > Events
 - ✓ Proof Obligations
 - ✓ inv6/WD
 - ✓ INITIALISATION/inv1/INV
 - ✓ INITIALISATION/inv2/INV
 - ✓ INITIALISATION/inv3/INV
 - ✓ INITIALISATION/inv4/INV
 - ✓ INITIALISATION/inv5/INV
 - ✓ INITIALISATION/inv6/INV
 - ✓ INITIALISATION/inv7/INV
 - ✓ get_starting_point/inv4/INV
 - ✓ get_starting_point/inv5/INV
 - ✓ get_speed/inv1/INV
 - ✓ get_speed/inv2/INV
 - ✓ get_speed/inv3/INV
 - ✓ get_speed/inv6/INV
 - ✗ get_speed/inv7/INV
 - ✓ get_speed/act3/WD

GENERATED AND PROVEN POS

- All POs are green **except** the one maintaining the **@inv7** invariant by the *get_speed* event.
- This invariant formalises the **PROP 2** property.
 - **the object moves** (*traveled_distance* $\neq 0$) when *speed* $\neq 0$.
- The *get_speed* event calculates the new value of *traveled_distance* that can be $<$ the new value of *measured_time*.
 - the new value of *speed* (*traveled_distance* \div *measured_time*) can be $= 0$ while *traveled_distance* $\neq 0$
 - \div makes **an integer division**

```

v mch_integer_version
> Variables
> Invariants
> Events
v Proof Obligations
  inv6/WD
  INITIALISATION/inv1/INV
  INITIALISATION/inv2/INV
  INITIALISATION/inv3/INV
  INITIALISATION/inv4/INV
  INITIALISATION/inv5/INV
  INITIALISATION/inv6/INV
  INITIALISATION/inv7/INV
  get_starting_point/inv4/INV
  get_starting_point/inv5/INV
  get_speed/inv1/INV
  get_speed/inv2/INV
  get_speed/inv3/INV
  get_speed/inv6/INV
  get_speed/inv7/INV
  get_speed/act3/WD
```

CONCLUSION

The basic types and operators of the Event-B language
are not adapted to our needs

OUTLINE

- The context of the work
- The motivating example
- The proposed approach
- Revisiting the motivating example
- Conclusion and future works

[Back to the begin](#) - [Back to the outline](#)

FLOATING-POINT NUMBERS

$$x = 3.14159265359 = \underbrace{314159265359}_{\text{significand}} \times \underbrace{10}_{\text{base}} \overset{\text{exponent}}{\overbrace{-11}}$$

FLOATING-POINT NUMBERS

$$x = 3.14159265359 = \underbrace{314159265359}_{\text{significand}} \times \underbrace{10}_{\text{base}}^{\text{exponent } -11}$$

We have chosen that the base always equals ten in our models.

$$x = s(x) \times 10^{e(x)}$$

- The proposed theory **does not model limited precision**.
- The **operators** defined in the theory involve **no precision loss**.

THE PROPOSED APPROACH

- To allow the **Event-B language** to **embed** this **FP representation**, we need to define two theories :

THE PROPOSED APPROACH

- To allow the **Event-B language** to **embed** this **FP representation**, we need to define two theories :
 1. the first one formalises **the power operator**.

THE PROPOSED APPROACH

- To allow the **Event-B language** to **embed** this **FP representation**, we need to define two theories :

1. the first one formalises **the power operator**.

✗ $\hat{}$ **operator** is **not implemented** in the automated proofs besides $\hat{0}$ and $\hat{1}$.

THE PROPOSED APPROACH

- To allow the **Event-B language** to **embed** this **FP representation**, we need to define two theories :
 1. the first one formalises **the power operator**.
 - ✗ **\wedge operator** is **not implemented** in the automated proofs besides \wedge^0 and \wedge^1 .
 2. the second one formalises **floating-point numbers** by specifying :
 - the corresponding **data type**,
 - the supported **arithmetic operators**,
 - some **axioms** and **theorems** that characterise the proposed modelling.

