Types, equations, dimensions and the Pi theorem

Types, equations, dimensions and the Pi theorem

NICOLA BOTTA

Potsdam Institute for Climate Impact Research, Germany and Chalmers University of Technology and University of Gothenburg, Sweden
(email: botta@pik-potsdam.de*)*

PATRIK JANSSON

Chalmers University of Technology and University of Gothenburg, Sweden
(email: patrikj@chalmers.se*)*

Abstract

The languages of mathematical physics and modelling are endowed with a rich "grammar of dimensions" that common abstractions of programming languages fail to represent. We propose a dependently typed domain-specific language (embedded in Idris) that captures this grammar. We apply it to formalize basic notions of dimensional analysis: those of dimension function, physical quantity, homomorphic measurement, the covariance principle and Buckingham's Pi theorem. We hope that the language makes mathematical physics more accessible to computer scientists and functional programming more palatable to modellers and physicists.

1 Introduction

Motivation. The main motivation for this work comes from a failure. During more than one decade, the authors have been advocating mathematical specifications, type-driven analysis and functional programming (FP) as methodologies to better understand, specify and solve problems in climate impact research, climate policy advice and, by large, global systems science [8, 11–13, 30, 32].

Alas, after ten years of advocacy and intense collaborations, we have hardly been able to convince any climate modeller of the usefulness of FP, let alone convert them to "thinking functionally" with Haskell [7], Agda [44], Idris [14], or Rocq [55]. Have we just done a bad job or is this failure a symptom of a deeper problem?

There is no need for FP in mathematical physics, is there? Physicists and modellers are well trained in exploiting established numerical libraries [24, 53, 54] and frameworks [45, 63] for approximating solutions of (ordinary, partial, stochastic) differential equations efficiently, implementing large computer-based models in FORTRAN, C, C++, Java, Python, or Julia and testing model predictions against analytical solutions or observations.

In engineering and in many physical sciences, this methodology has led to reliable computations and to computer-based modelling almost fully replacing physical modelling: for many applications, running computer programs is more flexible and much cheaper than running wind tunnels or full-scale experiments.

But there are important research areas in which empirical validations are beyond reach and the predictive capability of computer-based models is poorly known and needs to be questioned. In climate science but also in plasma physics, for example, it is simply impossible (or just too dangerous or too expensive) to test the correctness of computations empirically.

Journal of Functional Programming, Volume 36, Article 3 DOI: 10.46298/jfp.17762
Submitted August 2023; accepted March 2026

It is not possible to study the effectiveness of a policy designed to reduce greenhouse gas (GHG) emissions without implementing that policy; or, as argued by Lucarini [39], "the usual Galilean scientific validation criteria do not apply to climate science". In much the same way, plasma physicists and engineers cannot afford to damage hundreds of experimental tokamak fusion reactors to assess and validate optimal control options for such devices [27, 47].

In these domains, scientists need methodologies that bring confidence that their computations are correct well before such computations can actually be applied to real world systems. Formal specification is the key to both testing programs and showing the *absence* of errors in computations [30] and dependently typed FP languages have reached enough expressive power to support formulating very precise specifications. And yet climate modellers and physicists have, by and large, stayed away from FP languages. Why so?

Educational gaps. It is probably fair to say that most physicists and modellers have never heard of mathematical program specifications, not to mention FP and dependently typed languages. In much the same way, most computer scientists have hardly been exposed to the language of mathematical physics, say, for example, that of Arnold [1], Barenblatt et al. [3], Courant and Hilbert [20], Kuznetsov [38].

Originally very close to elementary set theory and calculus, this language has evolved over the last decades and fragmented into a multitude of dialects or DSLs, driven perhaps by the pervasive usage of imperative programming and computer-based modelling. Common traits of these dialects are the limited usage of currying and higher order functions, the overloading of the equality sign, the lack of referential transparency and explicit type information (although mnemonic rules are often introduced for encoding such information like in $x_{[0,T]}$ instead of $x : [0, T] \to \mathbb{R}$) and the usage of parentheses to denote both function application and function composition as in $\dot{x} = f(x)$ instead of $\forall t,\ \dot{x}(t) = f(x(t))$ or, in point-free notation, $\dot{x} = f \circ x$.

These DSLs represent a major difficulty for computer scientists and systematic efforts have been undertaken by one of the authors to make them more accessible to computer science students [6, 31, 33] and improve the dialogue between the computational sciences and the physical sciences. Our paper is also a contribution to such dialogue.

We argue that the DSLs of mathematical physics are endowed with a rich but hidden "grammar of dimensions" that (functional) programming languages have failed to exploit or even recognize. This grammar informs important notions of consistency and of correctness which, in turn, are the bread and butter of program analysis, testing and derivation.

