Laboratoire
Méthodes
Formelles

# A FLOATING-POINT NUMBERS THEORY FOR EVENT-B

The LMF Lab Seminar
Domaine Saint Paul, Saint-Rémy-lès-Chevreuse - June 13-14, 2024

**Idir AIT SADOUNE**
idiraitsadoune@lmf.cnrs.fr

# OUTLINE



Back to the outline - Back to the begin

# OUTLINE



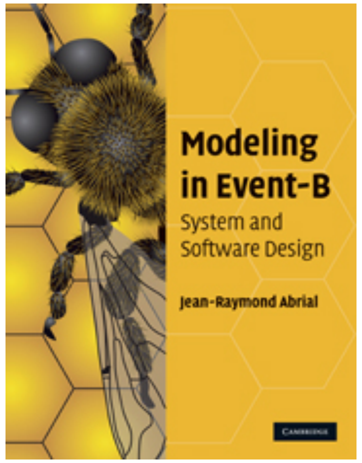
Back to the outline - Back to the begin

# 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 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 **Rodin** platform (an Eclipse-based IDE) is intended to support the construction and verification of **Event-B models**.
- 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 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 ≙
  any x
  where
    G(s, c, v, x)
  then
    BA(s, c, v, x, v')
  end
```

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

# 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.


- Michael J. Butler and Issam Maamria.
  *Practical theory extension in Event-B.* Theories of Programming and Formal Methods. 2013.
- Thai Son Hoang, Laurent Voisin, Asieh Salehi, Michael J. Butler, Toby Wilkinson, and Nicolas Beauger.
  *Theory plug-in for Rodin 3.x.* CoRR, abs/1701.08625, 2017.

# THE EVENT-B METHOD

