

This is a repository copy of *Automated Reasoning for Physical Quantities, Units, and Measurements in Isabelle/HOL*.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/198263/>

Version: Accepted Version

Proceedings Paper:

Foster, Simon David orcid.org/0000-0002-9889-9514 and Wolff, Burkhard (Accepted: 2023)
Automated Reasoning for Physical Quantities, Units, and Measurements in Isabelle/HOL.
In: The 27th International Conference on Engineering of Complex Computer Systems
(ICECCS 2023). IEEE. (In Press)


Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Automated Reasoning for Physical Quantities, Units, and Measurements in Isabelle/HOL

Simon Foster 
University of York, UK
simon.foster@york.ac.uk

Burkhard Wolff 
Université Paris-Saclay, France
wolff@lri.fr

Abstract—Formal verification of cyber-physical systems requires that we can accurately model physical quantities. SI units allow a higher degree of rigour, since we can ensure compatibility of quantities in calculations. In this paper, we contribute a mechanisation of the International System of Quantities (ISQ) and the SI unit system in Isabelle/HOL. We show how Isabelle can be used to provide a type system for physical quantities, and automated proof support. Quantities are parameterised by *dimension types* and so only quantities of the same dimension can be equated. 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).

I. INTRODUCTION

The International System of Quantities [1] (ISQ) is a standard for the definition of physical quantities, units, and measurement. It defines seven base quantities, such as mass, length, time, and current, along with several derived quantities, such as velocity, force, and energy. Each quantity has a corresponding unit of measurement, defined in the International System of Units [2] (SI). This is a coherent system of units, built on seven base units (metre, kilogram, second, etc.), 22 derived units (farad, lumen, watt, etc.), and 24 prefixes.

With the advent of cyber-physical and robotic systems, software needs to account for the physical environment in which a system operates and sensed measurements [3]. The integration of physical quantities into software engineering is therefore important to ensure that physical equations, models, and program code respect the physical units [4], [5]. As an example, a recent industrial case study on a certification of an autonomous underwater vehicle [6] develops a model-based assurance case, which includes system-level specifications and physical properties such as its dimensions and velocity.

The contribution of this paper is a mechanisation of the ISQ and SI in Isabelle/HOL, through a deep integration into Isabelle’s polymorphic type system [7]. Our aim is two-fold: (1) to provide a coherent mechanisation of the ISQ and its ontology of units; and (2) to support the use of units in formal models for cyber-physical systems [6], [3]. Our treatment of physical quantities allows the use of Isabelle’s type system in checking for correct use of units and reasoning about physical quantities. Our construction is validated by a test-set of known equivalences between both quantities and SI

units [2]. Moreover, the presented theory can be used for type-safe conversions between the SI and non-metric systems.

We introduce a parametric quantity type, $N[\mathcal{D}, S]$, where N is a numeric type, \mathcal{D} is a dimension type (e.g. L , M , T), and S is the system of units. We can specify quantities like 20 m with type $\mathbb{R}[L, SI]$, which is 20 metres in the SI (dimension length), and 30 lb of type $\mathbb{R}[M, BIS]$, which is 30 pounds in the BIS (dimension mass). Only quantities of the same dimension and unit system are comparable, and so “20 m = 30 lb” is a type error. We can also convert between different systems using function *metrify*, such that *metrify*(30 lb) \approx 9.07 kg.

In summary, our contributions are: (1) an embedding of the ISQ into Isabelle/HOL, including dimensions, quantities, units, and conversions; (2) a sound-by-construction quantity type system that checks dimensions and employs dimension coercions; (3) automated proof support for theorems involving quantities; (4) a formal ontology of units from the VIM [1] and SI Brochure [2], for use in specifications and models.

The structure of our paper is as follows. In §II we briefly survey related work. In §III we begin our contributions with dimension types. In §IV we use dimensions to implement the quantity type. In §V, we implement the SI unit system, and an associated ontology of units and equations. In §VI we describe conversions between unit systems. Finally, in §VII we conclude. Our Isabelle development can be found on the Archive of Formal Proofs [8].

II. RELATED WORK

The need for physical quantities and measurement in software engineering is widely acknowledged [9], [4], [5]. Quantity types are implemented in systems like MATLAB and Mathematica to support conversion between units, checking for unit consistency, and simplification of dimensions. There have also been numerous direct implementations of ISQ and SI for programming languages. Dimension Types have been presented by Kennedy [10], [11] for F^\sharp , and a more recent account is by Garrigue and Ly [12]. These works directly implement a type system for dimensions and units, while our approach formally derives such type inference inside HOL’s parametric polymorphism. In contrast to direct implementations, our approach assures correctness by construction.

Hayes et al. [9] extend the Z notation with an operation $M \odot D$ for a quantity of numeric type M and dimension type D , which has served as inspiration for our approach.

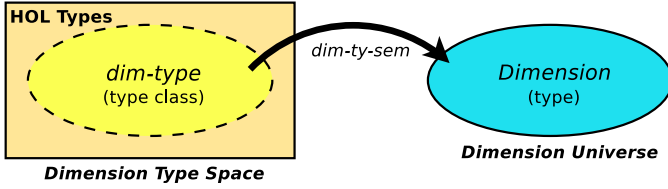


Fig. 1. Mapping dimension types into the dimension universe

Similarly, Gibson et al. [13] extend Event-B with dimensions, measurements, and unit conversions via a dimension universe construction. Aragon [14] formalises quantities as ordered pairs called q -numbers, consisting of a complex number and a unit label, and explores their algebraic structures. His algebraic properties have served as a benchmark.

