

# A FLOATING-POINT NUMBERS THEORY FOR EVENT-B

🎓 12<sup>th</sup> International Conference on Model & Data Engineering (MEDI 2023)

🏛️ Sousse, Tunisia 📅 2-4 November 2023



**Idir AIT SADOUNE**  
[idir.aitsadoue@centralesupelec.fr](mailto:idir.aitsadoue@centralesupelec.fr)

# OUTLINE

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

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

# OUTLINE

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

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

# THE EVENT-B METHOD

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



# USING EVENT-B METHOD

- The use of the **Event-B method** has continued to increase.
  - applied to various applications and domains.
- The **Event-B method** is adapted to analyse **discrete systems**.
  - offers the possibility of modelling **discrete behaviors**.

# 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**

$A(s, c) \vdash T(s, c)$

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

$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \vdash \exists v'. BA(s, c, v, x, v')$

$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \wedge BA(s, c, v, x, v') \vdash I(s, c, v')$

...

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

# A SIMPLE EXAMPLE

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

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

# THE EVENT-B MODEL

- Analyzing **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: distance_travelled ∈ ℕ
```

```
@inv2: measured_time ∈ ℕ1
```

```
@inv3: speed ∈ ℕ
```

```
@inv4: starting_position ∈ ℕ
```

```
@inv5: starting_time ∈ ℕ
```

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

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

# 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:  $\text{distance\_travelled} := p - \text{starting\_position}$ 
    @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.
- 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)

- ✓ 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

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

# FLOATING-POINT NUMBERS

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

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:

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 characterize 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** //  $x \text{ pow } n = x^n$

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

### axioms

@axm1:  $\forall n \cdot n \in \mathbb{N}_1 \Rightarrow 0 \text{ pow } n = 0$

@axm2:  $\forall x \cdot x \in \mathbb{Z} \wedge x \neq 0 \Rightarrow x \text{ pow } 0 = 1$

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

...

## THEOREMS

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

@thm2:  $\forall x, n, m \cdot \dots \Rightarrow (x \text{ pow } n) \text{ pow } m = x \text{ pow } (n \times m)$

@thm3:  $\forall x, y, n \cdot \dots \Rightarrow (x \times y) \text{ pow } n = (x \text{ pow } n) \times (y \text{ pow } n)$

...

**END**

MEDI 2023

12<sup>th</sup> International Conference on Model and Data Engineering  
2-4 of November 2023, Sousse, Tunisia

# 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

$\text{FLOAT\_Type} \hat{=} \text{NEW\_FLOAT}(s \in \mathbb{Z}, e \in \mathbb{Z}) \quad // \quad x = s(x) \times 10^{e(x)}$

## OPERATORS

$F0 \hat{=} \text{NEW\_FLOAT}(0,0) \quad // \quad 0 = 0 \times 10^0$

$F1 \hat{=} \text{NEW\_FLOAT}(1,0) \quad // \quad 1 = 1 \times 10^0$

$\text{FLOAT1\_Type} \hat{=} \{ x \cdot x \in \text{FLOAT\_Type} \wedge s(x) \neq 0 \mid x \}$

$\text{FLOAT}(x \in \mathbb{Z}) \hat{=} \text{NEW\_FLOAT}(x,0) \quad // \quad x = x \times 10^0$

$\text{l\_shift}(x \in \text{FLOAT\_Type}, \text{offset} \in \mathbb{N}) \hat{=}$   
 $\text{NEW\_FLOAT}(s(x) \times (10 \text{ pow offset}), e(x) - \text{offset})$

$\text{eq}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) \text{ INFIX } \hat{=}$   
