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

Simon Foster   Burkhart Wolff

February 6, 2022

Abstract

We present a theory in Isabelle/HOL [7] that builds 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 [2]. 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" from [2] 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).

1 Introduction

Modern Physics is based on the concept of quantifiable properties of physical phenomena such as mass, length, time, current, etc. These phenomena, called *quantities*, are linked via an *algebra of quantities* to derived concepts such as speed, force, and energy. The latter allows for a *dimensional analysis* of physical equations, which had already been the backbone of Newtonian Physics. In parallel, physicians developed their own research field called "metrology" defined as a scientific study of the *measurement* of physical quantities.

The relevant international standard for quantities and measurements is distributed by the *Bureau International des Poids et des Mesures* (BIPM), which also provides the *Vocabulaire International de Métrologie* (VIM) [2]. The VIM actually defines two systems: the *International System of Quantities* (ISQ) and the *International System of Units* (SI, abbreviated from the French 'Système international d'unités'). The latter is also documented in the *SI Brochure* [3], a standard that is updated periodically, most recently in 2019. Finally, the VIM defines concrete reference measurement procedures as well as a terminology for measurement errors.

Conceived as a refinement of the ISQ, the SI comprises a coherent system of units of measurement built on seven base units, which are the metre, kilogram, second, ampere, kelvin, mole, candela, and a set of twenty prefixes to the unit names and unit symbols, such as milli- and kilo-, that may be used when specifying multiples and fractions of the units. The system also specifies names for 22 derived units, such as lumen and watt, for other common physical quantities. While there is still nowadays a wealth of different measuring systems such as the *British Imperial System* (BIS) and the *United States Customary System* (USC), the SI is more or less the de-facto reference behind all these systems. [1]

The present Isabelle theory builds a formal model for both the ISQ and the SI, together with a deep integration into Isabelle's order-sorted polymorphic type system [6]. Quantities and units are represented in a way that they have a *quantity type* as well as a *unit type* based on its base vectors and their magnitudes. Since the algebra of quantities induces congruences on quantity and SI types, specific tactic support has been 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).

As a result of our theory development[2], it is possible to express "4500.0 kilogram times metre per second squared" has the type $\mathbb{R}[kg \cdot m \cdot s^{-3}]$ This type means that the magnitude *4500.0* (which by lexical convention is considered as a real number) of the dimension $M \cdot L \cdot T^{-3}$ is a quantity intended to be measured in the SI-system, which means that it actually represents a force measured in Newtons. Via a type synonym, the above type expression gets the type $\mathbb{R}$ *newton*.

In the example, the *magnitude* type part of this type is the real numbers $\mathbb{R}$. In general, however, magnitude types can be more general. If the term above is presented slightly differently as "4500 kilogram times metre per second squared", the inferred type will be $'\alpha[kg \cdot m \cdot s^{-3}]$ where $'\alpha$ is a magnitude belonging to the type-class numeral. This class comprises types like $\mathbb{N}$, $\mathbb{Z}$, 32 bit integers (*32word*), IEEE-754 floating-point numbers, as well as vectors belonging to the three-dimensional space $\mathbb{R}^3$, etc. Thus, our type-system allows to capture both conceptual entities in physics as well as implementation issues in concrete physical calculations on a computer.

As mentioned before, it is a main objective of this work to support the quantity calculus of ISQ and the resulting equations on derived SI entities (cf. [3]), both from a type checking as well as a proof-checking perspective. Our design objectives are not easily reconciled, however, and so some substantial theory engineering is required. On the one hand, we want a deep integration of dimensions and units into the Isabelle type system. On the other, we need to do normal-form calculations on types, so that, for example, the units $'\alpha[s \cdot m \cdot s^{-2}]$ and $'\alpha[m \cdot s^{-1}]$ can be equated.