THE POWER OPERATOR

THEORY thy_power_operator

AXIOMATIC DEFINITIONS

operators

pow($x \in \mathbb{Z}$, $n \in \mathbb{N}$) : \mathbb{Z} INFIX // x pow $n = x^n$

wd condition : $\neg (x = 0 \wedge n = 0)$ // 0^0 is not defined

THE POWER OPERATOR

THEORY thy_power_operator

AXIOMATIC DEFINITIONS

operators

pow($x \in \mathbb{Z}$, $n \in \mathbb{N}$) : \mathbb{Z} INFIX // x pow $n = x^n$
wd condition : $\neg (x = 0 \wedge n = 0)$ // 0^0 is not defined

axioms

@axm1: $\forall n. n \in \mathbb{N}_1 \Rightarrow 0 \text{ pow } n = 0$
@axm2: $\forall x. x \in \mathbb{Z} \wedge x \neq 0 \Rightarrow x \text{ pow } 0 = 1$
@axm3: $\forall x, n. x \in \mathbb{Z} \wedge x \neq 0 \wedge n \in \mathbb{N}_1 \Rightarrow x \text{ pow } n = x \times (x \text{ pow } (n - 1))$
...

THE POWER OPERATOR

THEORY thy_power_operator

AXIOMATIC DEFINITIONS

operators

pow($x \in \mathbb{Z}$, $n \in \mathbb{N}$) : \mathbb{Z} INFIX // x pow $n = x^n$
wd condition : $\neg (x = 0 \wedge n = 0)$ // 0^0 is not defined

axioms

@axm1: $\forall n. n \in \mathbb{N}_1 \Rightarrow 0 \text{ pow } n = 0$
@axm2: $\forall x. x \in \mathbb{Z} \wedge x \neq 0 \Rightarrow x \text{ pow } 0 = 1$
@axm3: $\forall x, n. x \in \mathbb{Z} \wedge x \neq 0 \wedge n \in \mathbb{N}_1 \Rightarrow x \text{ pow } n = x \times (x \text{ pow } (n - 1))$
...

THEOREMS

@thm1: $\forall x, n, m. \dots \Rightarrow x \text{ pow } (n + m) = (x \text{ pow } n) \times (x \text{ pow } m)$
@thm2: $\forall x, n, m. \dots \Rightarrow (x \text{ pow } n) \text{ pow } m = x \text{ pow } (n \times m)$
@thm3: $\forall x, y, n. \dots \Rightarrow (x \times y) \text{ pow } n = (x \text{ pow } n) \times (y \text{ pow } n)$
...

END

SOME REMARKS

- By using this theory, it **becomes possible to prove**, for example, that
 $5 \text{ pow } 3 = 125$

SOME REMARKS

- By using this theory, it **becomes possible to prove**, for example, that
 $5 \text{ pow } 3 = 125$
- **The proofs** of all theorems were made by **induction**
(following the rules defined by **Cervelle and Gervais - ABZ 2023**).

SOME REMARKS

- By using this theory, it **becomes possible to prove**, for example, that
 $5 \text{ pow } 3 = 125$
- **The proofs** of all theorems were made by **induction**
(following the rules defined by **Cervelle and Gervais - ABZ 2023**).
- We have chosen to define the **pow** operator in a **single theory** to offer the possibility of **reusing it** in other **Event-B components**.

THE FLOATING-POINT NUMBERS THEORY

THEORY thy_floating_point_numbers

DATATYPES

$\text{FLOAT_Type} \triangleq \text{NEW_FLOAT}(s \in \mathbb{Z}, e \in \mathbb{Z}) \text{ // } x = s(x) \times 10^{e(x)}$

THE FLOATING-POINT NUMBERS THEORY

THEORY thy_floating_point_numbers

DATATYPES

$\text{FLOAT_Type} \triangleq \text{NEW_FLOAT}(s \in \mathbb{Z}, e \in \mathbb{Z}) \text{ // } x = s(x) \times 10^{e(x)}$