```
THEORY thy1
IMPORT thy2

DATATYPES
  DT1, ..., DTn
OPERATORS
  OP11, ..., OP1n
AXIOMATIC DEFINITIONS
  operators
    OP21, ..., OP2n
  axioms
    A
THEOREMS
  T
PROOF RULES
  PR
END
```

**THEORY** $thy_1$
**IMPORT** $thy_2$

**DATATYPES**
$DT_1, \ldots, DT_n$
**OPERATORS**
$OP_{11}, \ldots, OP_{1n}$
**AXIOMATIC DEFINITIONS**
**operators**
$OP_{21}, \ldots, OP_{2n}$
**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**

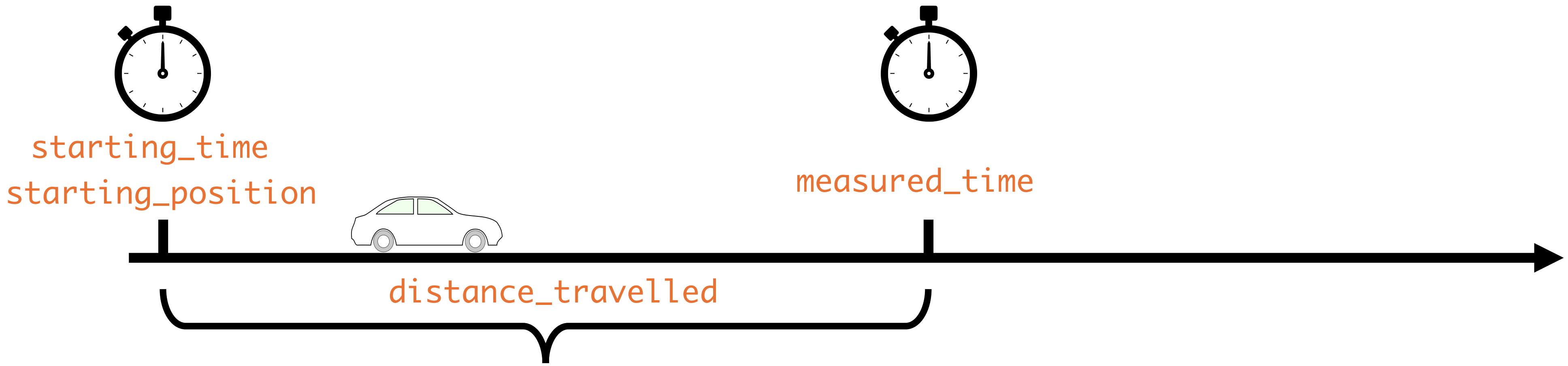
# OUTLINE



Back to the outline - Back to the begin

# A SIMPLE EXAMPLE

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



# A SIMPLE EXAMPLE

- Analysing **two functional properties**:
  - **PROP-1** : **the speed of the moving object** is equal to the $distance\_travelled$ divided by the $measured\_time$ ($v = d/t$).
  - **PROP-2** : when the $distance\_travelled$ 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** $\longrightarrow$ showing some **modelling and verification problems** :

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

# THE EVENT-B MODEL

- Analysing **two functional properties**:
  - **PROP-1** : **the speed of the moving object** is equal to the $distance\_travelled$ divided by the $measured\_time$ ($v = d/t$).
  - **PROP-2** : when the $distance\_travelled$ is strictly positive, the $speed$ of the moving object must also be strictly positive.
    - **the object moves** when its $speed$ is different from zero.

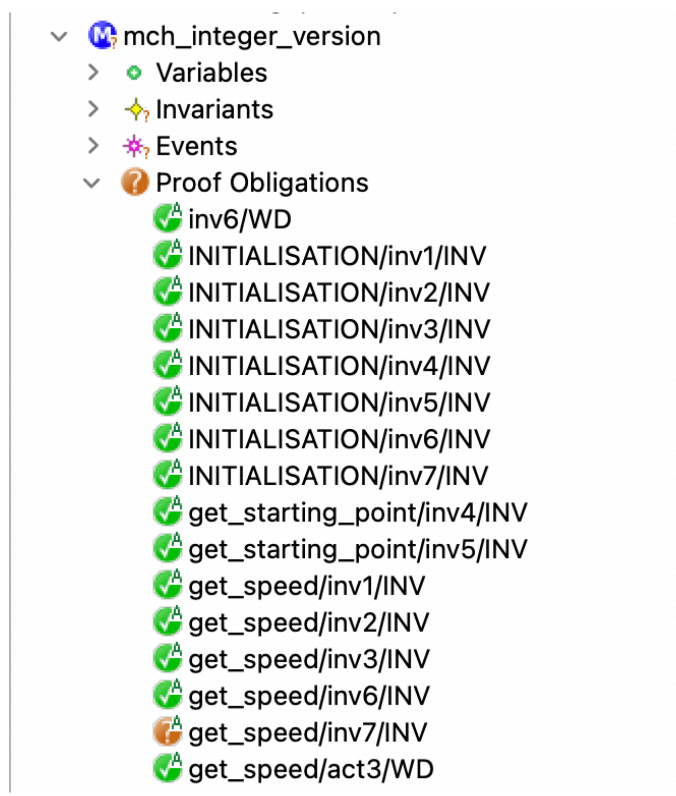
```
MACHINE mch_integer_version
...
INVARIANTS
  @inv1: distance_travelled ∈ ℕ
  @inv2: measured_time ∈ ℕ1
  @inv3: speed ∈ ℕ
  @inv4: starting_position ∈ ℕ
  @inv5: starting_time ∈ ℕ
  @inv6: speed = distance_travelled ÷ measured_time // PROP-1
  @inv7: distance_travelled > 0 ⇒ speed > 0 // PROP-2
```

# THE EVENT-B MODEL

```
MACHINE mch_integer_version
...
EVENTS
  ...
  get_speed ≙
    any p t
    where
      @grd1: p ∈ ℕ1 ∧ p > starting_position
      @grd2: t ∈ ℕ1 ∧ t > starting_time
    then
      @act1: distance_travelled := p − starting_position
      @act2: measured_time := t − starting_time
      @act3: speed := (p − starting_position) ÷ (t − starting_time)
    end
END
```

# GENERATED AND PROVEN POS

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



# CONCLUSION

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

# OUTLINE



Back to the outline - Back to the begin

# FLOATING-POINT NUMBERS

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

**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 theory formalises **the power operator**.
   ✘ ^ operator is **not implemented** in the provers besides ^0 and ^1.

2. the second theory 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 ∈ ℤ, n ∈ ℕ) : ℤ INFIX // x pow n = xⁿ
    wd condition : ¬ (x = 0 ∧ n = 0) // 0⁰ is not defined

  axioms
    @axm1: ∀ n · n ∈ ℕ1 ⇒ 0 pow n = 0
    @axm2: ∀ x · x ∈ ℤ ∧ x ≠ 0 ⇒ x pow 0 = 1
    @axm3: ∀ x,n · x ∈ ℤ ∧ x ≠ 0 ∧ n ∈ ℕ1 ⇒ x pow n = x × (x pow (n − 1))
    ...

THEOREMS
  @thm1: ∀ x,n,m · ... ⇒ x pow (n + m) = (x pow n) × (x pow m)
  @thm2: ∀ x,n,m · ... ⇒ (x pow n) pow m = x pow (n × m)
  @thm3: ∀ x,y,n · ... ⇒ (x × y) pow n = (x pow n) × (y pow n)
  ...

END
```

# SOME REMARKS

- By using this theory, it **becomes possible to prove**, for example, that 5 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** developments.