Our work provides an implementation of the ISQ that is foundational, in that we precisely implement the quantity calculus, but also applicable, because it permits automatic type checking of dimensions, efficient proof support, and code generation [15] to provide a verified baseline implementation.

III. DIMENSIONS

Dimensions differentiate quantities of different kinds. Quantities 10m and 10kg have the same magnitude, but are incomparable since they have the dimensions of length and mass, respectively. The seven base dimensions of the ISQ are each denoted by a symbol, such as L , M , T etc., and a dimension is a product of symbols, each raised to an integer power. For example, the area quantity has dimension L^2 , and the velocity quantity $L \cdot T^{-1}$. Here, we support a generic dimension system based on vectors for different unit systems.

In a type-theoretic context, we can parametrise a quantity by its dimension type. Our approach is shown in Figure 1. We define (1) a universe for dimensions; (2) a type class (*dim-type*) to syntactically characterise dimension types that can be injected into the universe; (3) define a set of unitary types and type constructors that instantiate *dim-type*, and can parametrise quantities. Effectively, this gives an inductive definition for a family of types over the dimension arithmetic operators.

Universe of Dimensions. Assume there are $k \in \mathbb{N}$ base quantities. A dimension has the form $d_1^{x_1} \cdot d_2^{x_2} \cdots d_n^{x_n}$, a product of dimension symbols (d_i) each raised to a power drawn from the vector \mathbf{x} . We model dimension vectors in Isabelle using the type definition $(N, I) \text{dimvec} \triangleq (I \Rightarrow N)$, where I is a finite type isomorphic to $\{1..k\}$ and N is a suitable numeric type. In particular, we define ISQ dimensions with the type $\text{Dimension} \triangleq (\mathbb{Z}, \text{sdim}) \text{dimvec}$, where $\text{sdim} = \{\text{Length}, \text{Mass}, \text{Time}, \dots\}$.

We now define the dimension constructors. The null dimension $\mathbf{1} \triangleq (\lambda i. 0)$ is for dimensionless quantities, such as mathematical constants. Function $\mathbf{b}(i) \triangleq \mathbf{1}(i \mapsto 1)$, for $i \in I$, constructs a base dimension from the base quantity i . A base dimension has exactly one entry in the vector mapping to 1, with the others all 0. We also define a predicate $\text{is-BaseDim} :: (N, I) \text{dimvec} \Rightarrow \mathbb{B}$, for base dimensions.

A product of two dimensions, $\mathbf{x} \cdot \mathbf{y} \triangleq (\lambda i. \mathbf{x}(i) + \mathbf{y}(i))$ pointwise sums together all of the powers, and an inverse, $\mathbf{x}^{-1} \triangleq (\lambda i. -\mathbf{x}(i))$ negates each of the powers. We obtain

division as $\mathbf{x}/\mathbf{y} \triangleq \mathbf{x} \cdot \mathbf{y}^{-1}$. Then, whenever $(N, +, 0, -)$ forms an abelian group, so does $((N, I) \text{dimvec}, \cdot, \mathbf{1}, ^{-1})$. The group laws can be used to equationally rewrite dimension expressions using the simplifier.

Another avenue to efficient proof for dimensions is provided through the Isabelle code generator [15]. Since the set of base quantities I is enumerable, we can always convert a dimension vector to a list of N , and vice-versa. Moreover, each of the dimension operators can be represented as list operators, with which we can perform efficient dimension arithmetic.

Dimension Types. We first introduce a dimension type class:

```
class dim-type = unitary +
  fixes dim-ty-sem ::  $\mathcal{D}$  itself  $\Rightarrow$  Dimension
```

```
class basedim-type = dim-type +
  assumes is-BaseDim: is-BaseDim( $QD(\mathcal{D})$ )
```

A type class characterises a family of types that each implement a given function signature with certain properties, such as algebraic structures like monoids and groups. The **class** command introduces a type class with a given name, potentially extending existing classes. The **fixes** subcommand declares a new typed symbol in the signature, and **assumes** introduces a property of the symbols in the signature.

The *dim-type* class characterises a *unitary* type \mathcal{D} (i.e. with cardinality 1) associated with a dimension. *Ditself* represents a type as a value in Isabelle/HOL. Thus, *dim-ty-sem* is a function from types inhabiting *dim-type* to particular dimensions, as shown in Figure 1. We can use the syntactic constructor $\text{TYPE}(\alpha)$ to obtain a value of type α itself, for a particular type α . This effectively introduces an isomorphism between dimensions at the value level and the type level. We introduce the notation $QD(\mathcal{D}) \triangleq \text{dim-ty-sem } \text{TYPE}(\mathcal{D})$, which obtains the dimension of a given dimension type. The class *basedim-type* further specialises *dim-type* for base dimensions.

We use these classes to capture the dimension type constructors. The base dimension types are distinct unitary types, for example $\text{Length} \triangleq \{()\}^1$. We define such a type for each of the seven base quantities, along with a special type called **1** for dimensionless quantities. Each base dimension instantiates *basedim-type* by mapping to the corresponding dimension symbol, such that, for example, $QD(\text{Length}) = \text{Length}$.