Isabelle's type system follows the Curry-style paradigm, which rules out the possibility of direct calculations on type-terms (in contrast to Coq-like systems). However, our semantic interpretation of ISQ and SI requires the foundation of the heterogeneous equivalence relation $\cong_Q$ in semantic terms. This means that we can relate quantities with syntactically different dimension types, yet with same dimension semantics. This paves the way for derived rules that do computations of terms, which represent type computations indirectly. This principle is the basis for the tactic support, which allows for the dimensional type checking of key definitions of the SI system inside Isabelle/HOL, i. e. without making use of code-generated reflection. For example, the crucial definitions adapted from the SI Brochure that give the concrete definitions for the metre and the kilogram can be presented as follows:

[1] See also https://en.wikipedia.org/wiki/International_System_of_Units.

[2] The sources can be found in the Isabelle Archive of Formal Proofs at https://www.isa-afp.org/entries/Physical_Quantities.html

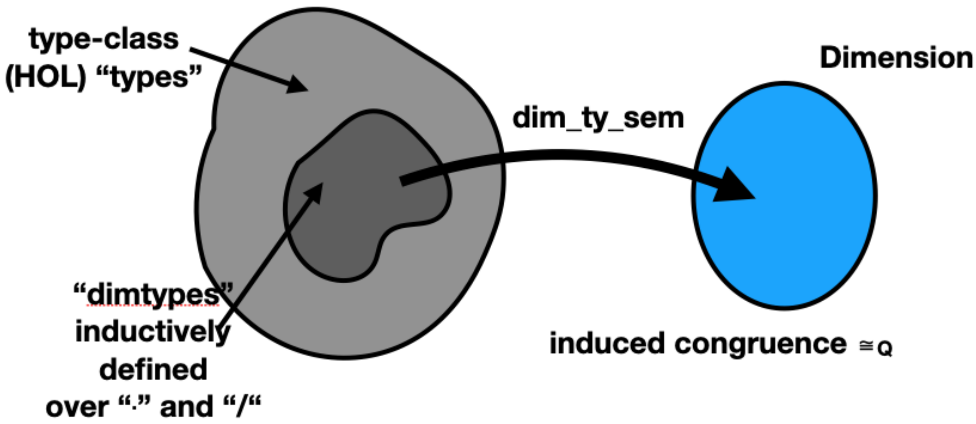

Figure 1: The "Inductive" Subset of *dim-types* interpreted in the *Dimension*-Type

theorem *metre-definition*

- $(1::'a) *_Q \mathit{metre} \cong_Q \mathbf{c} \cdot ((299792458::'a) *_Q \mathbf{1})^{-\mathbf{1}} \cdot \mathit{second}$
- $(1::'a) *_Q \mathit{metre} \cong_Q (9192631770::'a) / (299792458::'a) *_Q \mathbf{c} \cdot ((9192631770::'a) *_Q \mathit{second}^{-\mathbf{1}})^{-\mathbf{1}}$

  theorem *kilogram-definition*
- $(1::'\alpha) *_Q \mathit{kilogram} \cong_Q \mathbf{h} \cdot ((662607015::'\alpha) / (10::'\alpha)^{8} \cdot (1::'\alpha) / (10::'\alpha)^{34} *_Q \mathbf{1})^{-\mathbf{1}} \cdot \mathit{metre}^{-\mathbf{2}} \cdot \mathit{second}$

These equations giving the concrete definitions for the metre and kilogram in terms of the physical constants **c** (speed of light) and **h** (Planck constant) can be proven directly using the tactic *si−calc* provided by our theory.

