A Sound Type System for Physical Quantities, Units, and Measurements

Simon Foster and Burkhart Wolff
University of York,
LMF, Université Paris-Saclay

LMF Seminary @ Fremigny, 3 Dec 2021

Abstract

- I present an Isabelle theory building a formal model for both the International System of Quantities (ISQ) and the International System of Units (SI), which are both fundamental for physics and engineering. Both the ISQ and the SI are deeply integrated into Isabelle's type system. Quantities are parameterised by dimension types, which correspond to base vectors, and thus only quantities of the same dimension can be equated. Since the underlying "algebra of quantities" induces congruences on quantity and SI types, specific tactic support is developed to capture these. Our construction is validated by a test-set of known equivalences between both quantities and SI units. Moreover, the presented theory can be used for type-safe conversions between the SI system and others, like the British Imperial System (BIS).

Motivation

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as
    - mass,

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as
  - mass,
  - length,

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as
  - mass,
  - length,
  - time,

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as
  - mass,
  - length,
  - time,
  - electric current, etc.

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as
    - mass,
    - length,
    - time,
    - electric current, etc.
- These phenomena, called quantities (“grandeurs” / “Größen”), are linked via an algebra to derived concepts such as

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as
  - mass,
  - length,
  - time,
  - electric current, etc.
- These phenomena, called quantities (“grandeurs” / “Größen”), are linked via an algebra to derived concepts such as
  - speed,

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as
    - mass,
    - length,
    - time,
    - electric current, etc.
- These phenomena, called quantities (“grandeurs” / “Größen”), are linked via an algebra to derived concepts such as
    - speed,
    - force,

Motivation

- Modern Physics is based on the concept of quantifiable properties for phenomena such as
  - mass,
  - length,
  - time,
  - electric current, etc.
- These phenomena, called quantities (“grandeurs” / “Größen”), are linked via an algebra to derived concepts such as
  - speed,
  - force,
  - energy, and many others.

Motivation

Motivation

- The algebra of quantities allows for a *dimensional analysis* of physical equations.

Motivation

- The algebra of quantities allows for a *dimensional analysis* of physical equations.
- … and represents a kind of type-system going back to Newtons Principia Naturalis and the translation of Du Châtelet.

Motivation

- The algebra of quantities allows for a *dimensional analysis* of physical equations.
- … and represents a kind of type-system going back to Newtons Principia Naturalis and the translation of Du Châtelet.
- In parallel, physics developed the research field “metrology” for the study of the measurement of physical quantities.

Background I

Background I

- Relevant Standard:
  “Vocabulaire International de Metrologie” (VIM)
  [Bureau International des Poids et des Mesures, BIPM]

Background I

- Relevant Standard:
  "Vocabulaire International de Metrologie" (VIM)
  [Bureau International des Poids et des Mesures, BIPM]

- The VIM defines:

Background I

- Relevant Standard:
  “Vocabulaire International de Metrologie” (VIM)
  [Bureau International des Poids et des Mesures, BIPM]

- The VIM defines:
  - International System of Quantities (ISQ)

Background I

- Relevant Standard:
  "Vocabulaire International de Metrologie" (VIM)
  [Bureau International des Poids et des Mesures, BIPM]

- The VIM defines:
  - International System of Quantities (ISQ)
  - Système International of Mesurement Units (SI)

Background I

Background I

- Base Dimensions / Units: Derived Dimensions / Units

Background I

- Base Dimensions / Units: Derived Dimensions / Units

Unit name	Unit symbol	Dimension symbol	Quantity name
second [n 1]	s	T	time
metre	m	L	length
kilogram [n 2]	kg	M	mass
ampere	A	I	electric current
kelvin	K	Θ	thermodynamic temperature
mole	mol	N	amount of substance
candela	cd	J	luminous intensity