From this perspective, it is not very surprising that physicists and modellers have hardly been interested in FP. Standard FP abstractions emphasize type annotations that do not matter to physicists and modellers (for the climate scientist, all functions are, bluntly speaking, of type $\mathbb{R}^m \to \mathbb{R}^n$ for some natural numbers m and n), while at the same time failing to highlight differences that do matter like the one between a *length* and a *time*. We hope that this work will also help make FP a bit more palatable to physicists and modellers.

Outline. In section 2 we briefly review the role of equations, laws, types and *dimensions* in computer science, mathematical physics and modelling.

In section 3 we discuss the ideas of dimension, *physical quantity*, and *units of measurement* informally. This is mainly meant to guide the computer scientist out of her comfort zone but

also to answer a question that should be of interest also to readers who are familiar with modelling: "what does it mean for a parameter or for a variable to have a dimension?"

Section 4 is a short account of similarity theory [15, 16, 49] and of Buckingham's Pi theorem, mainly following Barenblatt et al. [3, Section 1]. This is going to be new ground for most computer scientists but, again, we hope to also provide a new angle to modellers who are young and have therefore mainly been imprinted with computer-based modelling.

In section 5 we introduce a minimal domain specific language for dimensionally consistent programming in Idris. We formalize the notions of dimension function, physical quantity, homomorphic measurement and dimensional (in)dependence.

In section 6, we apply the DSL to formulate the covariance principle (principle of relativity of measurements that is, there is no privileged system of units of measurement) and the Pi theorem in type theory. These formulations are, on the one hand, a benchmark for the DSL developed in section 5. On the other hand, they are a way to understand covariance and the Pi theorem and to make these notions more accessible to computer scientists. Perhaps not surprisingly, the theorem as discussed in dimensional analysis textbooks is not implementable. We discuss its non-constructive nature in section 6.2 and present a methodology for "applying" the theorem and constructing functions that provably respect the covariance principle in section 6.3 .

In section 7 we discuss possible extensions and generalizations of the DSL from section 5 and give a meaning to the equations and physical laws from section 2, while section 8 wraps up and discusses links between dimensional analysis and more general relativity principles.

All sections of this manuscript are available as literate Idris code in our gitlab repository [10]. The code associated with sections 5 and 6 is also available in non-literate Idris and Agda form in the `idris` and `agda` folders of the same repository.

Related work. Dimensional analysis (DA) [3, 15, 25, 34], is closely connected with the theory of physical similarity: under which conditions is it possible to translate observations made on a scaled model of a physical system to the system itself. This is captured in the fundamental principles of invariance and relativity (of units of measurement), as described by Arnold [1].

The theory of physical similarity was formulated well before the advent of digital computers and programming languages [15–17, 49] and its formalizations have been rare [48, 61, 62]. With the advent of digital computers and massive numerical simulations, physical modelling has been almost completely replaced by computer-based modelling, mainly for economic reasons, and the theory of physical similarity and DA are not any longer an integral component of the education of physicists, modellers and data analysts[1].

The notions of invariance (with respect to a group of transformations) and relativity (e.g., of units of measurement) in physics are similar to the notions of parametricity and polymorphism in computer science and thus it is perhaps not surprising that, as programming languages have gained more and more expressive power, mathematicians and computer scientists have started "rediscovering" DA, see Atkey et al. [2], Kennedy [36], Newman [43]. More recently the idea that dependent types can be applied to enforce the dimensional consistency of expressions involving scalar physical quantities has been generalized to expressions with vectors, tensors

[1] But Bridgman's work has been republished in 2007 by Kessinger Publishing and in 2018 by Forgotten Books and DA still plays a crucial role in the analysis of computer-based models in climate science [56–59].

and other derived quantities by McBride and Nordvall-Forsberg [41]. We return to these papers with a bit more details in section 7.4.

Dependent types can certainly be applied to enforce the dimensional consistency of expressions and libraries for annotating values of standard types with dimensional information are available in most programming languages [18, 23, 35, 46, 50, 52] and since quite some time [28].

But dependently typed languages can do more. The type checker of Idris, for example, can effectively assist the implementation of verified programs by interactively resolving the types of holes, suggesting tactics and recommending type consistent implementations. Similar support is available in other languages based on intensional type theory.

A DSL built on top of a dependently typed language that supports expressing Buckingham's Pi theorem should in principle be able to assist the interactive implementation of dimensionally consistent programs. It should support the programmer formulating the question of how a function that computes a force, say F, may depend on arguments m and a representing mass and acceleration, leverage on the type system of the host language and automatically derive $F\ m\ a = \alpha \cdot m \cdot a$ (for some α). The work presented here is a first step in this direction. We are not yet there and in section 8 we discuss which steps are left.