2 Background: Some Advanced Isabelle Constructs

This work uses a number of features of Isabelle/HOL and its meta-logic Isabelle/Pure, that are not necessarily available in another system of the LCF-Prover family and that needs therefore some explanation:

- Type-classes and order-sorted parametric polymorphism [5, 6]. Haskell-like type-classes allow for types depend on constants and represent therefore a restricted form of dependent types.
- The meta-logic `Pure` providing mechanisms to denote types inside the term-language: $'\alpha$ *itself* denotes an unspecified type and *TYPE* a constructor that injects the language of types into the language of terms.
- Code-generation: Reflection via *eval*
- The lifting package

3 Preliminary Algebraic Structures

At the core, the multiplicative operation on physical dimension results in additions of the exponents of base vectors:

$$(M^{\alpha 1} \cdot L^{\alpha 2} \cdot T^{\alpha 3} \cdot I^{\alpha 4} \cdot \Theta^{\alpha 5} \cdot N^{\alpha 6} \cdot J^{\alpha 7}) * (M^{\beta 1} \cdot L^{\beta 2} \cdot T^{\beta 3} \cdot I^{\beta 4} \cdot \Theta^{\beta 5} \cdot N^{\beta 6} \cdot J^{\beta 7})$$
$$= (M^{\alpha 1+\beta 1} \cdot L^{\alpha 2+\beta 2} \cdot T^{\alpha 3+\beta 3} \cdot I^{\alpha 4+\beta 4} \cdot \Theta^{\alpha 5+\beta 5} \cdot N^{\alpha 6+\beta 6} \cdot J^{\alpha 7+\beta 7})$$

This motivates type classes that represent this algebraic structure. We chose to represent it for the case of vectors of arbitrary length. We define the classes *group-mult* and the abelian multiplicative groups as follows:

notation *times* (**infixl** · *70*)

class *group-mult* = *inverse* + *monoid-mult* +
  assumes *left-inverse*: *inverse a* · *a* = *1*
  assumes *multi-inverse-conv-div* [*simp*]: *a* · (*inverse b*) = *a* / *b*
...
class *ab-group-mult* = *comm-monoid-mult* + *group-mult*
...
abbreviation (*input*) *npower* :: *′α*::{*power*,*inverse*} ⇒ *nat* ⇒ *′α* ((-⁻ˉ) [*1000*,*999*] *999*)
  where *npower x n* ≡ *inverse* (*x* ^ *n*)

... and derive the respective properties:

lemma *div-conv-mult-inverse* : *a* / *b* = *a* · (*inverse b*) **...**
lemma *diff-self* : *a* / *a* = *1* **...**
lemma *mult-distrib-inverse* : (*a* ∗ *b*) / *b* = *a* **...**
lemma *mult-distrib-inverse′* : (*a* ∗ *b*) / *a* = *b* **...**
lemma *inverse-distrib* : *inverse* (*a* ∗ *b*) = (*inverse a*) ∗ (*inverse b*) **...**
lemma *inverse-divid* : *inverse* (*a* / *b*) = *b* / *a* **...**

.

On this basis we define *dimension vectors* of arbitrary size via a type definition as follows:

typedef (*′β*, *′ν*) *dimvec* = *UNIV* :: (*′ν*::*enum* ⇒ *′β*) *set*
  morphisms *dim-nth dim-lambda* **..**

.

Here, the functions *dim-nth* and *dim-lambda* represent the usual function pair that establish the isomorphism between the defined type (*′β*, *′ν*) *dimvec* and an implementing domain, in this case the universal set of type (*′ν* ⇒ *′β*) *set*. Note that the index-type *′ν* is restricted to be enumerable by type class *enum*.

Via a number of intermediate lemmas over types, we can finally establish the desired result in Isabelle compactly as follows:

instance *dimvec* :: (*ab-group-add*, *enum*) *ab-group-mult* **by** (<**proof** *omitted*)

If $'\beta$ is an abelian additive group, and if the index type $'\nu$ is enumerable, $('\beta, '\nu)$ *dimvec* is an abelian multiplicative group.

4 The Domain: ISQ Dimension Terms and Calculations

In the following, we will construct a concrete semantic domain as instance of $('\beta, '\nu)$ *dimvec*. This is where the general model of the dimension vector space of section 3 becomes a specific instance of the current ISQ standard as defined [2]; should physicians discover one day a new physical quantity, this would just imply a change of the following enumeration. Moreover, we will define the ISQ standards dimensions as *base vectors* in this vector space; historically, there had been alternative proposals of a quantity system that boil down to the choice of another eigen-vector set in this vector space.