Background I

- Base Dimensions / Units: Derived Dimensions / Units

Unit name	Unit symbol	Dimension symbol	Quantity name
second [n 1]	s	T	time
metre	m	L	length
kilogram [n 2]	kg	M	mass
ampere	A	I	electric current
kelvin	K	Θ	thermodynamic temperature
mole	mol	N	amount of substance
candela	cd	J	luminous intensity

Name	Symbol	Derived quantity	Typical symbol
square metre	m^2	area	A
cubic metre	m^3	volume	V
metre per second	m/s	speed, velocity	v
metre per second squared	m/s^2	acceleration	a
reciprocal metre	m^{-1}	wavenumber	σ, $\tilde{\nu}$
		vergence (optics)	V, $1/f$
kilogram per cubic metre	kg/m^3	density	ρ

Background I

- Base Dimensions / Units: Derived Dimensions / Units

Unit name	Unit symbol	Dimension symbol	Quantity name
second [n 1]	s	T	time
metre	m	L	length
kilogram [n 2]	kg	M	mass
ampere	A	I	electric current
kelvin	K	Θ	thermodynamic temperature
mole	mol	N	amount of substance
candela	cd	J	luminous intensity

Name	Symbol	Derived quantity	Typical symbol
square metre	m^2	area	A
cubic metre	m^3	volume	V
metre per second	m/s	speed, velocity	v
metre per second squared	m/s^2	acceleration	a
reciprocal metre	m^{-1}	wavenumber	$\sigma, \tilde{\nu}$
		vergence (optics)	V, $1/f$
kilogram per cubic metre	kg/m^3	density	ρ

Name	Symbol	Quantity	In SI base units	In other SI units
hertz	Hz	frequency	s^{-1}	
newton	N	force, weight	$kg \cdot m \cdot s^{-2}$	
pascal	Pa	pressure, stress	$kg \cdot m^{-1} \cdot s^{-2}$	N/m^2
joule	J	energy, work, heat	$kg \cdot m^2 \cdot s^{-2}$	$N \cdot m = Pa \cdot m^3$
watt	W	power, radiant flux	$kg \cdot m^2 \cdot s^{-3}$	J/s
volt	V	electrical potential	$kg \cdot m^2 \cdot s^{-3} \cdot A^{-1}$	W/A = J/C

Background I

- Base Dimensions / Units: Derived Dimensions / Units

Unit name	Unit symbol	Dimension symbol	Quantity name
second [n 1]	s	T	time
metre	m	L	length
kilogram [n 2]	kg	M	mass
ampere	A	I	electric current
kelvin	K	Θ	thermodynamic temperature
mole	mol	N	amount of substance
candela	cd	J	luminous intensity

Name	Symbol	Derived quantity	Typical symbol
square metre	m^2	area	A
cubic metre	m^3	volume	V
metre per second	m/s	speed, velocity	v
metre per second squared	m/s^2	acceleration	a
reciprocal metre	m^{-1}	wavenumber	$\sigma, \tilde{\nu}$
		vergence (optics)	$V, 1/f$
kilogram per cubic metre	kg/m^3	density	ρ

Background I

- Base Dimensions / Units: Derived Dimensions / Units

Unit name	Unit symbol	Dimension symbol	Quantity name
second [n 1]	s	T	time
metre	m	L	length
kilogram [n 2]	kg	M	mass
ampere	A	I	electric current
kelvin	K	Θ	thermodynamic temperature
mole	mol	N	amount of substance
candela	cd	J	luminous intensity

Name	Symbol	Derived quantity	Typical symbol
square metre	m^2	area	A
cubic metre	m^3	volume	V
metre per second	m/s	speed, velocity	v
metre per second squared	m/s^2	acceleration	a
reciprocal metre	m^{-1}	wavenumber	$\sigma, \tilde{v}$
		vergence (optics)	V, $1/f$
kilogram per cubic metre	kg/m^3	density	ρ