 $s(\text{l\_shift}(x, e(x) - \min(\{e(x), e(y)\}))) = s(\text{l\_shift}(y, e(y) - \min(\{e(x), e(y)\})))$

$\text{gt}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) \text{ INFIX } \hat{=} \dots$

...

**END**

# THE FLOATING-POINT NUMBERS THEORY

**THEORY** thy\_floating\_point\_numbers

## DATATYPES

$\text{FLOAT\_Type} \hat{=} \text{NEW\_FLOAT}(s \in \mathbb{Z}, e \in \mathbb{Z}) \text{ // } x = s(x) \times 10^{e(x)}$

## OPERATORS

...

$\text{geq}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) \text{ INFIX } \hat{=} \dots$

$\text{lt}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) \text{ INFIX } \hat{=} \dots$

$\text{leq}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) \text{ INFIX } \hat{=} \dots$

...

$\text{plus}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) \text{ INFIX } \hat{=}$

$\text{NEW\_FLOAT}(s(\text{l\_shift}(x, e(x) - \min(\{e(x), e(y)\})))) + s(\text{l\_shift}(y, e(y) - \min(\{e(x), e(y)\}))),$   
 $\text{min}(\{e(x), e(y)\}))$

$\text{minus}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) \text{ INFIX } \hat{=}$

$\text{neg}(x \in \text{FLOAT\_Type}) \hat{=} \dots$

...

**END**

# THE FLOATING-POINT NUMBERS THEORY

**THEORY** thy\_floating\_point\_numbers

...

**OPERATORS**

...

$\text{mult}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) \text{ INFIX } \hat{=} \text{NEW\_FLOAT}(s(x) \times s(y), e(x) + e(y))$

$\text{f\_pow}(x \in \text{FLOAT\_Type}, n \in \mathbb{N}) \text{ INFIX } \hat{=} \text{NEW\_FLOAT}(s(x) \text{ pow } n, e(x) \times n)$

$\text{floor}(x \in \text{FLOAT\_Type}) \hat{=} \dots$

$\text{ceiling}(x \in \text{FLOAT\_Type}) \hat{=} \dots$

$\text{integer}(x \in \text{FLOAT\_Type}) \hat{=} \dots$

$\text{frac}(x \in \text{FLOAT\_Type}) \hat{=} \dots$

...

**END**

# 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**

...

$\text{inv\_WD}(a \in \text{FLOAT1\_Type}) \hat{=}$

$\exists n, z \cdot n \in \mathbb{N} \wedge z \in \mathbb{Z} \wedge 10 \text{ pow } n = s(a) \times z$

$\text{div\_WD}(a \in \text{FLOAT\_Type}, b \in \text{FLOAT1\_Type}) \hat{=}$

$\exists n, z \cdot n \in \mathbb{N} \wedge z \in \mathbb{Z} \wedge s(a) \times (10 \text{ pow } n) = s(b) \times z$

**AXIOMATIC DEFINITIONS**

**operators**

$\text{inv}(x \in \text{FLOAT\_Type}) : \text{FLOAT1\_Type}$

**wd condition** :  $\text{inv\_WD}(x)$

**axioms**

**axm1**:  $\forall x, y \cdot (\dots \Rightarrow ((x \text{ mult } y) = F1 \Leftrightarrow \text{inv}(x) = y))$

**axm2**:  $\forall x, y \cdot (\dots \Rightarrow ((x \text{ mult } y) \text{ eq } F1 \Leftrightarrow \text{inv}(x) \text{ eq } y))$

...

**END**

# THE CASE OF *inv* AND *div* OPERATORS

**THEORY** thy\_floating\_point\_numbers

...

**OPERATORS**

...

$\text{inv\_WD}(a \in \text{FLOAT1\_Type}) \hat{=}$

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

$\text{div\_WD}(a \in \text{FLOAT\_Type}, b \in \text{FLOAT1\_Type}) \hat{=}$

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

**AXIOMATIC DEFINITIONS**

...

**operators**

$\text{div}(x \in \text{FLOAT\_Type}, y \in \text{FLOAT\_Type}) : \text{FLOAT\_Type}$  **INFIX**

**wd condition** :  $\text{div\_WD}(x)$

**axioms**

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

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

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

...

**END**

# THE FLOATING-POINT NUMBERS THEORY

**THEORY** thy\_floating\_point\_numbers

...

**THEOREMS**

@thm1:  $\forall x, y \cdot (\dots \Rightarrow x \text{ eq } y \Leftrightarrow y \text{ eq } x)$

@thm2:  $\forall x \cdot (\dots \Rightarrow x \text{ geq } x \wedge x \text{ leq } x)$

@thm3:  $\forall x, y \cdot (\dots x \text{ leq } y \wedge y \text{ leq } x \Rightarrow x \text{ eq } y)$

@thm4:  $\forall x, y \cdot (\dots \Rightarrow x \text{ leq } y \vee y \text{ leq } x)$

@thm5:  $\forall x, y, z \cdot (\dots x \text{ leq } y \wedge y \text{ leq } z \Rightarrow x \text{ leq } z)$