Next, we introduce syntactic types to represent type-level dimension arithmetic. For product, we define a unitary type $(\mathcal{D}_1, \mathcal{D}_2) \text{DimTimes}$, whose parameters \mathcal{D}_1 and \mathcal{D}_2 inhabit the *dim-type* class. For inverse, we define $(\mathcal{D}) \text{DimInv}$, with \mathcal{D} in *dim-type*. We assign these *dim-ty-sem* implementations:

$$\begin{aligned} \text{dim-ty-sem}(d :: (\mathcal{D}_1, \mathcal{D}_2) \text{DimTimes}) &= QD(\mathcal{D}_1) \cdot QD(\mathcal{D}_2) \\ \text{dim-ty-sem}(d :: (\mathcal{D}) \text{DimInv}) &= QD(\mathcal{D})^{-1} \end{aligned}$$

The semantics of a *DimTimes* type calculates the underlying value-level dimension of each parameter, \mathcal{D}_1 and \mathcal{D}_2 , and multiplies them together. The *DimInv* type calculates the dimension and then takes the inverse. We provide mathematical

¹Types have a different namespace to definitions, so there is no clash.

syntax, so that we can write dimension types like $\mathbf{M} \cdot \mathbf{L}$ and \mathbf{T}^{-1} . We also define a type synonym for division, namely $(\mathcal{D}_1, \mathcal{D}_2) \text{DimDiv} \triangleq \mathcal{D}_1 \cdot \mathcal{D}_2^{-1}$, and give it the usual syntax. Moreover, we define a fixed number of powers and inverse powers at the type level, such as $\mathcal{D}^{-3} = (\mathcal{D} \cdot \mathcal{D} \cdot \mathcal{D})^{-1}$.

We can now also create the set of derived dimensions specified in the ISQ using type synonyms. For example, $\text{Velocity} \triangleq \mathbf{L} \cdot \mathbf{T}^{-1}$ and $\text{Pressure} \triangleq \mathbf{L}^{-1} \cdot \mathbf{M} \cdot \mathbf{T}^{-1}$, which provides a terminology of dimensions for use in formal specifications. We show further examples below, which also demonstrates the mathematical syntax for dimensions in Isabelle/HOL:

```
type_synonym Energy = "L2·M·T-2"
type_synonym Power = "L2·M·T-3"
type_synonym Force = "L·M·T-2"
type_synonym Pressure = "L-1·M·T-2"
type_synonym Charge = "I·T"
```

Dimension Normalisation. Dimension types with different syntactic forms are incomparable, because they have distinct type expressions. For example, whilst intuitively $\mathbf{L} \cdot \mathbf{T}^{-1} \cdot \mathbf{T} = \mathbf{L}$, these two expressions are different, and no built-in normalisation is available. As a result, we implement our own normalisation function, $\text{normalise}(\mathcal{D})$, for ISQ dimensions in Isabelle/ML, so that quantities over dimensions with distinct syntax can be related. This evaluates the vector of a dimension expression, and then uses this to produce a normal form.

We implement $\text{typ-to-dim} :: \text{typ} \Rightarrow \text{int list}$, which evaluates types formed of base dimensions and arithmetic operators. For example, $\text{typ-to-dim}(\mathbf{L}) = [1, 0, 0, 0, 0, 0, 0]$ and $\text{typ-to-dim}(\mathcal{D}^{-1}) = \text{map}(\lambda x. -x) (\text{typ-to-dim}(\mathcal{D}))$. Following evaluation, we can construct a normal form as an ordered dimension expression: $\mathbf{L}^{x_1} \cdot \mathbf{M}^{x_2} \cdot \mathbf{T}^{x_3} \dots \mathbf{J}^{x_7}$, with terms where $x_i = 0$ omitted. If every term is 0, then the function produces the dimensionless quantity, $\mathbf{1}$. For example, $\text{normalise}(\mathbf{T}^4 \cdot \mathbf{L}^{-2} \cdot \mathbf{M}^{-1} \cdot \mathbf{I}^2 \cdot \mathbf{M})$ yields the dimension type $\mathbf{L}^{-2} \cdot \mathbf{T}^4 \cdot \mathbf{I}^2$.

IV. PHYSICAL QUANTITIES AND MEASUREMENT

As for dimensions, we model quantities at both the value and type level, through a universe construction.

Quantity Universe. We specify our quantity universe as a record type $(N, I) \text{Quantity}$ by pairing a magnitude of type N with a dimension vector of type $(\text{int}, I) \text{dimvec}$. N is a suitable numeric type (e.g. \mathbb{Q} , \mathbb{R}), and I is the dimension index. For example, we define the zero and one quantities as $0 \triangleq (0, \mathbf{1})$ and $1 \triangleq (1, \mathbf{1})$, which are both dimensionless. We define functions $\text{mag} :: (N, I) \text{Quantity} \Rightarrow N$ and $\text{dim} :: (N, I) \text{Quantity} \Rightarrow I$, which extract resp. the magnitude and dimension. The arithmetic operators are defined as below:

$$\begin{aligned} (x, \mathcal{D}_1) \cdot (y, \mathcal{D}_2) &= (x \cdot y, \mathcal{D}_1 \cdot \mathcal{D}_2) \\ (x, \mathcal{D})^{-1} &= (x^{-1}, \mathcal{D}^{-1}) \\ (x, \mathcal{D}_1) / (y, \mathcal{D}_2) &= (x / y, \mathcal{D}_1 / \mathcal{D}_2) \\ (x, \mathcal{D}) + (y, \mathcal{D}) &= (x + y, \mathcal{D}) \\ (x, \mathcal{D}) - (y, \mathcal{D}) &= (x - y, \mathcal{D}) \\ (x, \mathcal{D}_1) \leq (y, \mathcal{D}_2) &\Leftrightarrow (x \leq y \wedge \mathcal{D}_1 = \mathcal{D}_2) \end{aligned}$$

The arithmetic operators are overloaded, and so can appear on both sides of these equations. Multiplication, inverse, and division are total operations that distribute through the pair. When multiplying two quantities, we need to multiply both the magnitudes and dimensions. For example, $(7, \mathbf{L} \cdot \mathbf{T}^{-1}) \cdot (2, \mathbf{T}) = (14, \mathbf{L})$. Addition and subtraction are specified only when the two quantities have the same dimension. Finally, the order on quantities is simply the order on the magnitudes, but with the requirement that the two dimensions are equal.