Nm = J

Objective

Objective

- Deep, semantically sound integration into the Isabelle type system

Objective

- Deep, semantically sound integration into the Isabelle type system
- … including the “dimension algebra”

Objective

- Deep, semantically sound integration into the Isabelle type system
- … including the “dimension algebra”
- … being polymorphic wrt. scalars, so “'α [m/s]”

Objective

- Deep, semantically sound integration into the Isabelle type system
- … including the “dimension algebra”
- … being polymorphic wrt. scalars, so “'α [m/s]”
    - ℤ[m/s]

Objective

- Deep, semantically sound integration into the Isabelle type system
- … including the “dimension algebra”
- … being polymorphic wrt. scalars, so “'α [m/s]”
  - ℤ[m/s]
  - ℝ[m/s] — 1-dimensional continuous speed

Objective

- Deep, semantically sound integration into the Isabelle type system
- … including the "dimension algebra"
- … being polymorphic wrt. scalars, so "'α [m/s]"
  - $\mathbb{Z}$[m/s]
  - $\mathbb{R}$[m/s] — 1-dimensional continuous speed
  - $\mathbb{R}^3$ [m] ⇒ $\mathbb{R}^3$ [N] — 3-dimensional force field

Objective

- Deep, semantically sound integration into the Isabelle type system
- … including the “dimension algebra”
- … being polymorphic wrt. scalars, so “'α [m/s]”
  - $\mathbb{Z}$[m/s]
  - $\mathbb{R}$[m/s] — 1-dimensional continuous speed
  - $\mathbb{R}^3$ [m] ⇒ $\mathbb{R}^3$ [N] — 3-dimensional force field
  - double^3[m/s] — double precision float

Background II

Background II

- Isabelle is built upon a Curry-Style type-system: (most-general) types were automatically inferred

λ x. E vs λx:τ. E

$x + y = y + x$

Background II

- Isabelle is built upon a Curry-Style type-system: (most-general) types were automatically inferred

λ x. E vs $\lambda x{:}\tau$. E

$x + y = y + x$

- Consequence: Calculations on types are in general impossible

Background II

- Isabelle is built upon a Curry-Style type-system: (most-general) types were automatically inferred

  λ x. E vs $\lambda x{:}\tau$. E

  x + y = y + x

- Consequence: Calculations on types are in general impossible

- … in particular with the universal HOL equality
  _=_:: $\alpha \Rightarrow \alpha \Rightarrow$bool rules out terms like

  1::nat[Nm] = 1::nat[J]

Background II

Background II

On the other hand, Isabelle type-system supports order-sorted polymorphism:

Background II

On the other hand, Isabelle type-system supports order-sorted polymorphism:

```
class ord =
  fixes less_eq :: "'a ⇒ 'a ⇒ bool"
    and less :: "'a ⇒ 'a ⇒ bool"
notation  less_eq  ("'(≤')") and less
```

Background II

On the other hand, Isabelle type-system supports order-sorted polymorphism:

```
class ord =
  fixes less_eq :: "'a ⇒ 'a ⇒ bool"
    and less :: "'a ⇒ 'a ⇒ bool"
notation  less_eq  ("'(≤')") and less
```

… which can be associated to semantic properties

Background II

On the other hand, Isabelle type-system supports order-sorted polymorphism:

```
class ord =
  fixes less_eq :: "'a ⇒ 'a ⇒ bool"
   and less :: "'a ⇒ 'a ⇒ bool"
notation  less_eq  ("'(≤')") and less
```

```
class preorder = ord +
  assumes less_le_not_le:
        "x < y ⟷ x ≤ y ∧ ¬ (y ≤ x)"
  and order_refl [iff]: "x ≤ x"
  and order_trans:
        "x ≤ y ⟹ y ≤ z ⟹ x ≤ z"
```

… which can be associated to semantic properties

Background II

Background II

- … and used for inductively defined sub-classes of types (**fun** is the internal name for the type constructor ⇒)