The definition of an enumeration and the proof that it can be accommodated to the required infrastructure of the *enum*-class is straight-forward, and the construction of our domain *Dimension* follows immediately:

datatype *sdim* = *Length* | *Mass* | *Time* | *Current* | *Temperature* | *Amount* | *Intensity*

instantiation *sdim* :: *enum*
begin
  definition *enum-sdim* = [*Length*, *Mass*, *Time*, *Current*, *Temperature*, *Amount*, *Intensity*]
  definition *enum-all-sdim P* $\longleftrightarrow$ *P Length* $\wedge$ *P Mass* $\wedge$ *P Time* $\wedge$ ...
  definition *enum-ex-sdim P* $\longleftrightarrow$ *P Length* $\vee$ *P Mass* $\vee$ *P Time* $\vee$ ...
  instance <**proof** *omitted*>
end

type-synonym *Dimension* = ($\mathbb{Z}$, *sdim*) *dimvec*

Note that the *enum*-class stems from the Isabelle/HOL library and is intended to present sufficient infrastructure for the code-generator. Note, further, that [2] discusses also the possibility of rational exponents, but finally defines them as integer numbers $\mathbb{Z}$.

A base dimension is a dimension where precisely one component has power 1: it is the dimension of a base quantity. Here we define the seven base dimensions. For the concrete definition of the seven base vectors we define a constructor:

definition *mk-BaseDim* :: *sdim* $\Rightarrow$ *Dimension* **where**
mk-BaseDim d = *dim-lambda* (λ *i*. *if* (*i* = *d*) *then 1 else 0*)

which lets us achieve a first major milestone on our journey: a *term* representation of base vectors together with the capability to prove and to compute dimension-algebraic equivalences. We introduce the ISQ dimension symbols defined in [2]:

abbreviation *LengthBD* (**L**) **where** $\mathbf{L} \equiv$ *mk-BaseDim Length*
abbreviation *MassBD* (**M**) **where** $\mathbf{M} \equiv$ *mk-BaseDim Mass*
...
abbreviation *BaseDimensions* $\equiv \{\mathbf{L}, \mathbf{M}, \mathbf{T}, \mathbf{I}, \Theta, \mathbf{N}, \mathbf{J}\}$

lemma *BD-mk-dimvec* [*si-def*]:
$\mathbf{L}$ = *mk-dimvec* [*1*, *0*, *0*, *0*, *0*, *0*, *0*]
$\mathbf{M}$ = *mk-dimvec* [*0*, *1*, *0*, *0*, *0*, *0*, *0*]
...

A demonstration of a computation [3] and a proof is shown in the example below:

value $\mathbf{L}{\cdot}\mathbf{M}{\cdot}\mathbf{T}^{-2}$

lemma $\mathbf{L}{\cdot}M{\cdot}\mathbf{T}^{-2}$ = *mk-dimvec* [*1*, *1*, $-$ *2*, *0*, *0*, *0*, *0*] **by** (*simp add*: *si-def*)

Note that the multiplication operation $(\cdot)$ is inherited from the fact that the *Dimension*-type is a proven instance of the *ab-group-mult*-class. So far, the language of dimensions is represented by a shallow embedding in the *Dimension* type.

5 Dimension Types and their Semantics in Terms of the *Dimension*-Type

The next section on our road is the construction of a sub-language of type-expressions. To this end, we define a *type class* by those type-terms for which we have an interpretation function *dim-ty-sem* into the values of the *Dimension*-type. For our construction it suffices that the type-symbols of this class have a *unitary*, i. e., one-elementary, carrier-set.

class *dim-type* = *unitary* +
fixes *dim-ty-sem* :: $'\alpha$ *itself* $\Rightarrow$ *Dimension*

class *basedim-type* = *dim-type* +
assumes *is-BaseDim*: *is-BaseDim* (*dim-ty-sem* (*TYPE*($'\alpha$)))

Recall that the the type constructor $'\alpha$ *itself* from Isabelle/Pure denotes an unspecified type and *TYPE* a constructor that injects the language of types into the language of terms. We also introduce a sub-type-class *basedim-type* for base-dimensions.