2 Equations, physical laws and types

In the preface to the second edition of "Programming from specifications" [42], Carroll Morgan starts with the observation that, in mathematics, $x^2 = 1$ is an equation and that $x = 1$ and $x = -1$ are equations too. He then goes on to point out that, because of the relationships between these three equations (the implications $x = 1 \Rightarrow x^2 = 1$ and $x = -1 \Rightarrow x^2 = 1$) and because $x = 1$ and $x = -1$ define the value of x "without further calculation", these two equations are called *solutions* of $x^2 = 1$.

Thus equations in mathematics sometimes represent *problems*. In dependently typed languages, these problems can be formulated explicitly and the resulting expressions can be checked for consistency. For example, in Idris one can specify the problem of finding a real number x whose square is 1 as

$$
\begin{aligned}
&x \quad\ : \mathbb{R} \\
&xSpec\ : x \uparrow 2 = 1
\end{aligned}
$$

where

$$
\begin{aligned}
&(\uparrow)\ : \mathbb{R} \to \mathbb{N} \to \mathbb{R} \\
&x \uparrow Z \quad\ = 1 \\
&x \uparrow (S\ n) = x \cdot (x \uparrow n)
\end{aligned}
$$

It is worth pointing out that 1) it is a context that is *not* immediately deducible from $x^2 = 1$ that determines the meaning of the equality sign in this equation and 2) that it is the types of x and the squaring function that make such context clear. For example, with $x : \mathbb{N}$ and $(\uparrow) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ there is just one solution, and in *Double*, $x \uparrow 2 = 2$ has no (exact) solutions.

In this paper we will often use equations between functions. In such equations, we use extensional equality implemented as follow

$$
\begin{aligned}
&(\doteq)\ : \{A, B : Type\} \to (A \to B) \to (A \to B) \to Type \\
&(\doteq)\ \{A\}\ f\ g = (x : A) \to f\ x = g\ x
\end{aligned}
$$

Because of the equivalence between logical propositions and types [60], the type $f \doteq g$ means $\forall x, f\ x = g\ x$ and values of this type are proofs of this equality. While it would be possible to work with setoid equality, or homotopy type theory, we make the pragmatic choice to use extensional equality (we explored the consequences in [9]).

2.1 Generic differential equations

In mathematical physics but also in the social sciences and in modelling it is common to specify problems implicitly in terms of equations.

Typically, such specifications are *generic* and come in the form of systems of differential equations. In this context, generic means that the problem equations are given in terms of functions which are not defined explicitly. (Thus, one has a family of systems parameterised by a function. The idea is then to study how properties of these systems, for example that of having stationary solutions, depend on properties of the parameter.) The focus is on the semantics and the syntax can be confusing. For example, the ordinary differential Equation (1.4) at page 18 of Kuznetsov [38]

$$\dot{x} = f(x) \tag{1}$$

is said to define a continuous-time *dynamical system* $(\mathcal{T}, X, \varphi)$. In this context, $\mathcal{T}$ is the *time* set (a real interval), X is the *state space* of the system, x is a function of type $\mathcal{T} \to X$, $\dot{x}$ (also of type $\mathcal{T} \to X$) is the first derivative of x, and f is a function of type $X \to X$ smooth enough to ensure existence and uniqueness of solutions. Thus, eq. (1) contains a type error. The twist here is that the equation is just an abbreviation for the specification

$$\begin{aligned} x &: \mathcal{T} \to X \\ xSpec &: D\,x \doteq f \circ x \end{aligned}$$

where we adopt the notation of Jansson et al. [33] and use $D\,x$ to denote the derivative of x. When not otherwise stated, $D\,x$ has the type of x. When quoting textbooks verbatim, we also denote $D\,x$ by $\dot{x}$ (and $D\ (D\,x)$ by $\ddot{x}$), dx/dt or similar. The third element of the dynamical system associated to eq. (1), φ, is a function of type $\mathcal{T} \to X \to X$. The idea is that $\varphi\ t\ x_0$ is the value of *the* solution of eq. (1) for initial condition x_0 at time t. Thus φ does depend on f although this is not immediately visible from its type.

In mathematical physics, it is very common to use the same notation for function application and function composition as in eq. (1). For example, Newton's principle of determinacy[2] is formalized in Equation (1) of [1] as

$$\ddot{\mathbf{x}} = F(\mathbf{x}, \dot{\mathbf{x}}, t) \tag{2}$$

where $\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}} : \mathcal{T} \to \mathbb{R}^N$ and $F : (\mathbb{R}^N, \mathbb{R}^N, \mathcal{T}) \to \mathbb{R}^N$. Again, the equation is to be understood as an abbreviation for the composition between F and the function taking t to the triple $(\mathbf{x}(t), \dot{\mathbf{x}}(t), t)$. The function F only has access to a state triple at the current instant in time, not to the full time evolution of the state.