OPERATORS

$F0 \triangleq \text{NEW_FLOAT}(0,0) \text{ // } 0 = 0 \times 10^0$

$F1 \triangleq \text{NEW_FLOAT}(1,0) \text{ // } 1 = 1 \times 10^0$

$\text{FLOAT1_Type} = \{ x \cdot x \in \text{FLOAT_Type} \wedge s(x) \neq 0 \mid x \}$

$\text{FLOAT}(x \in \mathbb{Z}) \triangleq \text{NEW_FLOAT}(x,0) \text{ // } x = x \times 10^0$

THE FLOATING-POINT NUMBERS THEORY

THEORY thy_floating_point_numbers

DATATYPES

$\text{FLOAT_Type} \triangleq \text{NEW_FLOAT}(s \in \mathbb{Z}, e \in \mathbb{Z}) \text{ // } x = s(x) \times 10^{e(x)}$

OPERATORS

$F0 \triangleq \text{NEW_FLOAT}(0,0) \text{ // } 0 = 0 \times 10^0$

$F1 \triangleq \text{NEW_FLOAT}(1,0) \text{ // } 1 = 1 \times 10^0$

$\text{FLOAT1_Type} = \{ x \cdot x \in \text{FLOAT_Type} \wedge s(x) \neq 0 \mid x \}$

$\text{FLOAT}(x \in \mathbb{Z}) \triangleq \text{NEW_FLOAT}(x,0) \text{ // } x = x \times 10^0$

$\text{l_shift}(x \in \text{FLOAT_Type}, \text{offset} \in \mathbb{N}) \triangleq$
 $\text{NEW_FLOAT}(s(x) \times (10^{\text{pow offset}}), e(x) - \text{offset})$

THE FLOATING-POINT NUMBERS THEORY

THEORY thy_floating_point_numbers

DATATYPES

$\text{FLOAT_Type} \triangleq \text{NEW_FLOAT}(s \in \mathbb{Z}, e \in \mathbb{Z}) \text{ // } x = s(x) \times 10^{e(x)}$

OPERATORS

$F0 \triangleq \text{NEW_FLOAT}(0,0) \text{ // } 0 = 0 \times 10^0$

$F1 \triangleq \text{NEW_FLOAT}(1,0) \text{ // } 1 = 1 \times 10^0$

$\text{FLOAT1_Type} = \{ x \cdot x \in \text{FLOAT_Type} \wedge s(x) \neq 0 \mid x \}$

$\text{FLOAT}(x \in \mathbb{Z}) \triangleq \text{NEW_FLOAT}(x,0) \text{ // } x = x \times 10^0$

$\text{l_shift}(x \in \text{FLOAT_Type}, \text{offset} \in \mathbb{N}) \triangleq$
 $\text{NEW_FLOAT}(s(x) \times (10 \text{ pow offset}), e(x) - \text{offset})$

$\text{eq}(x \in \text{FLOAT_Type}, y \in \text{FLOAT_Type}) \text{ INFIX} \triangleq$
 $s(\text{l_shift}(x, e(x) - \min(\{e(x), e(y)\}))) = s(\text{l_shift}(y, e(y) - \min(\{e(x), e(y)\})))$

$\text{gt}(x \in \text{FLOAT_Type}, y \in \text{FLOAT_Type}) \text{ INFIX} \triangleq \dots$

$\text{geq}(x \in \text{FLOAT_Type}, y \in \text{FLOAT_Type}) \text{ INFIX} \triangleq \dots$

$\text{lt}(x \in \text{FLOAT_Type}, y \in \text{FLOAT_Type}) \text{ INFIX} \triangleq \dots$

$\text{leq}(x \in \text{FLOAT_Type}, y \in \text{FLOAT_Type}) \text{ INFIX} \triangleq \dots$

THE FLOATING-POINT NUMBERS THEORY

THEORY thy_floating_point_numbers

...

OPERATORS

...

plus($x \in \text{FLOAT_Type}$, $y \in \text{FLOAT_Type}$) INFIX $\hat{=}$
NEW_FLOAT($s(l_shift(x, e(x) - \min(\{e(x), e(y)\}))) + s(l_shift(y, e(y) - \min(\{e(x), e(y)\})))$, $\min(\{e(x), e(y)\})$)

minus($x \in \text{FLOAT_Type}$, $y \in \text{FLOAT_Type}$) INFIX $\hat{=}$...

neg($x \in \text{FLOAT_Type}$) $\hat{=}$...

