A GENERIC EVENT-B THEORY FOR THE FORMALISATION OF THE INTERNATIONAL SYSTEM OF UNITS

This work was supported by a grant from the French national research agency ANR ANR-19-CE25-0010 (EBRP Project).

18^{th} International Conference on Verification and Evaluation of Computer and Communication Systems
Centre d'intégration Nano-INNOV - CEA-LIST, Palaiseau, France 5-6 November 2025

Idir AIT SADOUNE
idiraitsadoune@lmf.cnrs.fr

OUTLINE

Back to the outline - Back to the begin

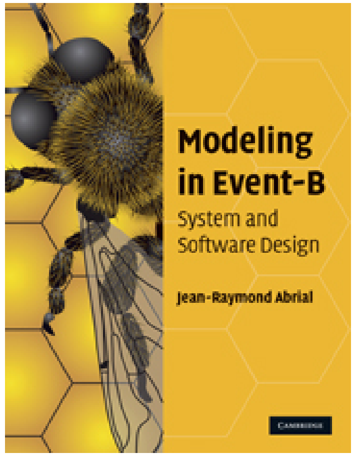
OUTLINE

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.

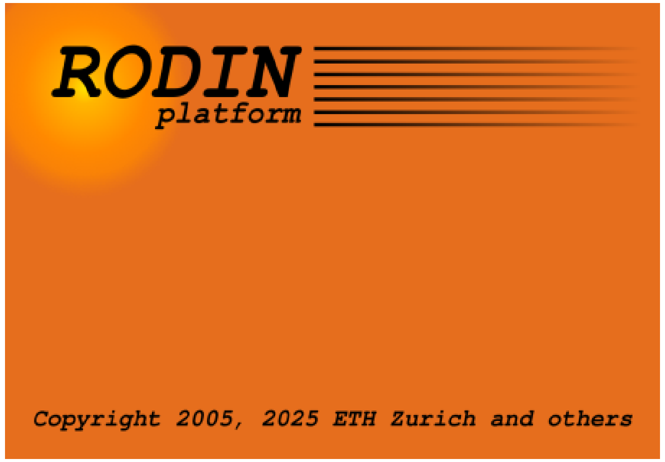
- The **Event-B method** is a formal method based on first-order logic and set theory.



- 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

MODELS AND PROOF OBLIGATIONS

```
CONTEXT ctx1
EXTENDS ctx2

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

```
MACHINE mch1
REFINES mch2
SEES ctxi

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
```

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

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

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

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

$$\ldots$$

THE EVENT-B METHOD

STATIC TYPE CHECKING

- **Event-B** supports static type checking using tools such as **Rodin** or **AtelierB**.
- These tools generate proof obligations (**POs**) to check the correct use of arithmetic operations (**Well-Defined** proof obligations - **WD POs**).
- **WD POs** ensure that expressions (axioms, theorems, invariants, guards, actions, etc.) are mathematically well-defined.
- **Example** $\longrightarrow$ for the expression $x \div y$, a **WD PO** ensures that $y \neq 0$.

THE EVENT-B METHOD

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.

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

- 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

THE THEORY PLUGIN