Background II

- … and used for inductively defined sub-classes of types (**fun** is the internal name for the type constructor ⇒)

```
instantiation "fun" :: (type, ord) ord

begin

definition  le_fun_def:
     "f ≤ g ⟷ (∀x. f x ≤ g x)"

definition less_fun_def:
     "(f::'a ⇒ 'b) < g ⟷ f ≤ g ∧ ¬ (g ≤ f)"


instance ..
```

```
instance "fun" :: (type, preorder) preorder
proof

qed (auto simp add: le_fun_def
        less_fun_def intro: order_trans
        order.antisym)

end
```

Background II

- … and used for inductively defined sub-classes of types (**fun** is the internal name for the type constructor ⇒)

instantiation "fun" :: (type, ord) ord

begin

definition le_fun_def:
"f ≤ g ⟷ (∀x. f x ≤ g x)"

definition less_fun_def:
"(f::'a ⇒ 'b) < g ⟷ f ≤ g ∧ ¬ (g ≤ f)"

instance ..

instance "fun" :: (type, preorder) preorder
proof

qed (auto simp add: le_fun_def
less_fun_def intro: order_trans
order.antisym)

end

The Plan

The Plan

- Semantic Domain of Dimension Types: an Executable Algebra

The Plan

- Semantic Domain of Dimension Types:
  an Executable Algebra
- Constructing the type-language of dimension types

The Plan

- Semantic Domain of Dimension Types:
  an Executable Algebra
- Constructing the type-language of dimension types
- Lifting to ISQ and SI Types

The Plan

- Semantic Domain of Dimension Types: an Executable Algebra
- Constructing the type-language of dimension types
- Lifting to ISQ and SI Types
- The derived algebra of Dimensions and Measurement (as defined by VIM)

The Plan

- Semantic Domain of Dimension Types:
  an Executable Algebra
- Constructing the type-language of dimension types
- Lifting to ISQ and SI Types
- The derived algebra of Dimensions and Measurement (as defined by VIM)
- Proof Support

An Executable Algebra of Dimensions

An Executable Algebra of Dimensions

- An finite-indexed family of types: D^I
-

An Executable Algebra of Dimensions

- An finite-indexed family of types:
-

```
typedef ('D, 'I) dimvec =
          "UNIV::('I::enum⇒'D) set"
```

An Executable Algebra of Dimensions

- An finite-indexed family of types:

```
typedef ('D, 'I) dimvec =
          "UNIV::('I::enum⇒'D) set"
```

- Theorem:

  (if I is finite and ordered)
  (D,0,_+_,_-) abelian group

  then $(D^I, 1, \_\circ\_, \_^{-1})$ abelian group

An Executable Algebra of Dimensions

- An finite-indexed family of types:
-

```
typedef ('D, 'I) dimvec =
          "UNIV::('I::enum⇒'D) set"
```

```
instance dimvec :: (ab_group_add,enum) ab_group_mult
proof …
```

An Executable Algebra of Dimensions

- An finite-indexed family of types:

```
typedef ('D, 'I) dimvec =
          "UNIV::('I::enum⇒'D) set"
```

-

```
instance dimvec :: (ab_group_add,enum) ab_group_mult
proof …
```

- Relevant instance: $\mathbb{Z}^{\{Length,Mass,Time,Current,Temp, Amount,Intensity\}}$

An Executable Algebra of Dimensions

- An finite-indexed family of types:

```
typedef ('D, 'I) dimvec =
           "UNIV::('I::enum⇒'D) set"
```

-

```
instance dimvec :: (ab_group_add,enum) ab_group_mult
proof …
```

- Relevant instance:

```
type_synonym Dimension = "(int, sdim) dimvec"
```

An Executable Algebra of Dimensions

- An finite-indexed family of types:

```
typedef ('D, 'I) dimvec =
          "UNIV::('I::enum⇒'D) set"
```

-

```
instance dimvec :: (ab_group_add,enum) ab_group_mult
proof …
```