The definition of the basic dimension type constructors is straightforward via a one-elementary set, *unit set*. The latter is adequate since we need just an abstract syntax for type expressions, so just one value for the `dimension`-type symbols. We define types for each of the seven base dimensions, and also for dimensionless quantities.

[3]The command **value** compiles the argument to SML code and executes it

typedef *Length* $= UNIV :: unit\ set$ **..** **setup-lifting** *type-definition-Length*
type-synonym $L = Length$
typedef *Mass* $= UNIV :: unit\ set$ **..** **setup-lifting** *type-definition-Mass*
type-synonym $M = Mass$
...

The following instantiation proof places the freshly constructed type symbol L in the class *basedim-type* by setting its semantic interpretation to the corresponding value in the *Dimension*-type.

instantiation *Length* :: *basedim-type*
begin
definition [*si-eq*]: *dim-ty-sem-Length* (α::*Length itself*) = **L**
instance <**proof** *omitted*>
end

Note that Isabelle enforces a convention to name the definition of an operation assumed in the interface of the class to be the concatenation of the interface name (e. g. *dim-ty-sem*) and the name of the class intantiation (e. g. *Length*). For the other 6 base-types we proceed analogously.

Dimension type expressions can be constructed by multiplication and division of the base dimension types above. Consequently, we need to define multiplication and inverse operators at the type level as well. On the class of dimension types (in which we have already inserted the base dimension types), the definitions of the type constructors for inner product and inverse is straightforward.

typedef ($'\alpha$::*dim-type*, $'\beta$::*dim-type*) *DimTimes* (**infixl** $\cdot$ *69*) = *UNIV* :: *unit set* **..**
setup-lifting *type-definition-DimTimes*

The type $'\alpha \cdot {}'\beta$ is parameterised by two types, $'\alpha$ and $'\beta$ that must both be elements of the *dim-type* class. As with the base dimensions, it is a unitary type as its purpose is to represent dimension type expressions. We instantiate *dim-type* with this type, where the semantics of a product dimension expression is the product of the underlying dimensions. This means that multiplication of two dimension types yields a dimension type.

instantiation *DimTimes* :: (*dim-type*, *dim-type*) *dim-type*
begin
definition *dim-ty-sem-DimTimes* :: ($'\alpha \cdot {}'\beta$) *itself* $\Rightarrow$ *Dimension* **where**
[*si-eq*]: *dim-ty-sem-DimTimes* x = (*dim-ty-sem* *TYPE*($'\alpha$)) $\cdot$ (*dim-ty-sem* *TYPE*($'\beta$))
instance by (*intro-classes*, *simp-all add*: *dim-ty-sem-DimTimes-def*, (*transfer*, *simp*)+)
end

Thus, the semantic interpretation of the product of two *dim-type*'s is a homomorphism over the product of two dimensions. Similarly, we define inversion of dimension types and prove that dimension types areclosed under this.