THE FLOATING-POINT NUMBERS THEORY

THEORY thy_floating_point_numbers

...

OPERATORS

...

plus($x \in \text{FLOAT_Type}$, $y \in \text{FLOAT_Type}$) INFIX $\hat{=}$
NEW_FLOAT($s(l_shift(x, e(x) - \min(\{e(x), e(y)\})) + s(l_shift(y, e(y) - \min(\{e(x), e(y)\})))$, $\min(\{e(x), e(y)\})$)

minus($x \in \text{FLOAT_Type}$, $y \in \text{FLOAT_Type}$) INFIX $\hat{=}$...
neg($x \in \text{FLOAT_Type}$) $\hat{=}$...

mult($x \in \text{FLOAT_Type}$, $y \in \text{FLOAT_Type}$) INFIX $\hat{=}$
NEW_FLOAT($s(x) \times s(y)$, $e(x) + e(y)$)

f_pow($x \in \text{FLOAT_Type}$, $n \in \mathbb{N}$) INFIX $\hat{=}$
NEW_FLOAT($s(x)$ pow n , $e(x) \times n$)

THE FLOATING-POINT NUMBERS THEORY

THEORY thy_floating_point_numbers

...

OPERATORS

...

plus($x \in \text{FLOAT_Type}$, $y \in \text{FLOAT_Type}$) INFIX $\hat{=}$
NEW_FLOAT($s(\text{l_shift}(x, e(x) - \min(\{e(x), e(y)\})) + s(\text{l_shift}(y, e(y) - \min(\{e(x), e(y)\})))$, $\min(\{e(x), e(y)\})$))

minus($x \in \text{FLOAT_Type}$, $y \in \text{FLOAT_Type}$) INFIX $\hat{=}$...
neg($x \in \text{FLOAT_Type}$) $\hat{=}$...

mult($x \in \text{FLOAT_Type}$, $y \in \text{FLOAT_Type}$) INFIX $\hat{=}$
NEW_FLOAT($s(x) \times s(y)$, $e(x) + e(y)$)

f_pow($x \in \text{FLOAT_Type}$, $n \in \mathbb{N}$) INFIX $\hat{=}$
NEW_FLOAT($s(x)$ pow n , $e(x) \times n$)

floor($x \in \text{FLOAT_Type}$) $\hat{=}$...
ceiling($x \in \text{FLOAT_Type}$) $\hat{=}$...
integer($x \in \text{FLOAT_Type}$) $\hat{=}$...
frac($x \in \text{FLOAT_Type}$) $\hat{=}$...

...

END

Laboratoire
Méthodes
Formelles



THE CASE OF `inv` AND `div` OPERATORS

- The proposed theory involves **no precision loss** for **plus** and **mult** operators.

THE CASE OF `inv` AND `div` OPERATORS

- The proposed theory involves **no precision loss** for **plus** and **mult** operators.
- The **division** sometimes **induces a precision loss**.
 - ✗ **ex.** we cannot precisely represent the result of $1/3$ or $2/3$

THE CASE OF `inv` AND `div` OPERATORS

- The proposed theory involves **no precision loss** for **plus** and **mult** operators.
- The **division** sometimes **induces a precision loss**.
 - ✗ **ex.** we cannot precisely represent the result of $1/3$ or $2/3$
- For the case of **inv** and **div** operators, we have defined **the Well-definedness conditions**.