```
THEORY thy₁
```

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

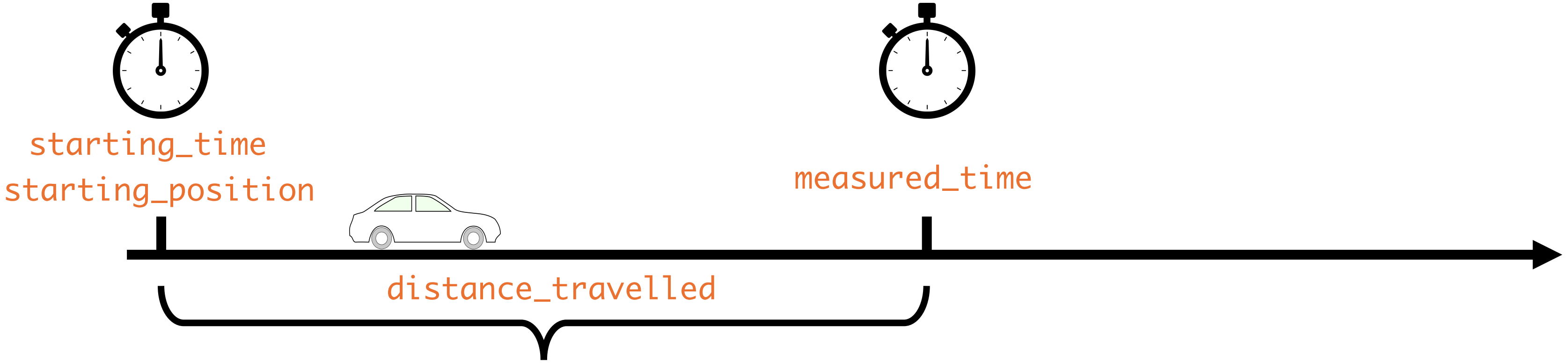
VARIABLES v
INVARIANTS
$I(s, c, v)$
THEOREMS
$T(s, c, v)$
EVENTS
$[events\_list]$
END

OUTLINE

Back to the outline - Back to the begin

A SIMPLE EXAMPLE

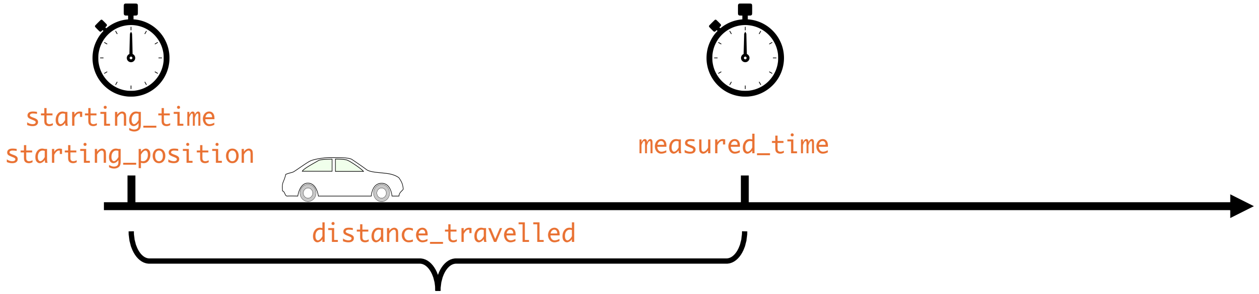
System that continuously calculates **a moving object's speed**



Analysing **two functional properties**:

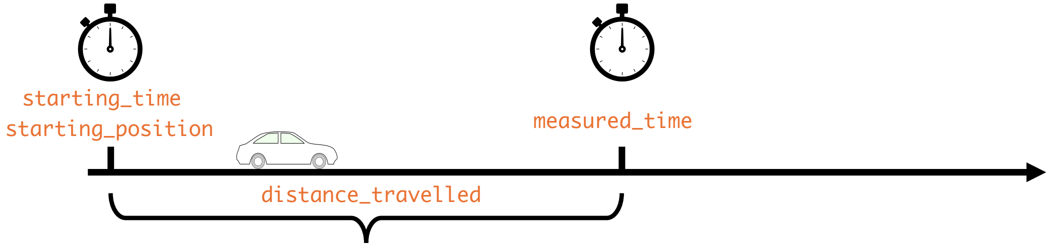
- **PROP-1** : **the velocity of the moving object** is equal to the $distance\_travelled$ divided by the $measured\_time$ ($v = d/t$).
- **PROP-2** : when the $distance\_travelled$ is strictly positive, the $speed$ of the moving object must also be strictly positive.
  - **the object moves** when its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

THE EVENT-B MODEL




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

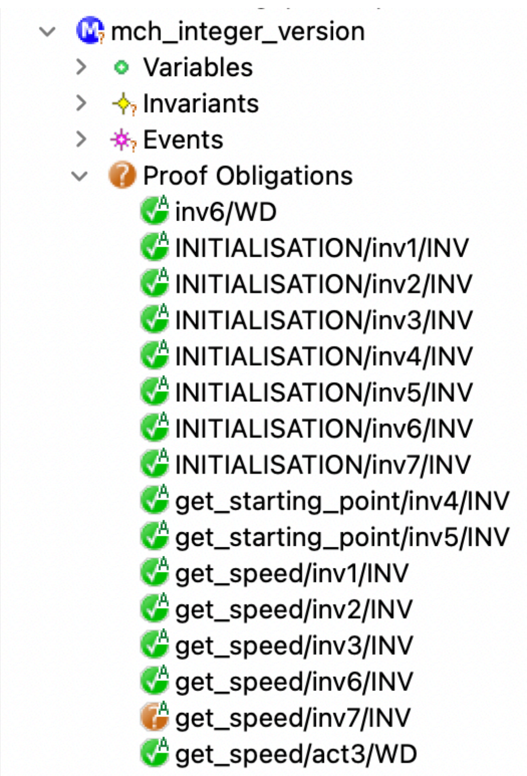
THE EVENT-B MODEL




```
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.
- **PROP 2** $\longrightarrow$ $distance\_travelled \neq 0$ when $speed \neq 0$.
  - the value of $distance\_travelled$ can be $<$ the value of $measured\_time$.
  - the value of $speed$ can be $= 0$ ($distance\_travelled \div measured\_time$) while $distance\_travelled \neq 0$



- **No possibility** to check the consistency of formulas annotated with measurement units.
  - **Example:** is the unit of speed (km/h) the same with the unit of the expression distance_travelled ÷ measured_time (km ÷ s) ?

CHALLENGES IN MODELLING CPS SYSTEMS

- More generally, **Cyber-Physical Systems** (**CPS**) models often require variables/expressions, formalising **measurements**/**physics and mechanics laws**.

- **Event-B** does not support measurements unit annotations for such variables and using **integer** variables is not sufficient to handle **small values** $(0 < v < 1)$.
  - converting from the smallest point of view to the most significant ones
  - from `Milli` to `Kilo`, for example

- Formal verification of CPS systems requires a physical measurement **model**, **e.g.** the International System of Units (**SI**).

- Using **explicit units** improves the **CPS** validation process by ensuring **unit compatibility** in arithmetic expressions.