- Relevant instance:

```
type_synonym Dimension = "(int, sdim) dimvec"
```

- Concept: A base dimension is a dimension where precisely one component has power 1

An Executable Algebra of Dimensions

```
abbreviation LengthBD      ("L") where "L ≡ mk_BaseDim Length"
abbreviation MassBD        ("M") where "M ≡ mk_BaseDim Mass"
abbreviation TimeBD        ("T") where "T ≡ mk_BaseDim Time"
abbreviation CurrentBD     ("I") where "I ≡ mk_BaseDim Current"
abbreviation TemperatureBD ("Θ") where "Θ ≡ mk_BaseDim Temperature"
abbreviation AmountBD      ("N") where "N ≡ mk_BaseDim Amount"
abbreviation IntensityBD   ("J") where "J ≡ mk_BaseDim Intensity"

abbreviation "BaseDimensions ≡ {L, M, T, I, Θ, N, J}"

lemma BD_mk_dimvec [si_def]:
  "L = mk_dimvec [1, 0, 0, 0, 0, 0, 0]"
  "M = mk_dimvec [0, 1, 0, 0, 0, 0, 0]"
  "T = mk_dimvec [0, 0, 1, 0, 0, 0, 0]"
  "I = mk_dimvec [0, 0, 0, 1, 0, 0, 0]"
  "Θ = mk_dimvec [0, 0, 0, 0, 1, 0, 0]"
  "N = mk_dimvec [0, 0, 0, 0, 0, 1, 0]"
  "J = mk_dimvec [0, 0, 0, 0, 0, 0, 1]"
  by (simp_all add: mk_BaseDim_code eval_nat_numeral)
```

An Executable Algebra of Dimensions

An Executable Algebra of Dimensions

- The executable algebra gives terms for a semantic universe of dimension types and its theory.
  In Isabelle, this gives the following infrastructure:

An Executable Algebra of Dimensions

- The executable algebra gives terms for a semantic universe of dimension types and its theory.
  In Isabelle, this gives the following infrastructure:

  - term "$L \cdot M \cdot T^{-2} / M$"

An Executable Algebra of Dimensions

- The executable algebra gives terms for a semantic universe of dimension types and its theory.
  In Isabelle, this gives the following infrastructure:

  - term “$L \cdot M \cdot T^{-2} / M$”
  - value "$L \cdot M \cdot T^{-2} / M$"

An Executable Algebra of Dimensions

- The executable algebra gives terms for a semantic universe of dimension types and its theory.
  In Isabelle, this gives the following infrastructure:

  - term “$L \cdot M \cdot T^{-2} / M$”
  - value "$L \cdot M \cdot T^{-2} / M$"
  - lemma "$L \cdot M \cdot T^{-2} / M$ = mk_dimvec [1, 0, - 2, 0, 0, 0, 0]”

Types for Dimensions

Types for Dimensions

- Inductive subclass of types dim_type's

Types for Dimensions

- Inductive subclass of types dim_type’s
  - … that posses a “semantic interpretation” in a physical dimension.

Types for Dimensions

- Inductive subclass of types dim_type's
  - … that posses a "semantic interpretation" in a physical dimension.

```
class dim_type = unitary +
  fixes   dim_ty_sem :: "'a itself ⇒ Dimension"
```

Types for Dimensions

- Inductive subclass of types dim_type's
  - … that posses a "semantic interpretation" in a physical dimension.

```
class dim_type = unitary +
  fixes   dim_ty_sem :: "'a itself ⇒ Dimension"
```

  - … we define the type constructor symbols L, M, T, I, Θ, N, J and define them arbitrarily

Types for Dimensions

- Inductive subclass of types dim_type's
  - … that posses a "semantic interpretation" in a physical dimension.

```
class dim_type = unitary +
  fixes   dim_ty_sem :: "'a itself ⇒ Dimension"
```