- Julien Cervelle and Frédéric Gervais.
  *Introducing Inductive Construction in B with the Theory Plugin*. ABZ, 2023.

# THE FLOATING-POINT NUMBERS THEORY

```
THEORY thy_floating_point_numbers

DATATYPES
  FLOAT_Type ≙ NEW_FLOAT(s ∈ ℤ, e ∈ ℤ) // x = s(x) × 10^e(x)

OPERATORS
  F0 ≙ NEW_FLOAT(0,0)  // 0 = 0 × 10^0
  F1 ≙ NEW_FLOAT(1,0)  // 1 = 1 × 10^0
  FLOAT1_Type ≙ { x · x ∈ FLOAT_Type ∧ s(x) ≠ 0 | x }
  FLOAT(x ∈ ℤ) ≙ NEW_FLOAT(x,0)  // x = x × 10^0

  l_shift(x ∈ FLOAT_Type, offset ∈ ℕ) ≙ NEW_FLOAT(s(x) × (10 pow offset), e(x) − offset)

  eq(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙
    s(l_shift(x, e(x) − min({e(x),e(y)}))) = s(l_shift(y, e(y) − min({e(x),e(y)})))
  gt(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  geq(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  lt(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  leq(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  ...

END
```

# THE FLOATING-POINT NUMBERS THEORY

```
THEORY thy_floating_point_numbers
...
OPERATORS
  ...
  plus(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙
    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 ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  neg(x ∈ FLOAT_Type) ≙ ...

  mult(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ NEW_FLOAT(s(x) × s(y) , e(x) + e(y))
  f_pow(x ∈ FLOAT_Type, n ∈ ℕ) INFIX ≙ NEW_FLOAT(s(x) pow n, e(x) × n)
  floor(x ∈ FLOAT_Type) ≙ ...
  ceiling(x ∈ FLOAT_Type) ≙ ...
  integer(x ∈ FLOAT_Type) ≙ ...
  frac(x ∈ FLOAT_Type) ≙ ...
  ...

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 $100 = 4 \times 25$)
    ✘ $inv(3) = 1/3 = 0.3333\ldots$ ($z$ **does not exist**)
  - To calculate $x \ \ div \ \ y$, we must find a $z$, with $10^n \times s(x) = z \times s(y)$.
    ✔ $2 \, div \, 5 = 2/5 = 0.4 = 4\times 10^{-1}$ ($z = 4$ because $10 \times 2 = 4 \times 5$)
    ✘ $2 \, div \, 3 = 2/3 = 0.6666\ldots.$ ($z$ **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
    div(x ∈ FLOAT_Type, y ∈ FLOAT_Type) : FLOAT_Type INFIX
    wd condition : div_WD(x)

  axioms
    axm1: ∀ x,y,z · (... ⇒ ((y mult z) = x ⇔ (x div y) = z))
    axm2: ∀ x,y,z · (... ⇒ ((y mult z) eq x ⇔ (x div y) eq z))
    axm3: ∀ x,y · (... ⇒ x mult inv(y) = x div y)
    ...

END
```