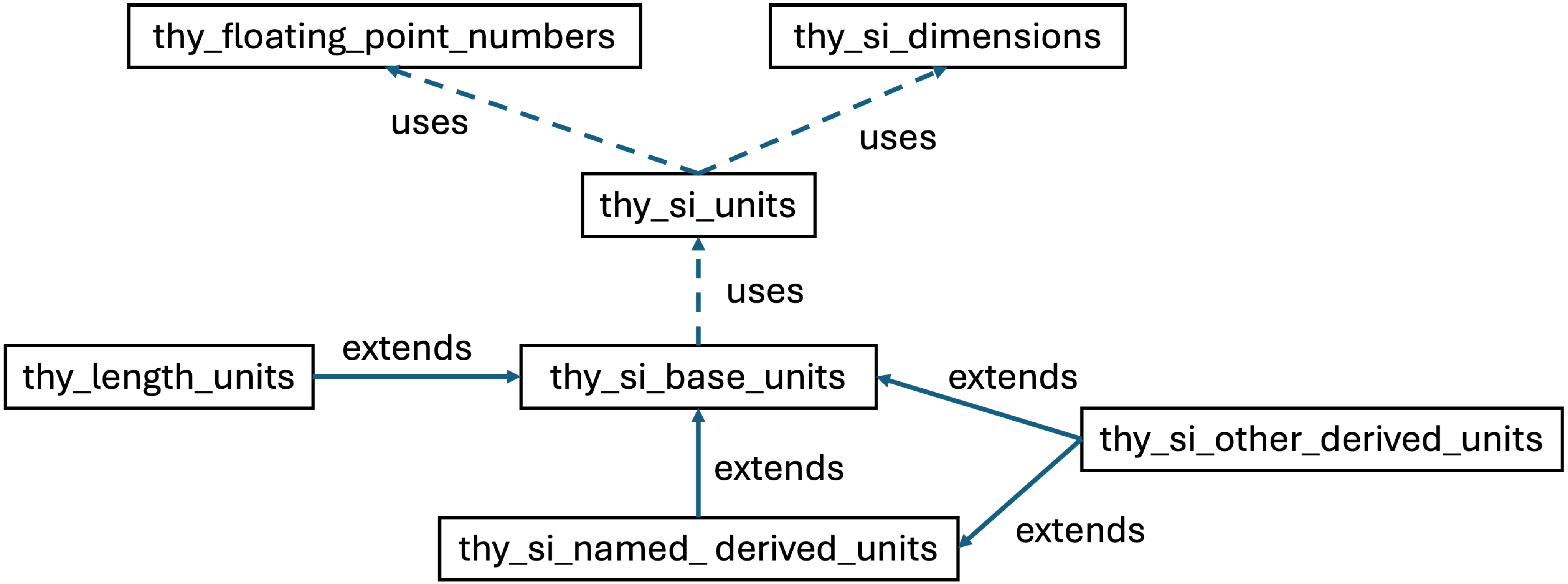
THE OBJECTIVES

- New syntaxe to formally annotate Event-B variables with measurement units.
- New generaic arithmetic operators for the annotated variables.
- New **Well-Defined Proof Obligations** (**WD POs**) to ensure unit consistency.
- Automatic checking of correct unit usage in arithmetic expressions.
- **Example:** $d = v/2\ a$
  $\rightarrow$ must ensure that the unit of d matches that of $v/2\ a$.

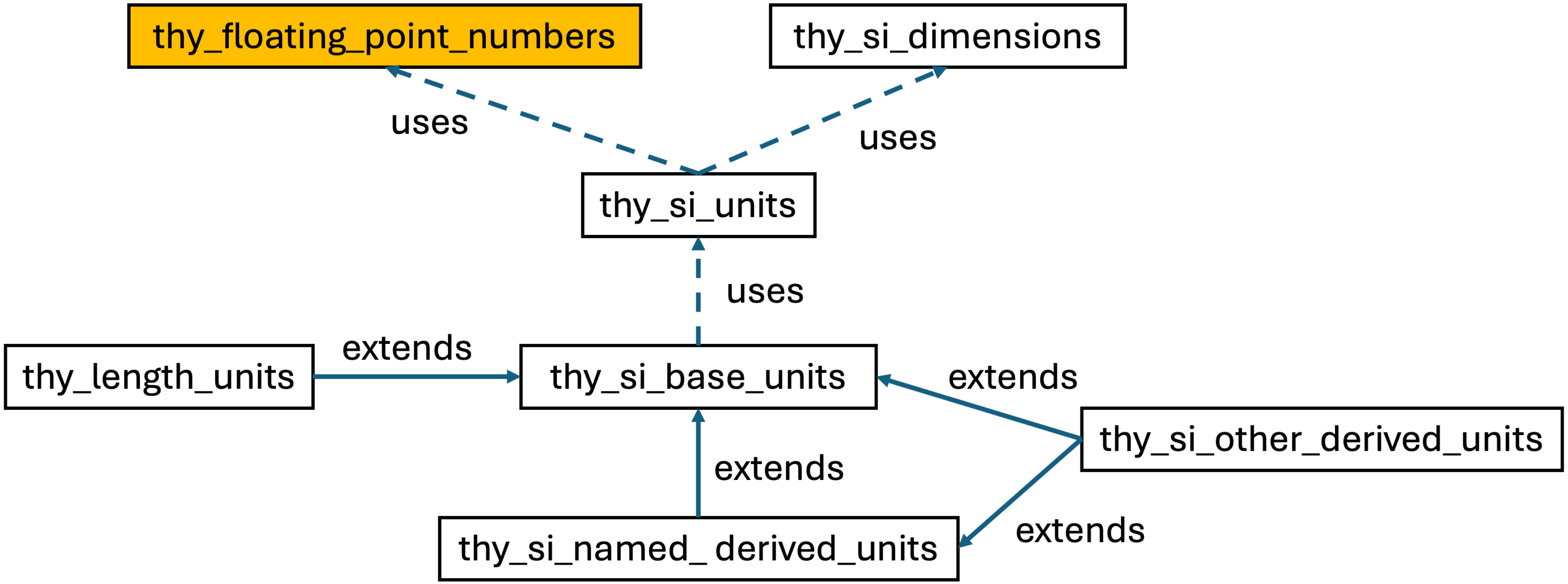
OUTLINE

Back to the outline - Back to the begin

PROPOSED APPROACH



PROPOSED APPROACH



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**.