THE CASE OF *inv* AND *div* OPERATORS

- The proposed theory involves **no precision loss** for **plus** and **mult** operators.
- The **division** sometimes **induces a precision loss**.
 - ✗ **ex.** we cannot precisely represent the result of $1/3$ or $2/3$
- For the case of **inv** and **div** operators, we have defined **the Well-definedness conditions**.
 - To calculate $inv(x)$, we must find a z , with $10^n = z \times s(x)$.
 - ✓ $inv(2.5) = 1/2.5 = 0.4 = 4 \times 10^{-1}$ ($z = 4$ because $100 = 4 \times 25$)
 - ✗ $inv(3) = 1/3 = 0.3333\dots$ (z **does not exist**)

THE CASE OF *inv* AND *div* OPERATORS

- The proposed theory involves **no precision loss** for **plus** and **mult** operators.
- The **division** sometimes **induces a precision loss**.
 - ✗ ex. we cannot precisely represent the result of $1/3$ or $2/3$
- For the case of **inv** and **div** operators, we have defined **the Well-definedness conditions**.
 - To calculate $inv(x)$, we must find a z , with $10^n = z \times s(x)$.
 - ✓ $inv(2.5) = 1/2.5 = 0.4 = 4 \times 10^{-1}$ ($z = 4$ because $100 = 4 \times 25$)
 - ✗ $inv(3) = 1/3 = 0.3333\dots$ (z **does not exist**)
 - To calculate $x \text{ div } y$, we must find a z , with $10^n \times s(x) = z \times s(y)$.
 - ✓ $2 \text{ div } 5 = 2/5 = 0.4 = 4 \times 10^{-1}$ ($z = 4$ because $10 \times 2 = 4 \times 5$)
 - ✗ $2 \text{ div } 3 = 2/3 = 0.6666\dots$ (z **does not exist**)

THE CASE OF `inv` AND `div` OPERATORS

THEORY `thy_floating_point_numbers`

...

OPERATORS

...

`inv_WD(a ∈ FLOAT1_Type) ≡`

`∃ n,z. n ∈ ℕ ∧ z ∈ ℤ ∧ 10 pow n = s(a) × z`

`div_WD(a ∈ FLOAT_Type, b ∈ FLOAT1_Type) ≡`

`∃ n,z. n ∈ ℕ ∧ z ∈ ℤ ∧ s(a) × (10 pow n) = s(b) × z`

AXIOMATIC DEFINITIONS

operators

`inv(x ∈ FLOAT_Type) : FLOAT1_Type`

wd condition : `inv_WD(x)`

...

axioms

@axm1: `∀ x,y.(... ⇒ ((x mult y) = F1 ⇔ inv(x) = y))`

@axm2: `∀ x,y.(... ⇒ ((x mult y) eq F1 ⇔ inv(x) eq y))`

...

END

THE CASE OF `inv` AND `div` OPERATORS

THEORY thy_floating_point_numbers

...

OPERATORS

...

$\text{inv_WD}(a \in \text{FLOAT1_Type}) \triangleq$
 $\exists n, z. n \in \mathbb{N} \wedge z \in \mathbb{Z} \wedge 10^{\text{pow } n} = s(a) \times z$

$\text{div_WD}(a \in \text{FLOAT_Type}, b \in \text{FLOAT1_Type}) \triangleq$
 $\exists n, z. n \in \mathbb{N} \wedge z \in \mathbb{Z} \wedge s(a) \times (10^{\text{pow } n}) = s(b) \times z$

AXIOMATIC DEFINITIONS

operators

...

$\text{div}(x \in \text{FLOAT_Type}, y \in \text{FLOAT_Type}) : \text{FLOAT_Type}$ INFIX

wd condition : $\text{div_WD}(x, y)$

...

axioms

@axm1: $\forall x, y, z. (\dots \Rightarrow ((y \text{ mult } z) = x \Leftrightarrow (x \text{ div } y) = z))$

@axm2: $\forall x, y, z. (\dots \Rightarrow ((y \text{ mult } z) \text{ eq } x \Leftrightarrow (x \text{ div } y) \text{ eq } z))$

@axm3: $\forall x, y. (\dots \Rightarrow x \text{ mult inv}(y) = x \text{ div } y)$

...

END

THE FLOATING-POINT NUMBERS THEORY

THEORY thy_floating_point_numbers

...