# THE FLOATING-POINT NUMBERS THEORY

```
THEORY thy_floating_point_numbers
...
THEOREMS
  @thm1: ∀ x,y · (... ⇒ x eq y ⇔ y eq x)
  @thm2: ∀ x · (... ⇒ x geq x ∧ x leq x)
  @thm3: ∀ x,y · (... x leq y ∧ y leq x ⇒ x eq y)
  @thm4: ∀ x,y · (... ⇒ x leq y ∨ y leq x)
  @thm5: ∀ x,y,z · (... x leq y ∧ y leq z ⇒ x leq z)
  @thm6: ∀ x,y,z · (... x leq y ⇒ (x plus z) leq (y plus z))
  @thm7: ∀ x,y,z · (... x leq y ⇒ (x mult z) leq (y mult z))
  @thm8: ∀ x · (... ⇒ x plus F0 eq x)
  @thm9: ∀ x,y · (... ⇒ x plus y = y plus x)
  @thm10: ∀ x,y · (... ⇒ x plus neg(y) = y minus x)
  @thm11: ∀ x · (... ⇒ x minus F0 eq x)
  @thm12: ∀ x · (... ⇒ x minus x eq F0)
  @thm13: ∀ x · (... ⇒ x mult F0 eq F0)
  @thm14: ∀ x · (... ⇒ x mult F1 = x)
  @thm15: ∀ x,y · (... ⇒ x mult y = y mult x)
  @thm16: ∀ x · (... ⇒ inv(x) = F1 div x)
  @thm17: ∀ x · (... ⇒ x div F1 = x)
  @thm18: ∀ x · (... ⇒ x div x = F1)
  @thm19: ∀ x · (... ⇒ x 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



Back to the outline - Back to the begin

# NATURAL VARIABLES

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

```
THEORY thy_floating_point_numbers
  ...
  PFLOAT_Type = { x · x ∈ FLOAT_Type ∧ s(x) ≥ 0 | x }
  PFLOAT1_Type = { x · x ∈ FLOAT_Type ∧ s(x) > 0 | x }
  ...
END
```

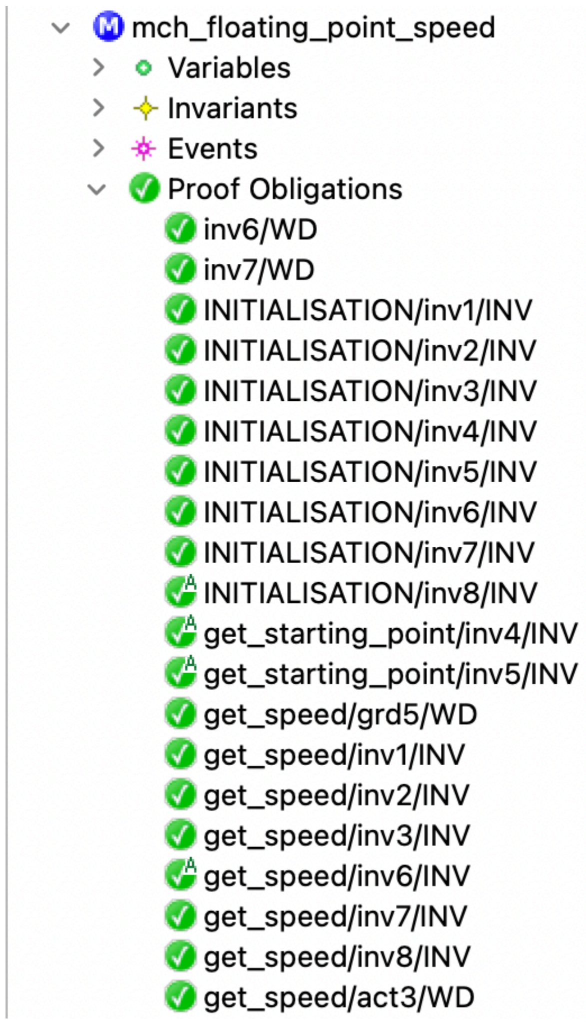
# REVISITING OUR EXAMPLE I

```
MACHINE mch_floating_point_version
...
INVARIANTS
  @inv1: distance_travelled ∈ PFLOAT_Type
  @inv2: measured_time ∈ PFLOAT1_Type
  @inv3: speed ∈ PFLOAT_Type
  @inv4: starting_position ∈ PFLOAT_Type
  @inv5: starting_time ∈ PFLOAT_Type
  @inv6: div_WD(distance_travelled, measured_time)
  @inv7: speed eq distance_travelled div measured_time
  @inv8: distance_travelled gt F0 ⇒ speed gt F0
...
END
```

# REVISITING OUR EXAMPLE II

```
MACHINE mch_floating_point_version
...
EVENTS
  ...
  get_speed ≙
    any p t
    where
      @grd1: p ∈ PFLOAT_Type ∧ p gt starting_position
      @grd2: t ∈ PFLOAT_Type ∧ t gt starting_time
      @grd3: div_WD(p minus starting_position, t minus 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
```

# 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.**

# OUTLINE



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