VECoS'25

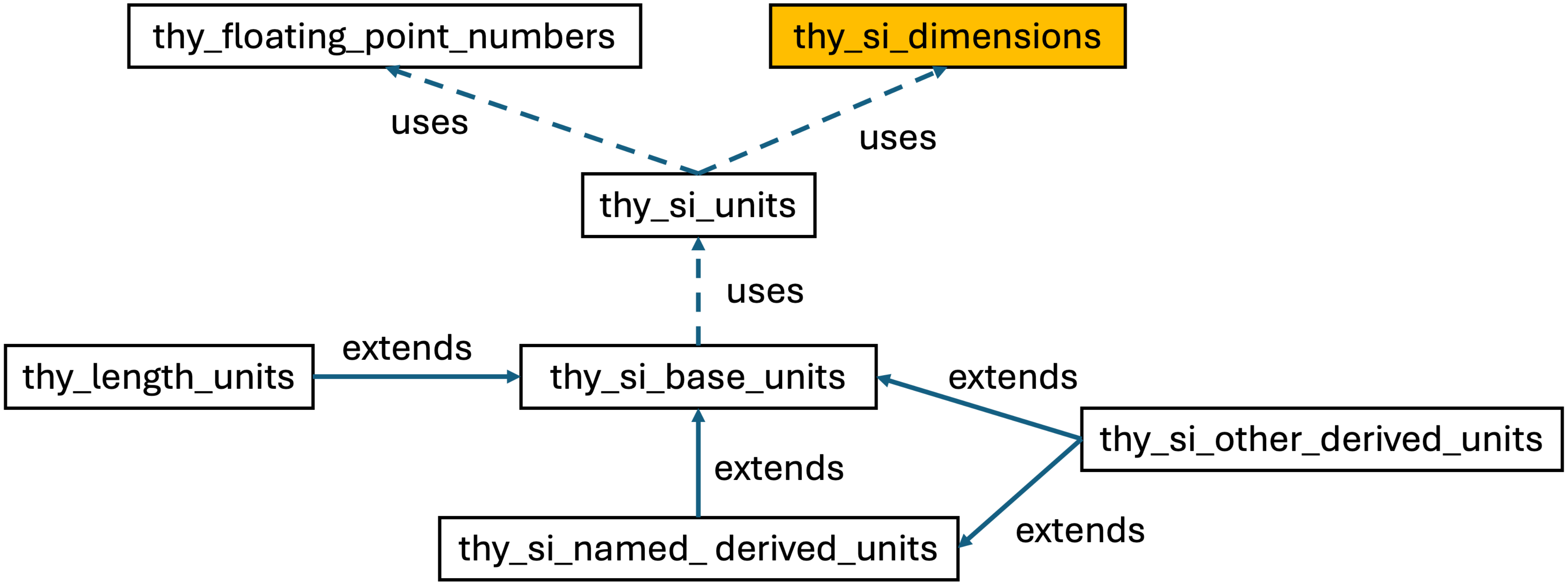
- Idir AIT SADOUNE, *A Floating-Point Numbers Theory for Event-B.* 12th International Conference on Model and Data Engineering, MEDI 2023.

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
  F1 ≙ NEW_FLOAT(1,0) // 10^0 = 1
  ...
  MILLI ≙ NEW_FLOAT(1,-3) // 10^-3
  CENTI ≙ NEW_FLOAT(1,-2) // 10^-2
  DECI ≙ NEW_FLOAT(1,-1) // 10^-1
  DECA ≙ NEW_FLOAT(1,1) // 10^1
  HECTO ≙ NEW_FLOAT(1,2) // 10^2
  KILO ≙ NEW_FLOAT(1,3) // 10^3
  ...
  eq(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  gt(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  ...
  plus(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  mult(x ∈ FLOAT_Type, y ∈ FLOAT_Type) INFIX ≙ ...
  ...
END
```

PROPOSED APPROACH



VECoS'25

DIMENSIONS FORMALISATION

- **SI System** $\rightarrow$ a coherent system of measurement based on seven base quantities.
- **Base Quantities**:
  Time (T), Length (L), Mass (M), Electric current (I), Thermodynamic temperature (Θ), Amount of substance (N), Luminous intensity (J).
- Each **base quantity** corresponds to **a base dimension**.
- Physical quantities are organized in a system of dimensions.
- **The dimension** of any quantity Q is expressed as:

$$dim\ Q = T^{\alpha} L^{\beta} M^{\gamma} I^{\delta} \Theta^{\varepsilon} N^{\zeta} J^{\eta}$$

➠ the exponents $\alpha, \beta, \gamma, \delta, \varepsilon, \zeta$ and η are **the dimensional exponents** (can be positive, negative, or zero).

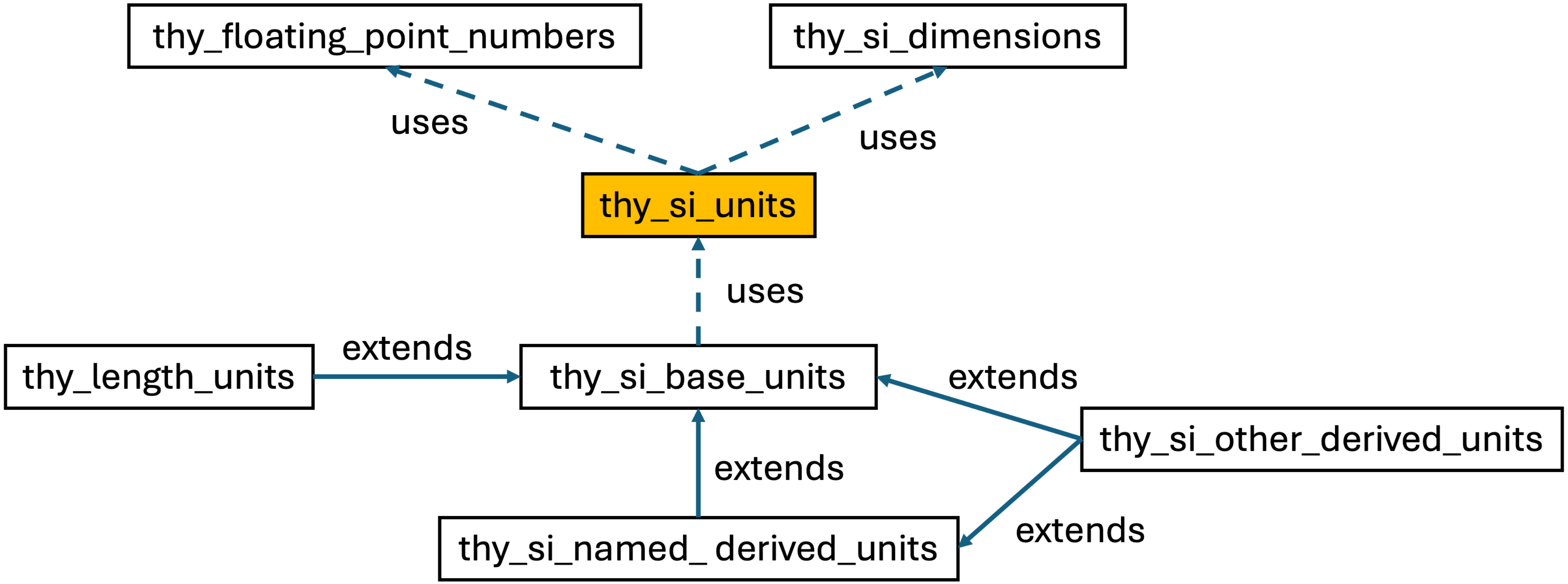
DIMENSIONS FORMALISATION