THEOREMS

@thm1: $\forall x, y. (\dots \Rightarrow x \text{ eq } y \Leftrightarrow y \text{ eq } x)$
@thm2: $\forall x. (\dots \Rightarrow x \text{ geq } x \wedge x \text{ leq } x)$
@thm3: $\forall x, y. (\dots \wedge x \text{ leq } y \wedge y \text{ leq } x \Rightarrow x \text{ eq } y)$
@thm4: $\forall x, y. (\dots \Rightarrow x \text{ leq } y \vee y \text{ leq } x)$
@thm5: $\forall x, y, z. (\dots \wedge x \text{ leq } y \wedge y \text{ leq } z \Rightarrow x \text{ leq } z)$
@thm6: $\forall x, y, z. (\dots \wedge x \text{ leq } y \Rightarrow (x \text{ plus } z) \text{ leq } (y \text{ plus } z))$
@thm7: $\forall x, y, z. (\dots \wedge x \text{ leq } y \Rightarrow (x \text{ mult } z) \text{ leq } (y \text{ mult } z))$
@thm8: $\forall x. (\dots \Rightarrow x \text{ plus } F0 \text{ eq } x)$
@thm9: $\forall x, y. (\dots \Rightarrow x \text{ plus } y = y \text{ plus } x)$
@thm10: $\forall x, y. (\dots \Rightarrow x \text{ plus } \text{neg}(y) = y \text{ minus } x)$
@thm11: $\forall x. (\dots \Rightarrow x \text{ minus } F0 \text{ eq } x)$
@thm12: $\forall x. (\dots \Rightarrow x \text{ minus } x \text{ eq } F0)$
@thm13: $\forall x. (\dots \Rightarrow x \text{ mult } F0 \text{ eq } F0)$
@thm14: $\forall x. (\dots \Rightarrow x \text{ mult } F1 = x)$
@thm15: $\forall x, y. (\dots \Rightarrow x \text{ mult } y = y \text{ mult } x)$
@thm16: $\forall x. (\dots \Rightarrow \text{inv}(x) = F1 \text{ div } x)$
@thm17: $\forall x. (\dots \Rightarrow x \text{ div } F1 = x)$
@thm18: $\forall x. (\dots \Rightarrow x \text{ div } x = F1)$
@thm19: $\forall x. (\dots \Rightarrow x \text{ mult } \text{inv}(x) = F1)$

SOME REMARKS

- Due to our choice to formalise **unlimited precision FP** numbers, some **properties** that are **not true** in the FP numbers world **can be deduced**.

SOME REMARKS

- Due to our choice to formalise **unlimited precision FP** numbers, some **properties** that are **not true** in the FP numbers world **can be deduced**.
 - the associativity of addition and multiplication, for example

SOME REMARKS

- Due to our choice to formalise **unlimited precision FP** numbers, some **properties** that are **not true** in the FP numbers world **can be deduced**.
 - the associativity of addition and multiplication, for example
- If this theory **is refined** (towards the **IEEE Standard 754**, for example), the developer must **pay attention** to this point.

OUTLINE

- The context of the work
- The motivating example
- The proposed approach
- **Revisiting the motivating example**
- Conclusion and future works

[Back to the begin](#) - [Back to the outline](#)

NATURAL VARIABLES

All **NATURAL** variables are typed by **PFLOAT_Type** set containing **positive floating-point numbers**.

THEORY thy_floating_point_numbers

...

$\text{PFLOAT_Type} = \{ x \cdot x \in \text{FLOAT_Type} \wedge s(x) \geq 0 \mid x \}$

$\text{PFLOAT1_Type} = \{ x \cdot x \in \text{FLOAT_Type} \wedge s(x) > 0 \mid x \}$

...

END

REVISITING OUR EXAMPLE I

MACHINE mch_integer_version

...

INVARIANTS

@inv1: traveled_distance $\in \mathbb{N}$

@inv2: measured_time $\in \mathbb{N}_1$

@inv3: speed $\in \mathbb{N}$

@inv4: starting_position $\in \mathbb{N}$

@inv5: starting_time $\in \mathbb{N}$

@inv6: speed = traveled_distance \div measured_time

@inv7: traveled_distance $> 0 \Rightarrow$ speed > 0

...