Measurement Systems. We extend the *Quantity* type to create “measurement systems”, which additionally specify the system of units being employed. A measurement system, with type $(N, I, S) \text{Measurement-System}$, is a quantity that specifies the system of units via type parameter S , which inhabits the type class *unit-system*. This extra parameter allows us to distinguish quantities using different systems of units, and so prevent improper mixing. For example, the presence of the *SI* tag means that length is measured in metres, whereas the presence of a tag such as *BIS* may indicate that yards is used. This also facilitates type-safe conversion between different systems. All the arithmetic operators are lifted to measurement systems. Since all such functions are monomorphic (e.g. of type $\alpha \Rightarrow \alpha \Rightarrow \alpha$), mixing of systems is avoided by construction.

Dimension Typed Quantities. Having defined our universe for quantities, we next enrich this representation with type-level dimensions. For expediency, we assume that all such quantities also have a measurement system attached. Moreover, we focus on quantities with dimensions from the ISQ.

We define a type $(N, \mathcal{D}, S) \text{QuantT}$ with characteristic set

$$\{x :: (N, \text{sdim}, S) \text{Measurement-System}. \text{dim}(x) = QD(\mathcal{D})\}$$

which is the set of ISQ quantities x whose dimension ($\text{dim}(x)$) agrees with the one in the dimension type \mathcal{D} . For convenience, we introduce the syntax $N[\mathcal{D}, S]$ for $(N, \mathcal{D}, S) \text{QuantT}$.

Lifting of operators $x+y$ and $x-y$ is straightforward for typed quantities, since they are monomorphic and only defined when the dimensions of x and y agree. We can then easily show that typed quantities form an additive abelian group. We also define a scalar multiplication $\text{scaleQ} :: N \Rightarrow N[\mathcal{D}, S] \Rightarrow N[\mathcal{D}, S]$, with notation $n *_{\mathbf{Q}} x$, which scales a quantity by a given number without changing the dimension. We can then show that typed quantities form an additive abelian group, and a real vector space, with $(*_\mathbf{Q})$ as the scalar multiplication operator.

Things are more involved when dealing with multiplication and division, since these need to perform type-level dimension arithmetic. For example, if we have quantities $x :: \mathbb{R}[\mathbf{I}, SI]$ and $y :: \mathbb{R}[\mathbf{T}, SI]$, then multiplication of x and y is well-defined, and should have the type $\mathbb{R}[\mathbf{I} \cdot \mathbf{T}, SI]$. As a result, we introduce bespoke functions for these operations:

$$\begin{aligned} qtimes &:: N[\mathcal{D}_1, S] \Rightarrow N[\mathcal{D}_2, S] \Rightarrow N[\mathcal{D}_1 \cdot \mathcal{D}_2, S] \\ qinverse &:: N[\mathcal{D}, S] \Rightarrow N[\mathcal{D}^{-1}, S] \\ qdivide &:: N[\mathcal{D}_1, S] \Rightarrow N[\mathcal{D}_2, S] \Rightarrow N[\mathcal{D}_1 / \mathcal{D}_2, S] \end{aligned}$$

Function *qtimes* multiplies two quantities, with the same measurement system, and “multiplies” the dimension types

using the type constructors in §III. Technically, no multiplication computation takes place, but rather a type constructor is inserted. Similarly, *qinverse* represents the inverse of the parametrised dimension, and *qdivide* is division. What is achieved here is analogous to dependent types, though with additional machinery for normalising dimension types.

The definitions of *qtimes* and *qinverse* are obtained by lifting of the underlying quantity operators [16]. We need to prove that the invariant of the *QuantT* type is satisfied, which involves showing that the family of typed quantities is closed under the two functions. For *qmult*, we need to prove that $\dim(x \cdot y) = QD(\mathcal{D}_1 \cdot \mathcal{D}_2)$, whenever $\dim(x) = QD(\mathcal{D}_1)$ and $\dim(y) = QD(\mathcal{D}_2)$, which follows by definition. For convenience, we give these functions the usual notation of $x \bullet y$, x^{-1} , and x/y , but embolden the operators to syntactically distinguish them. With *qtimes* and *qinverse*, we can also define positive and negative powers, such as $x^{-2} = (x \bullet x)^{-1}$.

Equality ($x = y$) in HOL is a homogeneous function of type $\alpha \rightarrow \alpha \rightarrow \mathbb{B}$; therefore, it cannot be used to compare objects of different types. Consequently, it cannot be used to compare quantities whose dimension types have different syntactic forms (e.g. $\mathbf{L} \cdot \mathbf{T}^{-1} \cdot \mathbf{T}$ and \mathbf{L}). This motivates a definition of *heterogeneous (in)equality* for quantities:

$$\begin{aligned} qequiv &:: N[\mathcal{D}_1, S] \Rightarrow N[\mathcal{D}_2, S] \Rightarrow \mathbb{B} \\ qless-eq &:: N[\mathcal{D}_1, S] \Rightarrow N[\mathcal{D}_2, S] \Rightarrow \mathbb{B} \end{aligned}$$

These functions are defined by lifting the functions ($=$) and (\leq) on the underlying quantities. They ignore the dimension types, but the underlying dimensions must nevertheless be equal. We give these functions the notation $x \cong y$ and $x \lesssim y$, respectively. (\cong) forms an equivalence relation, and (\lesssim) forms a preorder. Moreover, (\cong) is a congruence relation for (\bullet), ($^{-1}$), and (\ast_0).