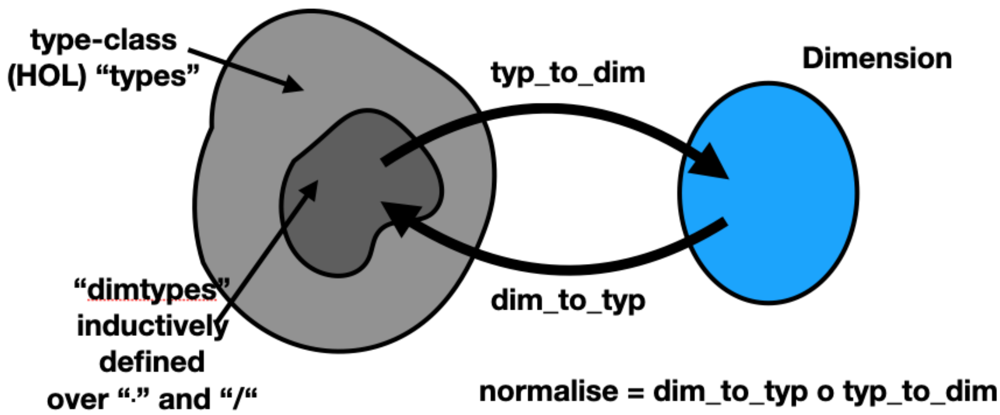

Figure 2: The ”Inductive” subset of *dim-types* interpreted in SML Lists

typedef $'\alpha$ *DimInv* $((\text{-}^{-1})$ [*999*] *999*$)$ = *UNIV* :: *unit set* **..**
setup-lifting *type-definition-DimInv*
instantiation *DimInv* :: (*dim-type*) *dim-type*
begin
  definition *dim-ty-sem-DimInv* :: $('-^{-1})$ *itself* $\Rightarrow$ *Dimension* **where**
  [*si-eq*]: *dim-ty-sem-DimInv* x = *inverse* (*dim-ty-sem* *TYPE*$('\alpha)$)
  instance by (*intro-classes*, *simp-all add*: *dim-ty-sem-DimInv-def*, (*transfer*, *simp*)+)
end

Finally, we introduce some syntactic sugar such as $'\alpha^4$ for $'\alpha \cdot {}'\alpha \cdot {}'\alpha \cdot {}'\alpha$ or $'\alpha^{-4}$ for $('\alpha^4)^{-1}$.

By the way, we also implemented two morphisms on the SML-level underlying Isabelle, which is straight-forward and omitted here (C.f. Figure 2). These functions yield for:

ML‹ *Dimension-Type.typ-to-dim* @{*typ* $L^{-2}{\cdot}M^{-1}{\cdot}T^4{\cdot}I^2{\cdot}M$};
  Dimension-Type.dim-to-typ [*1,2,0,0,0,3,0*];
  Dimension-Type.normalise @{*typ* $L^{-2}{\cdot}M^{-1}{\cdot}T^4{\cdot}I^2{\cdot}M$}›

the system output:

val it = [~*2*, *0*, *4*, *2*, *0*, *0*, *0*]: *int list*
val it = $L \cdot M^2 \cdot N^3$: **typ**
val it = $L^{-2} \cdot T^4 \cdot I^2$: **typ**

6 ISQ Quantity and SI Types

6.1 The Semantic Domain of Physical Quantities

Here, we give a semantic domain for particular values of physical quantities. A quantity is usually expressed as a number and a measurement unit, and the goal is to support this. First, though, we give a more general semantic domain where a quantity has a magnitude and a dimension.

record $('\alpha)$ *Quantity* =
  mag :: $'\alpha$ — Magnitude of the quantity.
  dim :: *Dimension* — Dimension of the quantity – denotes the kind of quantity.

The magnitude type is parametric as we permit the magnitude to be represented using any kind of numeric type, such as $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$, though we usually minimally expect a field.

By a number of class instantiations, we lift the type $'\alpha$ *Quantity* into the class *comm-monoid-mult*, provided that the magnitude is of that class. The following homomorphisms hold:

lemma *mag-times* [*simp*]: $mag\ (x \cdot y) = mag\ x \cdot mag\ y$ <**proof**>
lemma *dim-times* [*simp*]: $dim\ (x \cdot y) = dim\ x \cdot dim\ y$ <**proof**>
lemma *mag-inverse* [*simp*]: $mag\ (inverse\ x) = inverse\ (mag\ x)$ <**proof**>
lemma *dim-inverse* [*simp*]: $dim\ (inverse\ x) = inverse\ (dim\ x)$ <**proof**>

record $('\alpha,\ 's{::}unit\text{-}system)$ *Measurement-System* = $('\alpha)$ *Quantity* +
  unit-sys :: $'s$ — The system of units being employed

where *unit-system* again forces the carrier-set of its instances to have cardinality 1.