END

REVISITING OUR EXAMPLE I

MACHINE mch_floating_point_version

...

INVARIANTS

@inv1: traveled_distance \in PFLOAT_Type

@inv2: measured_time \in PFLOAT1_Type

@inv3: speed \in PFLOAT_Type

@inv4: starting_position \in PFLOAT_Type

@inv5: starting_time \in PFLOAT_Type

@inv7: speed eq traveled_distance div measured_time

@inv8: traveled_distance $\geq 0 \Rightarrow$ speed ≥ 0

...

END

REVISITING OUR EXAMPLE I

MACHINE mch_floating_point_version

...

INVARIANTS

@inv1: traveled_distance \in PFLOAT_Type

@inv2: measured_time \in PFLOAT_Type

@inv3: speed \in PFLOAT_Type

@inv4: starting_position \in PFLOAT_Type

@inv5: starting_time \in PFLOAT_Type

@inv6: div_WD(traveled_distance, measured_time)

@inv7: speed eq traveled_distance div measured_time

@inv8: traveled_distance gt F0 \Rightarrow speed gt F0

...

END

REVISITING OUR EXAMPLE II

MACHINE mch_integer_version

...

EVENTS

...

get_speed $\hat{=}$

any p t

where

@grd1: $p \in \mathbb{N}_1 \wedge p > \text{starting_position}$

@grd2: $t \in \mathbb{N}_1 \wedge t > \text{starting_time}$

then

@act1: $\text{traveled_distance} := p - \text{starting_position}$

@act2: $\text{measured_time} := t - \text{starting_time}$

@act3: $\text{speed} := (p - \text{starting_position}) \div (t - \text{starting_time})$

end

END

REVISITING OUR EXAMPLE II

MACHINE mch_floating_point_version

...

EVENTS

...

get_speed $\hat{=}$

any p t

where

@grd1: $p \in \text{PFLOAT_Type} \wedge p \text{ gt starting_position}$

@grd2: $t \in \text{PFLOAT_Type} \wedge t \text{ gt starting_time}$

then

@act1: traveled_distance := p minus starting_position

@act2: measured_time := t minus starting_time

@act3: speed := (p minus starting_position) div (t minus starting_time)

end

END

REVISITING OUR EXAMPLE II

MACHINE mch_floating_point_version

...

EVENTS

...

get_speed $\hat{=}$

any p t

where

@grd1: $p \in \text{PFLOAT_Type} \wedge p \text{ gt starting_position}$

@grd2: $t \in \text{PFLOAT_Type} \wedge t \text{ gt starting_time}$

@grd3: $\text{div_WD}(p \text{ minus starting_position}, t \text{ minus starting_time})$

then

@act1: $\text{traveled_distance} := p \text{ minus starting_position}$

@act2: $\text{measured_time} := t \text{ minus starting_time}$

@act3: $\text{speed} := (p \text{ minus starting_position}) \text{ div } (t \text{ minus starting_time})$

end

END

GENERATED AND PROVEN POS

- All generated POs have been proven.

- ✓ M mch_floating_point_speed
 - > Variables
 - > Invariants
 - > Events
 - ✓ Proof Obligations
 - ✓ inv6/WD
 - ✓ inv7/WD
 - ✓ INITIALISATION/inv1/INV
 - ✓ INITIALISATION/inv2/INV
 - ✓ INITIALISATION/inv3/INV
 - ✓ INITIALISATION/inv4/INV
 - ✓ INITIALISATION/inv5/INV
 - ✓ INITIALISATION/inv6/INV
 - ✓ INITIALISATION/inv7/INV
 - ✓ INITIALISATION/inv8/INV
 - ✓ get_starting_point/inv4/INV
 - ✓ get_starting_point/inv5/INV
 - ✓ get_speed/grd5/WD
 - ✓ get_speed/inv1/INV
 - ✓ get_speed/inv2/INV
 - ✓ get_speed/inv3/INV
 - ✓ get_speed/inv6/INV
 - ✓ get_speed/inv7/INV
 - ✓ get_speed/inv8/INV
 - ✓ get_speed/act3/WD

GENERATED AND PROVEN POS

- All generated POs have been proven.
- The `get_speed/inv8/INV` PO becomes ✓.
 - ➡ thanks to handling small values (`]0..1[`),
 - ➡ and to the new `div` operator specification.