```
DATATYPES
  SI_DIMENSION_Type ≙ SI_DIMENSION(
    exp_d1 ∈ ℤ, // length dimension
    exp_d2 ∈ ℤ, // mass dimension
    exp_d3 ∈ ℤ, // time dimension
    exp_d4 ∈ ℤ, // electric current dimension
    exp_d5 ∈ ℤ, // thermodynamic temperature dimension
    exp_d6 ∈ ℤ, // amount of substance dimension
    exp_d7 ∈ ℤ) // luminous intensity dimension
OPERATORS
  L_DIM (exp_d ∈ ℤ) ≙  SI_DIMENSION(exp_d,0,0,0,0,0,0) // length quantity
  M_DIM (exp_d ∈ ℤ) ≙ SI_DIMENSION(0,exp_d,0,0,0,0,0) // mass quantity
  T_DIM (exp_d ∈ ℤ) ≙ SI_DIMENSION(0,0,exp_d,0,0,0,0) // time quantity
  ...
  DIM_MULT(dim1 ∈ SI_DIMENSION_Type, dim2 ∈ SI_DIMENSION_Type) ≙
    SI_DIMENSION(..., exp_di(dim1)+exp_di(dim2)), ...)

  DIM_DIV(dim1 ∈ SI_DIMENSION_Type, dim2 ∈ SI_DIMENSION_Type) ≙
    SI_DIMENSION(..., exp_di(dim1)−exp_di(dim2)), ...)

  HAVE_SAME_EXP_DIMENSIONS(dim1 ∈ SI_DIMENSION_Type, dim2 ∈ SI_DIMENSION_Type) ≙
    dim1=dim2
  ...
```

PROPOSED APPROACH



UNIT OF A QUANTITY

- A **unit** is formalised using a product of a multiplier with dimension shifted by an offset:

$$unit = multiplier \times dimension + offset$$

- Multiplier
  - represents prefixes applied to base units.
  - **examples**: milli, centi, deci, deca, kilo, etc.
  - used to express multiples or submultiples of a **base unit** (e.g., $1km = 1000m$).
- Offset
  - defines a shift relative to a **base unit**.
  - **example**: the degree Celsius is offset by 273.15 from the Kelvin (K) unit.
  - useful for units that are not directly proportional to their base unit.

VECoS'25

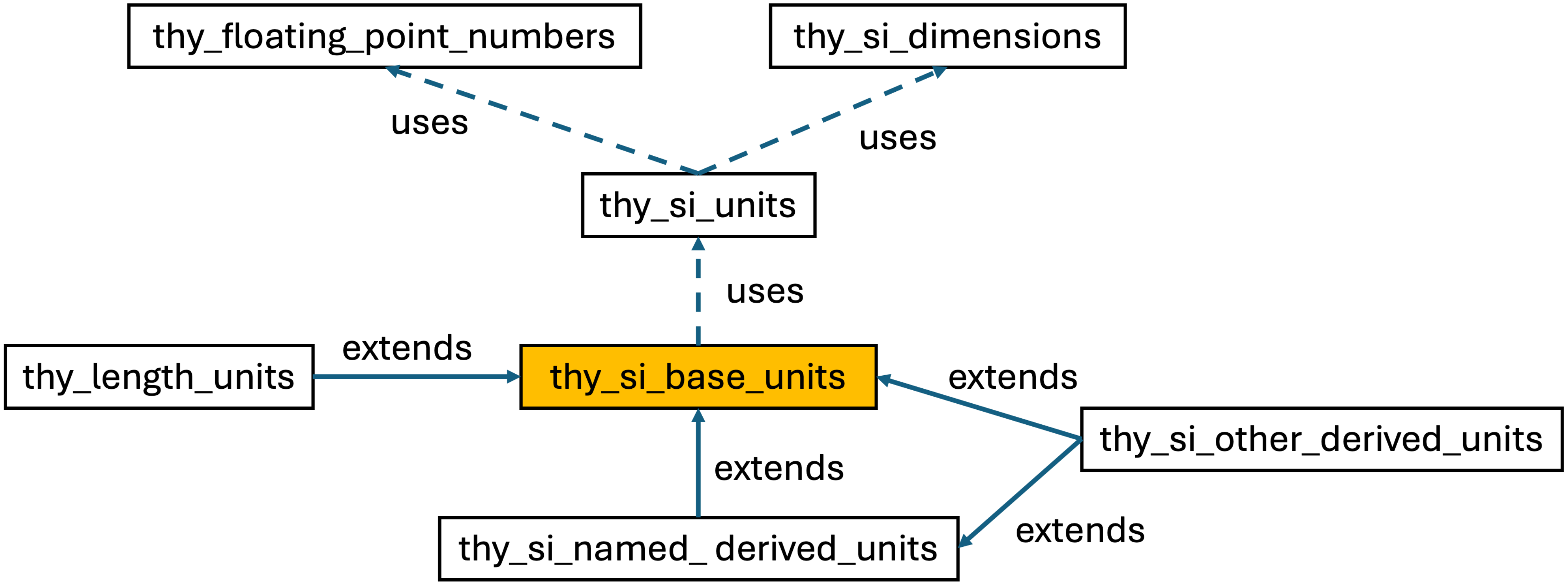
UNIT OF A QUANTITY