6.2 Dimension Typed Measurement Systems

We can now define the type of parameterized quantities $'\alpha['d,\ 's]$ by $('\alpha,\ 's)$ *Measurement-System*'s, which have a dimension equal to the semantic interpretation of the $'d$:

typedef (**overloaded**) $('\alpha,\ 'd{::}dim\text{-}type,\ 's{::}unit\text{-}system)$ *QuantT* (-[-, -] [*999*,*0*,*0*] *999*)
  $= \{x :: ('\alpha,\ 's)\ Measurement\text{-}System.\ dim\ x = dim\text{-}ty\text{-}sem\ TYPE('d)\}$
  morphisms *fromQ toQ* <*non−emptyness* **proof** *omitted*>

setup-lifting *type-definition-QuantT*

where $'s$ is a tag-type characterizing the concrete measuring system (e. g., SI, BIS, UCS, ...). Via the class *unit-system* these tag-types are again restricted to carrier-sets of cardinality 1.

Intuitively, the term x can be read as "x is a quantity with magnitude of type $'\alpha$, dimension type $'d$, and measured in system $'s$.

6.3 Operators in Typed Quantities

We define several operators on typed quantities. These variously compose the dimension types as well. Multiplication composes the two dimension types. Inverse constructs and inverted dimension type. Division is defined in terms of multiplication and inverse.

lift-definition
qtimes :: $('\alpha{::}\textit{comm-ring-1})['\tau 1{::}\textit{dim-type},\ 's{::}\textit{unit-system}]$
$\Rightarrow\ '\alpha['\tau 2{::}\textit{dim-type},\ 's] \Rightarrow\ '\alpha['\tau 1 \cdot '\tau 2,\ 's]$ (**infixl** $\cdot$ *69*)
is $(*)$ **by** (*simp add: dim-ty-sem-DimTimes-def times-Quantity-ext-def*)

lift-definition
qinverse :: $('\alpha{::}\textit{field})['\tau{::}\textit{dim-type},\ 's{::}\textit{unit-system}] \Rightarrow\ '\alpha['\tau^{-1},\ 's]$ $((\text{-}^{-\mathbf{1}})\ [999]\ 999)$
is *inverse* **by** (*simp add: inverse-Quantity-ext-def dim-ty-sem-DimInv-def*)

Additionally, a scalar product $(*_Q)$ and an addition on the magnitude component is introduced that preserves the algebraic properties of the magnitude type:

lift-definition *scaleQ*
:: $'\alpha \Rightarrow\ '\alpha{::}\textit{comm-ring-1}['d{::}\textit{dim-type},\ 's{::}\textit{unit-system}] \Rightarrow\ '\alpha['d,\ 's]$ (**infixr** $*_Q$ *63*)
is $\lambda\ r\ x.\ (\!|\ \textit{mag} = r * \textit{mag}\ x,\ \textit{dim} = \textit{dim-ty-sem}\ \textit{TYPE}('d),\ \textit{unit-sys} = \textit{unit}\ |\!)$ <**proof**>

instantiation *QuantT* :: (*plus*, *dim-type*, *unit-system*) *plus*
begin
lift-definition *plus-QuantT* :: $'\alpha['d,\ 's] \Rightarrow\ '\alpha['d,\ 's] \Rightarrow\ '\alpha['d,\ 's]$
is $\lambda\ x\ y.\ (\!|\ \textit{mag} = \textit{mag}\ x + \textit{mag}\ y,\ \textit{dim} = \textit{dim-ty-sem}\ \textit{TYPE}('d),\ \textit{unit-sys} = \textit{unit}\ |\!)$ <**proof**>
instance ..
end