Proof Support. We implement an interpretation-based proof strategy for typed quantity (in)equalities. When proving HOL (in)equalities, it suffices to show (in)equality of the magnitudes as, owing to homogeneous nature of these relations, the dimensions are equal by construction. For our heterogeneous operators, we also need to prove that the two dimensions are equal, for example using group laws.

We supply a proof method called *si-simp*, which uses the simplifier to perform transfer, interpretation, and arithmetic rewriting. An additional method called *si-calc* also compiles dimension vectors using the code generator, and can thus efficiently prove dimension equalities. We can, for example, prove the following algebraic laws automatically:

$$a \ast_0 (x + y) = (a \ast_0 x) + (a \ast_0 y) \quad x \bullet y \cong y \bullet x \quad (x \bullet y)^{-1} \cong x^{-1} \bullet y^{-1}$$

Coercion and Normalisation. The need for heterogeneous quantity relations (\cong , \lesssim) can be avoided by the use of coercions to convert between syntactic representations of dimensions. We can use Isabelle’s sophisticated syntax and checking pipeline to normalise dimensions, and so automatically coerce quantities to a normal form. This improves usability, since the usual relations ($=$) and (\leq) can be used directly.

We implement a function *dnorm* $:: N[\mathcal{D}_1, S] \Rightarrow N[\mathcal{D}_2, S]$, which converts between quantities with distinct but equivalent dimension forms. It checks whether the source and target dimensions (\mathcal{D}_1 and \mathcal{D}_2) are the same. If so, then it performs the coercion by erasing the original type and instating \mathcal{D}_2 . Otherwise, it returns a valid quantity of the target dimension, but with magnitude 0. This is acceptable because we guard the use of *dnorm* with a dimension type-check to ensure $\mathcal{D}_1 = \mathcal{D}_2$. For example, if we have $x :: \mathbb{R}[\mathbf{L} \cdot \mathbf{T}^{-1} \cdot \mathbf{T}, SI]$, then we can use *dnorm*(x) $:: \mathbb{R}[\mathbf{L}, SI]$ to obtain a quantity with an equivalent dimension, since $QD(\mathbf{L} \cdot \mathbf{T}^{-1} \cdot \mathbf{T}) = QD(\mathbf{L})$. For two equivalent quantities $x \cong y$, we have it that *dnorm*(x) = *dnorm*(y).

Next, we extend Isabelle’s checking pipeline to allow dimension normalisation, so that \mathcal{D}_2 can be automatically calculated. We implement an ML function *check-quant*, which takes a term and enriches it with dimension information. Whenever it encounters an instance of *dnorm*(t), it extracts the type of t , which should be $N[\mathcal{D}, S]$. We then enrich the instance of *dnorm* to have the type $N[\mathcal{D}, S] \Rightarrow N[\text{normalise}(\mathcal{D}), S]$.

We insert *check-quant* into the term checking pipeline using API function `Syntax_Phases.term_check`, which adds a new checking phase. We add normalisation after type inference so that we can use the unnormalised dimension type expression as input to *check-quant*. The soundness of this transformation does not depend on the correctness of *normalise*, since if an incorrect dimension is calculated, *dnorm* returns 0.

V. UNIT SYSTEMS AND THE SI

An SI unit is a quantity in the ISQ with magnitude 1. A *base unit* in system \mathcal{S} is a unit with a base dimension. Base units are described by the predicate *is-base-unit* $:: N[\mathcal{D}, S] \Rightarrow \mathbb{B}$, defined as *is-base-unit*(x) $\triangleq (\text{mag}(x) = 1 \wedge \text{is-BaseDim}(x))$. We introduce the constructor *BUNIT*(\mathcal{D}, S), which constructs a base unit using the base dimension type \mathcal{D} in the system \mathcal{S} .

For the SI, we create a unitary type *SI*, and instantiate the *unit-system* class. We then define the base units of the SI:

$$\begin{aligned} \text{metre} &\triangleq \text{BUNIT}(\mathbf{L}, SI) & \text{kilogram} &\triangleq \text{BUNIT}(\mathbf{M}, SI) \\ \text{ampere} &\triangleq \text{BUNIT}(\mathbf{I}, SI) & \text{kelvin} &\triangleq \text{BUNIT}(\Theta, SI) \\ \text{mole} &\triangleq \text{BUNIT}(\mathbf{N}, SI) & \text{candela} &\triangleq \text{BUNIT}(\mathbf{J}, SI) \end{aligned}$$

Since the *second* is very often the unit of time, we make it a polymorphic base unit, so that it can exist in several systems. For convenience, we create type synonyms for specifying units at the type level, for example $N \text{ metre} \triangleq N[\text{Length}, SI]$.

We now can now express quantities with SI units. For example, $20 \ast_0 \text{ metre}$ is the *metre* unit scaled by 20, and has the inferred type of $\mathbb{R}[\mathbf{L}, SI]$. Compound units can also be expressed, such as $10 \ast_0 (\text{metre} \bullet \text{second}^{-1})$, which has inferred type $\mathbb{R}[\mathbf{L} \cdot \mathbf{T}^{-1}]$. Unit equations, such as $(\text{metre} \bullet \text{second}^{-1}) \bullet \text{second} \cong \text{metre}$, can be proved using the *si-calc* proof strategy:

Lemma "(metre · second⁻¹) · second \cong_0 metre"
by si_calc

Similarly, coercions can be used to prove conjectures such as *dnorm*(($5 \ast_0 (\text{metre}/\text{second})$) \bullet ($10 \ast_0 \text{second}$)) = $50 \ast_0 \text{metre}$.