```
DATATYPES
  SI_UNIT_Type ≙ SI_UNIT(multiplier ∈ FLOAT_Type, dimension ∈ SI_DIMENSION_Type, offset ∈ FLOAT_Type)
  MEASURE_Type ≙ MEASURE(value ∈ FLOAT_Type, unit ∈ SI_UNIT_Type)
OPERATORS
  UNIT_MULT(u1 ∈ SI_UNIT_Type, u2 ∈ SI_UNIT_Type) ≙
    SI_UNIT(multiplier(u1) mult multiplier(u2), DIM_MULT(dimension(u1), dimension(u2)), F0)
  UNIT_DIV(u1 ∈ SI_UNIT_Type, u2 ∈ SI_UNIT_Type) ≙
    SI_UNIT(multiplier(u1) div multiplier(u2), DIM_DIV(dimension(u1), dimension(u2)), F0)
  ...
  SI_MEASURE_Type(t ∈ SI_UNIT_Type) ≙ {x · x ∈ MEASURE_Type ∧ unit(x) = t | x}
  HAVE_THE_SAME_UNIT(m1 ∈ MEASURE_Type, m2 ∈ MEASURE_Type) ≙ unit(m1) = unit(m2)

  SI_EQ(m1 ∈ MEASURE_Type, m2 ∈ MEASURE_Type) ≙
    wd : HAVE_THE_SAME_UNIT(m1,m2)
    def : value(m1) eq value(m2)
  ...
  SI_PLUS(m1 ∈ MEASURE_Type, m2 ∈ MEASURE_Type) ≙
    wd : HAVE_THE_SAME_UNIT(m1,m2)
    def : MEASURE(value(m1) plus value(m2), unit(m1))
  ...
  SI_MULT(m1 ∈ MEASURE_Type, m2 ∈ MEASURE_Type) ≙
    MEASURE(value(m1) mult value(m2), UNIT_MULT(unit(m1), unit(m2)))
  ...
  SI_CONVERT(u ∈ SI_UNIT_Type, m ∈ MEASURE_Type) ≙
    wd : HAVE_SAME_EXP_DIMENSIONS(dimension(unit(m)),dimension(u))
    def : // v2 = (v1 − o1) × (m1 × d1)/(m2 × d2) + o2
```

VECoS'2

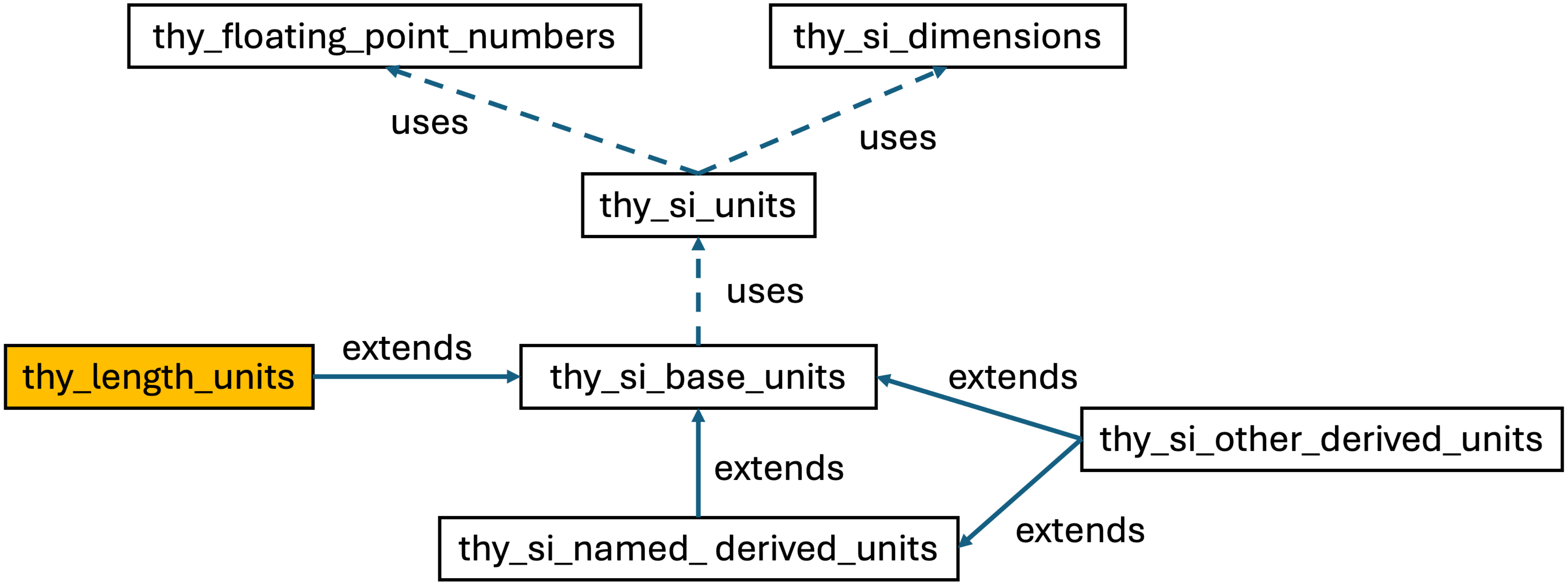
PROPOSED APPROACH



SI BASE UNITS FORMALISATION