6.4 Predicates on Typed Quantities

The standard HOL order $(\leq)$ and equality $(=)$ have the homogeneous type $'a \Rightarrow\ 'a \Rightarrow \textit{bool}$ and so they cannot compare values of different types. Consequently, we define a heterogeneous order and equivalence on typed quantities. Both operations were defined as lifting of the core operations.

lift-definition *qless-eq* :: $'\alpha{::}\textit{order}['d{::}\textit{dim-type},'s{::}\textit{unit-system}] \Rightarrow\ '\alpha['d{::}\textit{dim-type},'s] \Rightarrow \textit{bool}$
(**infix** $\lesssim_Q$ *50*) **is** $(\leq)$ **.**

lift-definition *qequiv* :: $'\alpha['d_1{::}\textit{dim-type},\ 's{::}\textit{unit-system}] \Rightarrow\ '\alpha['d_2{::}\textit{dim-type},\ 's] \Rightarrow \textit{bool}$
(**infix** $\cong_Q$ *50*) **is** $(=)$ **.**

These are both fundamentally the same as the usual order and equality relations, but they permit potentially different dimension types, *'d1* and *'d2*. Two typed quantities are comparable only when the two dimension types have the same semantic dimension.

The equivalence properties on $(\cong_Q)$ hold and even a restricted form of congruence inside the *dim-type*'s can be established.

6.5 SI as Typed Quantities

It is now straight-forward to define an appropriate tag-type *SI* and to introduce appropriate syntactic abbreviations that identify the type $'\alpha[m]$ with $'\alpha[m]$, $'\alpha[kg]$ with $'\alpha[kg]$, $'\alpha[s]$ with $'\alpha[s]$, etc, i. e. the standard's symbols for measurements in the 'système international des mesurements' (SI). Since these are just syntactic shortcuts, all operations and derived propertied in this section also apply to the SI system, as well as equivalent presentations of the British Imperial System (BIS) or the US-customer system (UCS). If needed, type-safe conversion operations between these systems can be defined, whose precision will depend on the underlying magnitude types, however.

7 Validation by the VIM and the 'Brochure'

8 Related Work and Conclusion

This work has drawn inspiration from some previous formalisations of the ISQ and SI, notably Hayes and Mahoney's formalisation in Z[4] and Aragon's algebraic structure for physical quantities[1]. To the best of our knowledge, our mechanisation represents the most comprehensive account of ISQ and SI in a theory prover.

References

[1] S. Aragon. The algebraic structure of physical quantities. *Journal of Mathematical Chemistry*, 31(1), May 2004.

[2] Bureau International des Poids et Mesures and Joint Committee for Guides in Metrology. Basic and general concepts and associated terms (vim) (3rd ed.). Technical report, BIPM, JCGM, 2012. Version 2008 with minor corrections.

[3] Bureau International des Poids et Mesures and Joint Committee for Guides in Metrology. The International System of Units (SI). Technical report, BIPM, JCGM, 2019. 9th edition.

[4] I. J. Hayes and B. P. Mahony. Using units of measurement in formal specifications. *Formal Aspects of Computing*, 7(3):329–347, 1995. doi: 10.1007/BF01211077. URL https://doi.org/10.1007/BF01211077.

[5] T. Nipkow and C. Prehofer. Type reconstruction for type classes. *J. Funct. Program.*, 5(2):201–224, 1995. doi: 10.1017/S0956796800001325. URL https://doi.org/10.1017/S0956796800001325.

[6] T. Nipkow and G. Snelting. Type classes and overloading resolution via order-sorted unification. In J. Hughes, editor, *Functional Programming Languages and Computer Architecture, 5th ACM Conference, Cambridge, MA, USA, August 26-30, 1991, Proceedings*, volume 523 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 1991. doi: 10.1007/3540543961_1. URL https://doi.org/10.1007/3540543961_1.

[7] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. doi: 10.1007/3-540-45949-9.