  - … we define the type constructor symbols L, M, T, I, Θ, N, J and define them arbitrarily

```
typedef           Length      = "UNIV :: unit set"
type_synonym L = Length
typedef           Mass      = "UNIV :: unit set"
type_synonym M = Mass
…
```

Types for Dimensions

Types for Dimensions

- Inductive subclass of types dim_type's

Types for Dimensions

- Inductive subclass of types dim_type’s
  - … by setting for each type constructor symbol its semantics to the corresponding value in the dimension algebra:

Types for Dimensions

- Inductive subclass of types dim_type's
  - … by setting for each type constructor symbol its semantics to the corresponding value in the dimension algebra:

```
instantiation Length :: dim_type
begin
  definition [si_eq]: "dim_ty_sem_Length (_::Length itself) = L"
  instance <proof … >
end
```

Types for Dimensions

Types for Dimensions

- Inductive subclass of types dim_type's

Types for Dimensions

- Inductive subclass of types dim_type’s
  - remains to construct the product inside the dim_type-class:

Types for Dimensions

- Inductive subclass of types dim_type's
  - remains to construct the product inside the dim_type-class:

```
typedef ('a::dim_type, 'b::dim_type) DimTimes (infixl "·" 69) = "UNIV :: unit set

instantiation DimTimes :: (dim_type, dim_type) dim_type
begin
  definition dim_ty_sem_mult :: "('a · 'b) itself ⇒ Dimension"
  where    "dim_ty_sem_mult x = dim_ty_sem(TYPE 'a) · dim_ty_sem(TYPE 'b)"
  instance <proof …>
end
```

Types for Dimensions

- Inductive subclass of types dim_type's
  - remains to construct the product inside the dim_type-class:

```
typedef ('a::dim_type, 'b::dim_type) DimTimes (infixl "·" 69) = "UNIV :: unit set

instantiation DimTimes :: (dim_type, dim_type) dim_type
begin
  definition dim_ty_sem_mult :: "('a · 'b) itself ⇒ Dimension"
  where    "dim_ty_sem_mult x = dim_ty_sem(TYPE 'a) · dim_ty_sem(TYPE 'b)"
  instance <proof …>
end
```

  - the inversion constructor ($\_^{-1}$) is built analogously.

Types for Quantities and Measurements

Types for Quantities and Measurements

- We repeat this construction analogously for

Types for Quantities and Measurements

- We repeat this construction analogously for
  - Quantities and Measurement Systems

Types for Quantities and Measurements

- We repeat this construction analogously for
  - Quantities and Measurement Systems

```
record ('A, 'I::enum) Quantity =
  mag  :: 'A                    — ‹ Magnitude of the quantity. ›
  dim  :: "(ℤ, 'I) dimvec"  — ‹ Dimension of the quantity. ›

record ('A, 'I::enum, 's::unit_system) Measurement_System
   = "('A, 'I::enum) Quantity" +
      unit_sys  :: 's — ‹ The system of units being employed, e.g. SI, BIS, ACS,… ›
```

Types for Quantities and Measurements

- We repeat this construction analogously for
  - Quantities and Measurement Systems

```
record ('A, 'I::enum) Quantity =
  mag  :: 'A                   — ‹ Magnitude of the quantity. ›
  dim  :: "(ℤ, 'I) dimvec"  — ‹ Dimension of the quantity. ›

record ('A, 'I::enum, 's::unit_system) Measurement_System
   = "('A, 'I::enum) Quantity" +
     unit_sys  :: 's — ‹ The system of units being employed, e.g. SI, BIS, ACS,… ›
```

  - … where the tag-type *'s* says:

this magnitude has been measured in system 's …

Types for Quantities and Measurements

Types for Quantities and Measurements

- We repeat this construction analogously for

Types for Quantities and Measurements

- We repeat this construction analogously for
  - … and this leads to the type definition, that "freezes" a non-empty set to a type:

Types for Quantities and Measurements

- We repeat this construction analogously for
  - … and this leads to the type definition, that "freezes" a non-empty set to a type:

```
typedef ('D, 'd::dim_type, 's::unit_system) gmt  ("_[_, _]" [999,0,0] 999)
        = "{x :: ('D, sdim, 's) Measurement_System. dim x = dim_ty_sem(TYPE 'D)}"
```

Types for Quantities and Measurements

- We repeat this construction analogously for
  - … and this leads to the type definition, that “freezes” a non-empty set to a type:

```
typedef ('D, 'd::dim_type, 's::unit_system) gmt  ("_[_, _]" [999,0,0] 999)
        = "{x :: ('D, sdim, 's) Measurement_System. dim x = dim_ty_sem(TYPE 'D)}"
```

  - … which induces the syntax for type expressions:

Types for Quantities and Measurements

- We repeat this construction analogously for
  - … and this leads to the type definition, that "freezes" a non-empty set to a type:

```
typedef ('D, 'd::dim_type, 's::unit_system) gmt  ("_[_, _]" [999,0,0] 999)
        = "{x :: ('D, sdim, 's) Measurement_System. dim x = dim_ty_sem(TYPE 'D)}"
```

  - … which induces the syntax for type expressions:

$$\mathbb{Z}[L \cdot T^{-1}, SI]$$

Types for Quantities and Measurements

- We repeat this construction analogously for
  - … and this leads to the type definition, that "freezes" a non-empty set to a type:

```
typedef ('D, 'd::dim_type, 's::unit_system) gmt  ("_[_, _]" [999,0,0] 999)
        = "{x :: ('D, sdim, 's) Measurement_System. dim x = dim_ty_sem(TYPE 'D)}"
```

  - … which induces the syntax for type expressions:

$\mathbb{Z}[L \cdot T^{-1}, SI]$

$\mathbb{Z}[m \cdot s^{-1}]$

Types for Quantities and Measurements

- We repeat this construction analogously for
  - … and this leads to the type definition, that “freezes” a non-empty set to a type:

```
typedef ('D, 'd::dim_type, 's::unit_system) gmt  ("_[_, _]" [999,0,0] 999)
        = "{x :: ('D, sdim, 's) Measurement_System. dim x = dim_ty_sem(TYPE 'D)}"
```

  - … which induces the syntax for type expressions:

$\mathbb{Z}[L \cdot T^{-1}, SI]$

$\mathbb{R}[L \cdot T^{-1}, SI]$

$\mathbb{Z}[m \cdot s^{-1}]$

$\mathbb{R}[m \cdot s^{-1}]$

Types for Quantities and Measurements

- We repeat this construction analogously for
  - … and this leads to the type definition, that "freezes" a non-empty set to a type:

```
typedef ('D, 'd::dim_type, 's::unit_system) gmt  ("_[_, _]" [999,0,0] 999)
        = "{x :: ('D, sdim, 's) Measurement_System. dim x = dim_ty_sem(TYPE 'D)}"
```

  - … which induces the syntax for type expressions:

$\mathbb{Z}[L \cdot T^{-1}, SI]$	$\mathbb{Z}[m \cdot s^{-1}]$
$\mathbb{R}[L \cdot T^{-1}, SI]$	$\mathbb{R}[m \cdot s^{-1}]$
$\mathbb{R}^3\,[L,SI] \Rightarrow \mathbb{R}^3\,[F,SI]$	$\mathbb{R}^3\,[m] \Rightarrow \mathbb{R}^3\,[N]$

Algebra of Quantities and Measurements

Algebra of Quantities and Measurements

- The semantic equality on Dimension induces for types of dim_type class a congruence:

Algebra of Quantities and Measurements

- The semantic equality on Dimension induces for types of dim_type class a congruence:

$(\cong_Q)$ :: 'D['a::dim_type,'s::unit_system]⇒'D['b::dim_type,'s] ⇒ bool

Algebra of Quantities and Measurements

- The semantic equality on Dimension induces for types of dim_type class a congruence:

$(\cong_Q)$ :: 'D['a::dim_type,'s::unit_system]⇒'D['b::dim_type,'s] ⇒ bool

- … which allow for the derivation of the algebra:

Algebra of Quantities and Measurements

- The semantic equality on Dimension induces for types of dim_type class a congruence:

$(\cong_Q)$ :: 'D['a::dim_type,'s::unit_system]⇒'D['b::dim_type,'s] ⇒ bool

- … which allow for the derivation of the algebra:

"$a \cong_Q a$"	"$a \cong_Q b \Longrightarrow b \cong_Q a$"	"$\mathbf{1} \cdot x \cong_Q x$"
"$[\![a \cong_Q b; b \cong_Q c]\!] \Longrightarrow a \cong_Q c$"	"$(x \cdot y) \cdot z \cong_Q x \cdot (y \cdot z)$"	"$x \cdot \mathbf{1} \cong_Q x$"
"$a *_Q x + y = (a *_Q x) + (a *_Q y)$"	"$a + b *_Q x = (a *_Q x) + (b *_Q x)$"	"$0 *_Q x = 0$"
"$a *_Q b *_Q x = a \cdot b *_Q x$"	"$(a *_Q x) \cdot y = a *_Q x \cdot y$"	"$1 *_Q x = x$"
"$-a *_Q x = a *_Q -x$"	"$x \cdot (a *_Q y) = a *_Q x \cdot y$".	"$a *_Q x \cong_Q (a *_Q \mathbf{1}) \cdot x$"

Proof Automation

Proof Automation

- Semantic interpretation and semantic equivalences are organised to proof support,

Proof Automation

- Semantic interpretation and semantic equivalences are organised to proof support,
- … wrapped up in tactics “si_simp” and “si_calc”

Proof Automation

- Semantic interpretation and semantic equivalences are organised to proof support,
- … wrapped up in tactics “si_simp” and “si_calc”
- … we checked all the VIM reference equations:

Proof Automation

- Semantic interpretation and semantic equivalences are organised to proof support,
- … wrapped up in tactics “si_simp” and “si_calc”
- … we checked all the VIM reference equations:

lemma cel_to_kelvin: "T°C = (T $*_Q$ kelvin) + (273.15 $*_Q$ kelvin)" by (si_simp)

theorem metre_definition:

"1 $*_Q$ metre $\cong_Q$ (**c** / (299792458 $*_Q$ **1**)) · second"

"1 $*_Q$ metre $\cong_Q$ (9192631770 / 299792458) $*_Q$ (**c** / Δv_{Cs})"

where Δv_{Cs} ≡ 9192631770 $*_Q$ hertz (* caesium frequency *)

Conclusion

Conclusion

- A Formal Model of Dimensions and Measurements

Conclusion

- A Formal Model of Dimensions and Measurements
- A derived sound and complete (by construction) type system for ISQ and SI

Conclusion

- A Formal Model of Dimensions and Measurements
- A derived sound and complete (by construction) type system for ISQ and SI
- Validated by the catalogue of VIM definitions for SI units and derived equivalences

Conclusion

- A Formal Model of Dimensions and Measurements
- A derived sound and complete (by construction) type system for ISQ and SI
- Validated by the catalogue of VIM definitions for SI units and derived equivalences

- … useful in physics and engineering

Conclusion

- A Formal Model of Dimensions and Measurements
- A derived sound and complete (by construction) type system for ISQ and SI
- Validated by the catalogue of VIM definitions for SI units and derived equivalences

- … useful in physics and engineering
- … available as component in the Isabelle AFP