```
OPERATORS
  METRE_UNIT ≙ SI_UNIT(F1, L_DIM(1), F0) // m
  KILO_GRAM_UNIT ≙ SI_UNIT(KILO, M_DIM(1), F0) // kg
  SECOND_UNIT ≙ SI_UNIT(F1, T_DIM(1), F0) // s
  AMPERE_UNIT ≙ SI_UNIT(F1, I_DIM(1), F0) // A
  KELVIN_UNIT ≙ SI_UNIT(F1, O_DIM(1), F0) // K
  MOLE_UNIT ≙ SI_UNIT(F1, N_DIM(1), F0) // mol
  CANDELA_UNIT ≙ SI_UNIT(F1, J_DIM(1), F0) // cd
```

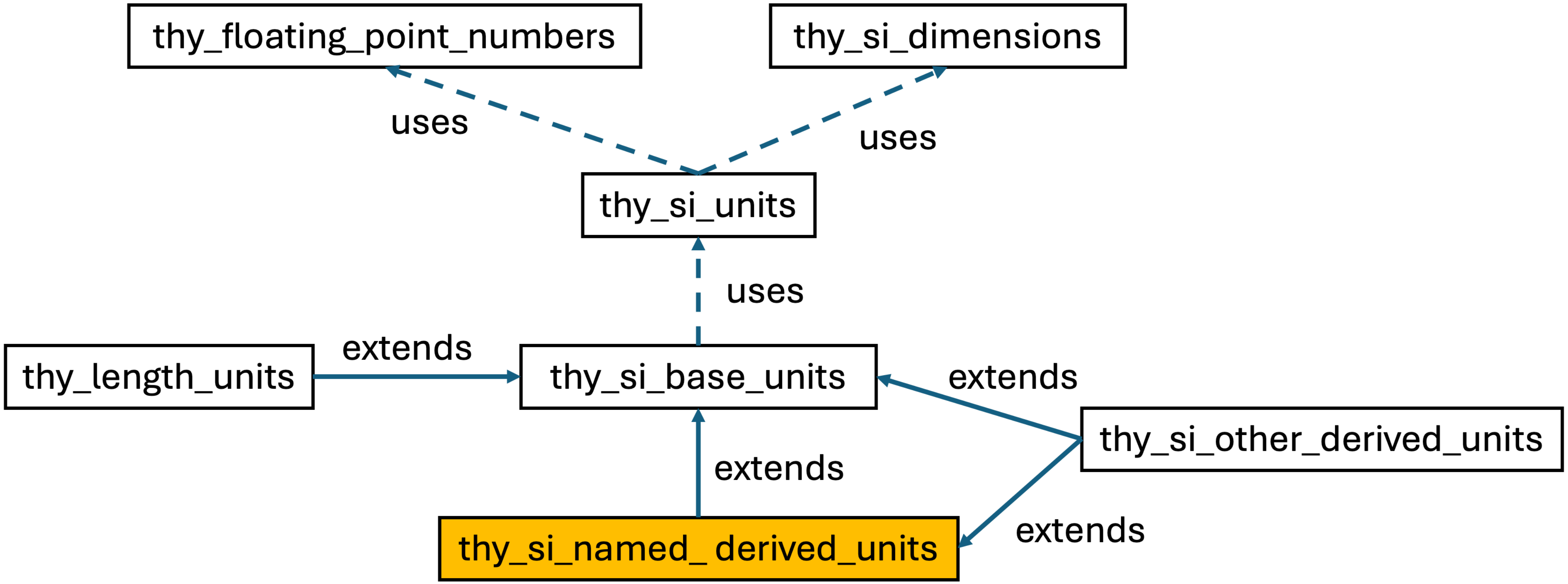
PROPOSED APPROACH



LENGTH UNITS FORMALISATION