- ✓ `mch_floating_point_speed`
 - > Variables
 - > Invariants
 - > Events
 - ✓ Proof Obligations
 - ✓ `inv6/WD`
 - ✓ `inv7/WD`
 - ✓ `INITIALISATION/inv1/INV`
 - ✓ `INITIALISATION/inv2/INV`
 - ✓ `INITIALISATION/inv3/INV`
 - ✓ `INITIALISATION/inv4/INV`
 - ✓ `INITIALISATION/inv5/INV`
 - ✓ `INITIALISATION/inv6/INV`
 - ✓ `INITIALISATION/inv7/INV`
 - ✓ `INITIALISATION/inv8/INV`
 - ✓ `get_starting_point/inv4/INV`
 - ✓ `get_starting_point/inv5/INV`
 - ✓ `get_speed/grd5/WD`
 - ✓ `get_speed/inv1/INV`
 - ✓ `get_speed/inv2/INV`
 - ✓ `get_speed/inv3/INV`
 - ✓ `get_speed/inv6/INV`
 - ✓ `get_speed/inv7/INV`
 - ✓ `get_speed/inv8/INV`
 - ✓ `get_speed/act3/WD`

GENERATED AND PROVEN POS

- All generated POs have been proven.
- The `get_speed/inv8/INV` PO becomes ✓.
 - ➡ thanks to handling small values (`]0..1[`),
 - ➡ and to the new `div` operator specification.

The floating-point numbers theory is more suitable than the basic integers of Event-B.

- ✓ M mch_floating_point_speed
 - > Variables
 - > Invariants
 - > Events
 - ✓ Proof Obligations
 - ✓ inv6/WD
 - ✓ inv7/WD
 - ✓ INITIALISATION/inv1/INV
 - ✓ INITIALISATION/inv2/INV
 - ✓ INITIALISATION/inv3/INV
 - ✓ INITIALISATION/inv4/INV
 - ✓ INITIALISATION/inv5/INV
 - ✓ INITIALISATION/inv6/INV
 - ✓ INITIALISATION/inv7/INV
 - ✓ INITIALISATION/inv8/INV
 - ✓ get_starting_point/inv4/INV
 - ✓ get_starting_point/inv5/INV
 - ✓ get_speed/grd5/WD
 - ✓ get_speed/inv1/INV
 - ✓ get_speed/inv2/INV
 - ✓ get_speed/inv3/INV
 - ✓ get_speed/inv6/INV
 - ✓ get_speed/inv7/INV
 - ✓ get_speed/inv8/INV
 - ✓ get_speed/act3/WD

OUTLINE

- The context of the work
- The motivating example
- The proposed approach
- Revisiting the motivating example
- Conclusion and future works

[Back to the begin](#) - [Back to the outline](#)

CONCLUSION

- Extending the **Event-B type-checking system** by an approach using the **theory plugin**.

CONCLUSION

- Extending the **Event-B type-checking system** by an approach using the **theory plugin**.
- Development of a **floating point number theory** formalising floating point numbers.
 - an extension of the **Event-B power operator**.
 - an **abstract representation** of the **floating-point numbers**.
 - a set of theorems and associated **rewrite** and **inference rules**.

FUTURE WORKS

- Refining the proposed theory to any **more concrete implementation** (the **IEEE standard 754**, for example).

FUTURE WORKS

- Refining the proposed theory to any **more concrete implementation** (the **IEEE standard 754**, for example).
- Developing a **more general theory** formalising the standard units of **measurement** defined by the **International System of Units (SI)**.
 - extends the **floating point number theory**.
 - helpful in **modelling cyber-physical/hybrid** systems.

THANK YOU

[Back to the begin](#) - [Back to the outline](#)