We construct an ontology of derived units from the VIM and SI Brochure [2, page 137]; a selection is shown below:

$$\begin{aligned} \text{hertz} &\triangleq \text{second}^{-1} \\ \text{joule} &\triangleq \text{kilogram} \bullet \text{metre}^2 \bullet \text{second}^{-2} \\ \text{watt} &\triangleq \text{kilogram} \bullet \text{metre}^2 \bullet \text{second}^{-3} \\ \text{coulomb} &\triangleq \text{ampere} \bullet \text{second} \end{aligned}$$

Isabelle can infer the dimension type of each such unit, for example *watt* has the dimension $\mathbf{M} \cdot \mathbf{L}^2 \cdot \mathbf{T}^{-3}$.

The SI defines 24 prefixes, which can be used to scale SI units. We give a selection of these below:

$$\begin{aligned} \text{hecto} &\triangleq 10^2 & \text{kilo} &\triangleq 10^3 & \text{mega} &\triangleq 10^6 & \text{giga} &\triangleq 10^9 \\ \text{deci} &\triangleq 10^{-1} & \text{centi} &\triangleq 10^{-2} & \text{milli} &\triangleq 10^{-3} & \text{micro} &\triangleq 10^{-6} \end{aligned}$$

Prefixes are abstract numbers in N , which can be used to scale units. For example, we can write $40 \text{ }_{\text{q}}\text{milli }_{\text{q}}\text{metre}$.

The SI also has a notion of “accepted” units [2, page 145], which are quantities often used as units, but without a magnitude of 1. We give a selection of these below:

$$\begin{aligned} \text{minute} &\triangleq 60 \text{ }_{\text{q}}\text{second} & \text{hour} &\triangleq 60 \text{ }_{\text{q}}\text{minute} \\ \text{day} &\triangleq 24 \text{ }_{\text{q}}\text{hour} & \text{degree} &\triangleq (\pi/180) \text{ }_{\text{q}}\text{radian} \\ \text{litre} &\triangleq 1/1000 \text{ }_{\text{q}}\text{metre}^3 & \text{tonne} &= 10^3 \text{ }_{\text{q}}\text{kilogram} \end{aligned}$$

These quantities can readily be treated as units in our mechanisation, though the type does not reflect the unit. For example, the units *day*, *hour*, and *year* all have the dimension \mathbf{T} . We can prove unit equation theorems such as $1 \text{ }_{\text{q}}\text{hour} = 3600 \text{ }_{\text{q}}\text{second}$, $1 \text{ }_{\text{q}}\text{day} = 86400 \text{ }_{\text{q}}\text{second}$, and $1 \text{ }_{\text{q}}\text{hectare} = 1 \text{ }_{\text{q}}(\text{hecto }_{\text{q}}\text{metre})^2$ using *si-simp*, which can act as the basis for unit conversions. Similarly, we can use prefixes to express relations between derived quantities, such as $25 \text{ }_{\text{q}}\text{metre/second} = 90 \text{ }_{\text{q}}(\text{kilo }_{\text{q}}\text{metre})/\text{hour}$.

The SI units are defined in terms of exact values for 7 physical constants [2, page 127]. We define these in Isabelle:

$$\begin{aligned} \Delta\nu_{Cs} &= 9192631770 \text{ }_{\text{q}}\text{hertz} \\ \mathbf{c} &= 299792458 \text{ }_{\text{q}}(\text{metre} \bullet \text{second}^{-1}) \\ \mathbf{h} &= (6.62607015 \cdot 10^{-34}) \text{ }_{\text{q}}(\text{joule} \bullet \text{second}) \\ \mathbf{e} &= (1.602176634 \cdot 10^{-19}) \text{ }_{\text{q}}\text{coulomb} \\ \mathbf{k} &= (1.380649 \cdot 10^{-23}) \text{ }_{\text{q}}(\text{joule/kelvin}) \\ N_A &= 6.02214076 \cdot 10^{23} \text{ }_{\text{q}}(\text{mole}^{-1}) \\ K_{cd} &= 683 \text{ }_{\text{q}}(\text{lumen/watt}) \end{aligned}$$

$\Delta\nu_{Cs}$ is the hyperfine transition frequency of the caesium 133 atom. Constant \mathbf{c} is the speed of light in a vacuum, and \mathbf{h} is the Planck constant. Constant \mathbf{e} is the elementary charge, and \mathbf{k} is the Boltzmann constant. N_A is the Avagadro constant. K_{cd} is the luminous efficacy of monochromatic radiation of frequency $540 \cdot 10^{12} \text{ Hz}$. With these constants, we can arrange their equations to verify defining theorems for each unit:

$$\begin{aligned} \text{second} &\cong (9192631770 \text{ }_{\text{q}}\mathbf{1})/\Delta\nu_{Cs} \\ \text{metre} &\cong (\mathbf{c}/(299792458 \text{ }_{\text{q}}\mathbf{1})) \bullet \text{second} \\ \text{kilogram} &\cong (\mathbf{h}/(6.62607015 \cdot 10^{-34}) \text{ }_{\text{q}}\mathbf{1}) \bullet \text{metre}^{-2} \bullet \text{second} \end{aligned}$$