```
OPERATORS
  MILLI_METRE_UNIT ≙ SI_UNIT(MILLI, L_DIM(1), F0) // mm
  CENTI_METRE_UNIT ≙ SI_UNIT(CENTI, L_DIM(1), F0) //cm
  DECI_METRE_UNIT ≙ SI_UNIT(DECI, L_DIM(1), F0) //dm
  DECA_METRE_UNIT ≙ SI_UNIT(DECA, L_DIM(1), F0) //dam
  HECTO_METRE_UNIT ≙ SI_UNIT(HECTO, L_DIM(1), F0) //hm
  KILO_METRE_UNIT ≙ SI_UNIT(KILO, L_DIM(1), F0) //km
  ...
```

PROPOSED APPROACH

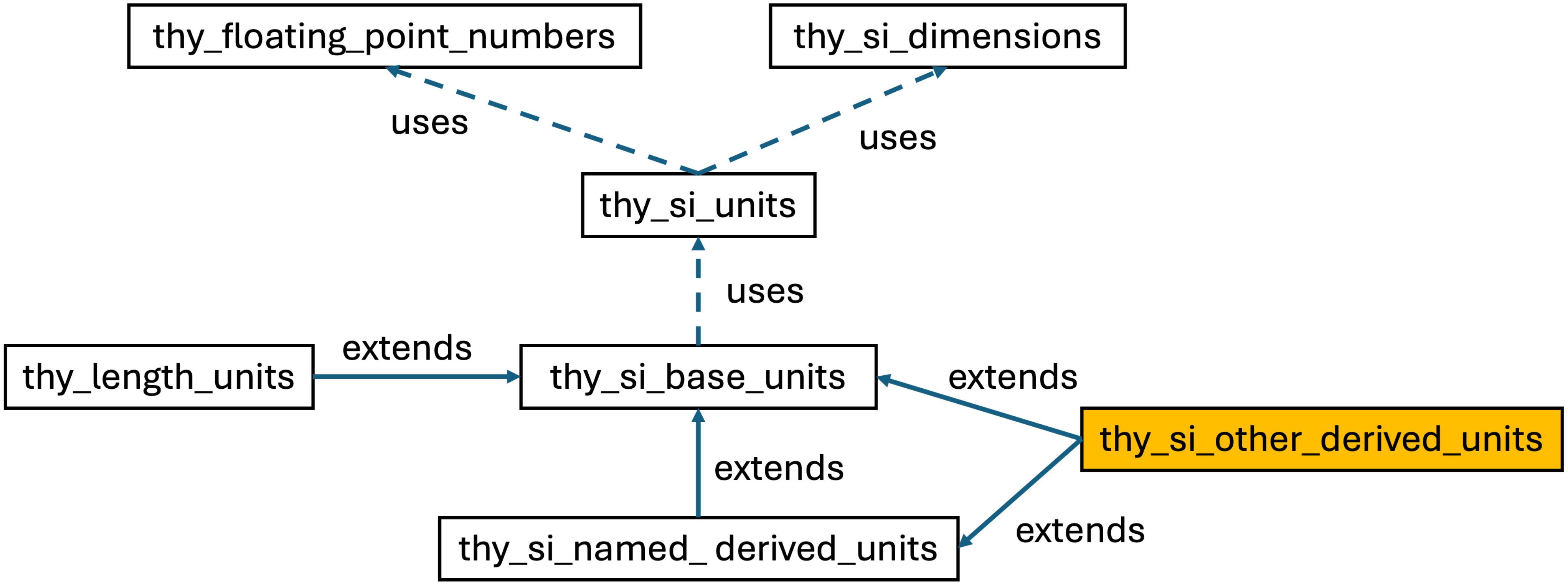


THE NAMED DERIVED UNIT FORMALISATION

- **Derived units** $\longrightarrow$ defined as products of powers of base units (dimensions).
- **Coherent derived units** $\longrightarrow$ occur when the numerical factor in the product is one.
- **Special coherent derived units** $\longrightarrow$ 22 units in the SI have special names, e.g. radian, hertz, coulomb, degree Celsius, etc.
- These 22 named units are defined by combining the seven base units.
- These 22 coherent derived units + 7 base units form the core of the **International System of Units** (**SI**).

```
OPERATORS
  HERTZ_UNIT ≙ // 1/s
    UNIT_INV(SECOND_UNIT)
  COULOMB_UNIT ≙ // s A
    UNIT_MULT(SECOND_UNIT, AMPERE_UNIT)
  NEWTON_UNIT ≙ // kg m / s^2
    UNIT_MULT(KILO_GRAM_UNIT, UNIT_DIV(METRE_UNIT, UNIT_MULT(SECOND_UNIT,SECOND_UNIT)))
  ...
```

PROPOSED APPROACH



THE OTHER DERIVED UNIT FORMALISATION

The **seven base units** and **twenty-two units with special names** may be combined to express the units of other derived physical quantities.

```
OPERATORS
  SQUARE_METRE_UNIT ≙ //area m^2
    UNIT_MULT(METRE_UNIT, METRE_UNIT)
  CUBIC_METRE_UNIT ≙ // volume m^3
    UNIT_MULT(SQUARE_METRE_UNIT, METRE_UNIT)
  METRE_PER_SECOND_UNIT ≙ // speed, velocity m/s
    UNIT_DIV(METRE_UNIT, SECOND_UNIT)
  METRE_PER_SECOND_SQUARED_UNIT ≙ // acceleration m/s^2
    UNIT_DIV(METRE_UNIT, UNIT_MULT(SECOND_UNIT, SECOND_UNIT))
  ...
  COULOMB_PER_CUBIC_METRE_UNIT ≙ // electric charge density
    UNIT_DIV(COULOMB_UNIT, CUBIC_METRE_UNIT) // coulomb/m^3 = s.A/m^3
  ...
```

NON-SI UNITS FORMALISATION

The most used **Non-SI units** that accepted for use with the SI Units and that we can find in the official SI Brochure, can be formalised as a `SI_UNIT_Type` datatype

```
OPERATORS
  NONSI_UNIT(v ∈ FLOAT_Type, u ∈ SI_UNITE_Type) ≙
      SI_UNIT(v mult multiplier(u), dimension(u), offset(u))

  MINUTE_UNIT ≙ NONSI_UNIT(FLOAT(60), SECOND_UNIT)
  HOUR_UNIT ≙ NONSI_UNIT(FLOAT(3600), SECOND_UNIT)
  HECTARE_UNIT ≙ NONSI_UNIT(FLOAT(10000), SQUARE_METRE_UNIT)
  LITRE_UNIT ≙ NONSI_UNIT(NEW_FLOAT(1,-3), CUBIC_METRE_UNIT)
  ...
```

OUTLINE

Back to the outline - Back to the begin

REFINEMENT BASED APPROACH

We have used the **Event-B refinement** to deal separately with the problem of using small values and the problem of correctly using measurement units.

OUTLINE

Back to the outline - Back to the begin

CONCLUSION - PROPOSAL

- Develop a measurement units theory using the Theory plugin.
- Extend the Event-B type-checking system to handle reasoning about measurement units.
- Introduce a formal method for annotating Event-B variables with their associated units of measurement.

THANK YOU

PDF version of the slides

Back to the begin - Back to the outline