@thm6:  $\forall x, y, z \cdot (\dots x \text{ leq } y \Rightarrow (x \text{ plus } z) \text{ leq } (y \text{ plus } z))$

@thm7:  $\forall x, y, z \cdot (\dots x \text{ leq } y \Rightarrow (x \text{ mult } z) \text{ leq } (y \text{ mult } z))$

@thm8:  $\forall x \cdot (\dots \Rightarrow x \text{ plus } F0 \text{ eq } x)$

@thm9:  $\forall x, y \cdot (\dots \Rightarrow x \text{ plus } y = y \text{ plus } x)$

@thm10:  $\forall x, y \cdot (\dots \Rightarrow x \text{ plus } \text{neg}(y) = y \text{ minus } x)$

@thm11:  $\forall x \cdot (\dots \Rightarrow x \text{ minus } F0 \text{ eq } x)$

@thm12:  $\forall x \cdot (\dots \Rightarrow x \text{ minus } x \text{ eq } F0)$

...

**END**

# THE FLOATING-POINT NUMBERS THEORY

**THEORY** thy\_floating\_point\_numbers

...

**THEOREMS**

...

@thm13:  $\forall x \cdot (\dots \Rightarrow x \text{ mult } F0 \text{ eq } F0)$

@thm14:  $\forall x \cdot (\dots \Rightarrow x \text{ mult } F1 = x)$

@thm15:  $\forall x, y \cdot (\dots \Rightarrow x \text{ mult } y = y \text{ mult } x)$

@thm16:  $\forall x \cdot (\dots \Rightarrow \text{inv}(x) = F1 \text{ div } x)$

@thm17:  $\forall x \cdot (\dots \Rightarrow x \text{ div } F1 = x)$

@thm18:  $\forall x \cdot (\dots \Rightarrow x \text{ div } x = F1)$

@thm19:  $\forall x \cdot (\dots \Rightarrow x \text{ mult inv}(x) = F1)$

...

**END**

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

# NATURAL VARIABLES

**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**

- All **NATURAL** variables are typed by **PFLOAT\_Type** set containing positive floating-point numbers.

# REVISITING OUR EXAMPLE I

**MACHINE** mch\_integer\_version

...

**INVARIANTS**

@inv1: distance\_travelled  $\in \mathbb{N}$

@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 = distance\_travelled  $\div$  measured\_time

@inv7: distance\_travelled  $> 0 \Rightarrow$  speed  $> 0$

...

**END**

# REVISITING OUR EXAMPLE I

**MACHINE** mch\_floating\_point\_version

...

**INVARIANTS**

@inv1: distance\_travelled  $\in$  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 distance\_travelled div measured\_time

@inv8: distance\_travelled gt F0  $\Rightarrow$  speed gt F0

...

**END**

# REVISITING OUR EXAMPLE I

**MACHINE** mch\_floating\_point\_version

...

**INVARIANTS**

@inv1: distance\_travelled  $\in$  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

@inv6: div\_WD(distance\_travelled, measured\_time)

@inv7: speed eq distance\_travelled div measured\_time

@inv8: distance\_travelled 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: distance\_travelled :=  $p - \text{starting\_position}$

@act2: measured\_time :=  $t - \text{starting\_time}$

@act3: 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  $\in$  PFLOAT_Type  $\wedge$  p gt starting_position
    @grd2: t  $\in$  PFLOAT_Type  $\wedge$  t gt starting_time
  then
    @act1: distance_travelled := 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:  $\text{distance\_travelled} := p \text{ minus starting\_position}$ 
```




```
    @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

- ✓ **M** mch\_floating\_point\_speed
  - >  Variables
  - >  Invariants
  - >  Events
  - ✓ **P** 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

- 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.**

# OUTLINE

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

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

# CONCLUSION

- Extending the **Event-B type-checking system** by an approach using the **theory plugin**.
- Development of a **floating point number theory** formalizing 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).
- Developing a **more general theory** formalizing 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](#)

**MEDI 2023**

12<sup>th</sup> International Conference on Model and Data Engineering  
2-4 of November 2023, Sousse, Tunisia