The *second* is equal to the duration of 9192631770 periods of the radiation of the ^{133}Cs atom. The *metre* is the length travelled by light in a period of $1/299792458$ seconds. For *kilogram*, the equation effectively defines the unit kg m s^{-1} , and then applies the unit $\text{m}^{-2} \text{ s}$ to obtain a quantity of dimension \mathbf{M} . Each equation is proved using *si-calc*, which serves to validate our implementation of the SI. Finally, we complete our ontology of derived units [2, page 137]:

$$\begin{aligned} \text{newton} &\triangleq \text{kilogram} \bullet \text{metre} \bullet \text{second}^{-2} \\ \text{pascal} &\triangleq \text{kilogram} \bullet \text{metre}^{-1} \bullet \text{second}^{-2} \\ \text{volt} &\triangleq \text{kilogram} \bullet \text{metre}^2 \bullet \text{second}^{-3} \bullet \text{ampere}^{-1} \end{aligned}$$

We can prove the corresponding unit equations, which show equivalences between SI units:

$$\begin{aligned} \text{joule} &\cong \text{newton} \bullet \text{metre} & \text{watt} &\cong \text{joule/second} \\ \text{volt} &= \text{watt/ampere} & \text{farad} &\cong \text{coulomb/volt} \end{aligned}$$

The remaining derived units are all mechanised.

Finally, temperature in the SI is defined in Kelvin, but degrees celcius is more usual. So, we define $T^\circ\text{C} \triangleq (T + 273.15) \text{ }_{\text{q}}\text{kelvin}$, where 273.15 is the freezing point of water.

VI. UNIT CONVERSIONS AND NON-SI SYSTEMS

Aside from the SI, other units systems remain in use, notably metric systems such as CGS (centimetre-gram-second), and imperial systems, including the United States Customary system (USC) and the British Imperial System (BIS). With our present system of quantities, we can already describe imperial units, in terms of the SI units, as shown below:

$$\begin{aligned} \text{yard} &\triangleq 0.9144 \text{ }_{\text{q}}\text{metre} & \text{mile} &\triangleq 1760 \text{ }_{\text{q}}\text{yard} \\ \text{pound} &\triangleq 0.4535937 \text{ }_{\text{q}}\text{kilogram} & \text{stone} &\triangleq 14 \text{ }_{\text{q}}\text{pound} \\ \text{pint} &\triangleq 0.56826125 \text{ }_{\text{q}}\text{litre} & \text{gallon} &\triangleq 8 \text{ }_{\text{q}}\text{pint} \end{aligned}$$

Here, we define the international yard and pound, units of length and mass, which have exact metric definitions, and several derived units, for use in SI quantities.

However, this masks an inherent problem with imperial units: they have several definitions depending on the context. Whilst the international yard is 0.9144 metres, the BIS and USC both have slightly different definitions. For precise measurements we must specify the unit system, and define unit conversions. Even for metric systems, it is sometimes desirable to use different units, such as in the CGS system, where centimetres and grams are used as base units.

A conversion schema, $\mathcal{S}_1 \Rightarrow_U \mathcal{S}_2$, is a 7-tuple of rational numbers each greater than zero that can convert quantities between systems \mathcal{S}_1 and \mathcal{S}_2 . Each rational number encodes a conversion factor for each of the dimensions of the ISQ. We define a quantity conversion function $qconv :: (\mathcal{S}_1 \Rightarrow_U \mathcal{S}_2) \Rightarrow N[\mathcal{D}, \mathcal{S}_1] \Rightarrow N[\mathcal{D}, \mathcal{S}_2]$, whose definition is below:

$$qconv_C(m, \mathbf{d}) = \left(\left(\prod_{1 \leq i \leq 7} C_i^{\mathbf{d}_i} \right) \cdot m, \mathbf{d} \right)$$

Given a quantity (m, \mathbf{d}) , and a conversion schema C , $qconv$ calculates the conversion factor for m by raising each element

of C to the corresponding dimension element \mathbf{d}_i . For example, if we wish to convert cubic (international) yards to cubic metres, then we first need the conversion factor from yards to metres, which is 0.9144. Then, we take this value and raise it to the power of 3, and so the overall conversion factor is 0.764555. The dimension is unchanged by this operation.

The BIS is a non-metric standard for weights and measures. We model the BIS by creation of a unit system with the type BIS , and define $yard \triangleq BUNIT(L, BIS)$ and $pound \triangleq BUNIT(M, BIS)$. Moreover, we can create derived units such as $foot \triangleq 1/3 *_0 yard$ and $inch \triangleq 1/12 *_0 foot$. Then, we can formally specify quantities measured according to the BIS.

We can convert between the SI and BIS with schema $BSI :: BIS \Rightarrow_U SI$. The factors for length and mass required for this conversion are 0.9143993 and 0.453592338, respectively. We can then, for example, convert a BIS quantity of 1 ounce to grams using the conversion $qconv_{BSI}(1 *_0 ounce) \approx 37.8 *_0 gram^2$. We also create unit systems for the UCS and CGS systems, with suitable conversion factors.

Whilst we can use quantity conversions between systems directly, it is often more convenient to use the SI as a frame of reference for different unit systems. We therefore create a type class to representation metrification:

```
class metrifiable = unit-system+
fixes convschema ::  $S \text{ itself} \Rightarrow (S \Rightarrow_U SI)$ 
```

A unit system S is metrifiable if there is a conversion schema from S to SI . Consequently, the BIS, UCS, CGS, and the SI itself are all metrifiable. Consequently, for any pair of metrifiable systems, S_1 and S_2 , we define a generic conversion function $QMC_{S_1 \rightarrow S_2} :: N[\mathcal{D}, S_1] \Rightarrow N[\mathcal{D}, S_2]$, which performs conversion via metrification. This function first uses the conversion schema for S_1 to convert to the SI system, and then uses the inverse schema for S_2 to convert from SI to S_2 . For example, we can show that $QMC_{CGS \rightarrow BIS}(12 *_0 centimetre) \approx 4.724 *_0 inch$. We can therefore use the Isabelle type system to precisely specify what system a measurement is made in, and seamlessly convert between a variety of other systems.

VII. CONCLUSIONS

We have presented a mechanisation of the ISQ in Isabelle/HOL, which allows us to precisely specify physical dimensions, units, and unit systems. We use Isabelle's type system to ensure that only measurements of the same dimension and unit system can be combined. We have presented a substantial theory development of about 2500 lines of definitions and proofs that captures the ISQ and SI as defined in the VIM [1]. Our type system for physical quantities is, by construction, sound and complete. We provided a validation of our theory by checking the mandatory definitions and corollaries from the VIM and the SI Brochure [2]. An earlier version of our implementation was also applied in an industrial case study on a formal model for an autonomous underwater vehicle [6], which provides further validation.

²We use exact rational arithmetic for this in Isabelle/HOL, but we present an approximate decimal expansion for ease of comprehension.

There are a number of directions for future work. The current approach to handling dimension mismatches using coercions could be better automated by using the coercive subtyping mechanism [17]. This effectively extends the type inference algorithm so that type mismatches can be automatically resolved by insertion of registered coercion functions. At the same time, our approach to characterising dimension types, illustrated in Figure 1, is not specific to the ISQ, and could be generalised to other problems that are typically solved with dependent types. For example, we could normalise type expressions containing arithmetic operators to relate vectors parametrised by the length. Therefore, in future work we will investigate a generic approach using universe constructions to justify type-level functions and coercions.

REFERENCES

- [1] Bureau International des Poids et Mesures and Joint Committee for Guides in Metrology, "Basic and general concepts and associated terms (VIM) (3rd ed.)." BIPM, JCGM, Tech. Rep., 2012.
- [2] —, "The International System of Units (SI)," BIPM, JCGM, Tech. Rep., 2019, 9th edition.
- [3] A. Cavalcanti, J. Baxter, and G. Carvalho, "RoboWorld: Where can my robot work?" in *Software Engineering and Formal Methods (SEFM)*, ser. LNCS, vol. 13085. Springer, 2021, pp. 3–22.
- [4] B. D. Hall, "Software for calculation with physical quantities," in *2020 IEEE International Workshop on Metrology for Industry 4.0 & IoT*, 2020, pp. 458–463.
- [5] D. Flater, "A system of quantities from software metrology," *Measurement*, vol. 168, 2021.
- [6] S. Foster, Y. Nemouchi, C. O'Halloran, N. Tudor, and K. Stephenson, "Formal model-based assurance cases in Isabelle/SACM: An autonomous underwater vehicle case study," in *Proc. 8th Intl. Conf. on Formal Methods in Software Engineering (FormalISE)*. ACM, 2020.
- [7] T. Nipkow and G. Snelting, "Type classes and overloading resolution via order-sorted unification," in *Functional Programming Languages and Computer Architecture, 5th ACM Conference*, ser. LNCS, vol. 523. Springer, 1991, pp. 1–14.
- [8] S. Foster and B. Wolff, "A sound type system for physical quantities, units, and measurements," *Archive of Formal Proofs*, October 2020, https://isa-afp.org/entries/Physical_Quantities.html.
- [9] I. J. Hayes and B. P. Mahony, "Using units of measurement in formal specifications," *Formal Aspects of Computing*, vol. 7, no. 3, pp. 329–347, 1995. [Online]. Available: <https://doi.org/10.1007/BF01211077>
- [10] A. Kennedy, "Dimension types," in *Programming Languages and Systems — ESOP '94*, D. Sannella, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 348–362.
- [11] —, "Types for units-of-measure: Theory and practice," in *Central European Functional Programming School – Third Summer School (CEFP 2009)*, ser. LNCS, Z. Horváth, R. Plasmeijer, and V. Zsóok, Eds., vol. 6299. Springer, 2009, pp. 268–305.
- [12] J. Garrigue and D. Ly, "Des unités dans le typeur," in *28ièmes Journées Francophones des Langages Applicatifs*, Gourette, France, Jan. 2017. [Online]. Available: <https://hal.archives-ouvertes.fr/hal-01503084>
- [13] J. P. Gibson and D. Méry, "Explicit modelling of physical measures: From Event-B to Java," in *Proc. Workshop on Handling IMPLICIT and EXPLICIT knowledge in formal system development (IMPEX)*, ser. EPTCS, R. Laleau, D. Méry, S. Nakajima, and E. Troubitsyna, Eds., vol. 271, 2017, pp. 64–79.
- [14] S. Aragon, "The algebraic structure of physical quantities," *Journal of Mathematical Chemistry*, vol. 31, no. 1, May 2004.
- [15] F. Haftmann and T. Nipkow, "Code generation via higher-order rewrite systems," in *10th Intl. Symp. on Functional and Logic Programming (FLOPS)*, ser. LNCS, vol. 6009. Springer, 2010, pp. 103–117.
- [16] B. Huffman and O. Kuncar, "Lifting and transfer: A modular design for quotients in Isabelle/HOL," in *CPP 2013*, ser. LNCS, vol. 8307. Springer, 2013, pp. 131–146.
- [17] D. Traytel, S. Berghofer, and T. Nipkow, "Extending hindley-milner type inference with coercive structural subtyping," in *APLAS 2011*, ser. LNCS, vol. 7078. Springer, 2011, pp. 89–104.