Keeping in mind that $f(x)$ often just means $f \circ x$ and with a little bit of knowledge of the specific domain, it is often not difficult to understand the problem that equations represent. For example, in bifurcation theory, a slightly more general form of eq. (1)

[2] Newton's principle of determinacy maintains that the initial state of a mechanical systems (the positions and the velocities of its points at an initial time) uniquely determines its motion, see [1], page 4.

$$\dot{x} = f(x, p) \tag{3}$$

with parameter $p : P$ and $f : (X, P) \to X$ (again, with some abuse of notation) is often put forward to discuss three different but closely related problems:

- The problem of predicting how the system evolves in time from an initial condition $x_0 : X$ and for a given $p : P$.
- The problem of finding *stationary* points that is, values of $x : X$ such that $f(x, p) = 0$ for a given $p : P$ and to study their *stability*.
- The problem of finding *critical* parameters that is, values of $p : P$ at which the set of stationary points $\{x \mid x : X, f(x, p) = 0\}$ (or a more general *invariant* set associated with f) exhibits structural changes[3].

Mathematical models of physical systems often involve *functional* equations and systems of partial differential equations. Understanding the problems associated with these equations from a FP perspective requires some more domain-specific expertise. But even these more advanced problems can often be understood by applying "type-driven" analysis, see for example the discussion of the Lagrangian in Chapter 3 "Types in Mathematics" [33].

2.2 Physical laws, specific models

Perhaps surprisingly, understanding basic physical principles (and also mathematical models of specific systems) in terms of well-typed expressions is often tricky. Consider, for example, Newton's second law as often stated in elementary textbooks

$$F = ma \tag{4}$$

or the relationship between pressure, density and temperature in an ideal gas

$$p = \rho RT \tag{5}$$

In these equations F denotes a *force*, m a *mass*, a an *acceleration*, p a *pressure*, ρ a *density*, T a *temperature* and R a gas-specific constant. But what are the types of these quantities? What kind of equalities do these types imply?

In climate science, "conceptual" models of specific components of the climate system often consist of simple, low-dimensional systems of ordinary differential equations. For example, Stommel's seminal 1961 paper starts by describing a simple system consisting of a vessel of water with "temperature T and salinity S (in general variable in time) separated by porous walls from an outside vessel whose temperature T_e and salinity S_e are maintained at constant values", see Fig. I at page 1 of [51].

The evolution of the temperature and of the salinity in the vessel are then modeled by two uncoupled linear differential equations:

$$\frac{dT}{dt} = c(T_e - T) \quad \text{and} \quad \frac{dS}{dt} = d(S_e - S) \tag{6}$$

In this context, T and S represent functions of type $\mathbb{R} \to \mathbb{R}$ and T_e, S_e, the "temperature transfer coefficient" c, and the "salinity transfer coefficient" d are real numbers. We can formulate the problem of computing solutions of eq. (6) through the specification:

[3] In the context of climate research, such critical values are often called "tipping points".

$$TSpec\ :\ D\ T\ \doteq\ \lambda t \rightarrow c \cdot (T_e - T\ t); \qquad SSpec\ :\ D\ S\ \doteq\ \lambda t \rightarrow d \cdot (S_e - S\ t)$$

The λ-expression in *TSpec* denotes the function that maps t to $c \cdot (T_e - T\ t)$. Thus (again, because of the equivalence between types and logical propositions) any (total) *definition* of *TSpec* is (equivalent to) a proof that, $\forall t,\ dT(t)/dt = c(T_e - T(t))$ and its *declaration* specifies the task[4] of providing one such proof.

The next step at the end of page 1 of Stommel's paper is to make eq. (6) "non-dimensional" by introducing

$$\tau = c\ t, \quad \delta = \frac{d}{c}, \quad y = T/T_e, \quad x = S/S_e \tag{7}$$

This yields

$$\frac{dy}{d\tau} = 1 - y \quad \text{and} \quad \frac{dx}{d\tau} = \delta(1 - x) \tag{8}$$

at the top of page 2. From there, Stommel then goes on discussing how a *density* of the vessel (a function of its temperature and salinity and, thus, of x and y) evolves in time as the solution of eq. (8) for arbitrary initial conditions x_0, y_0

$$y(\tau) = 1 + (y_0 - 1)e^{-\tau} \quad \text{and} \quad x(\tau) = 1 + (x_0 - 1)e^{-\delta\tau} \tag{9}$$

approaches 1 as τ increases. Although Stommel's discussion is in many ways interesting, we do not need to be concerned with it here. What we need to understand, however, are the types of τ, δ, x and y and how eq. (8) can be derived from eq. (6) given eq. (7). The first two equations of eq. (7) are definitions of τ and δ:

$$\begin{array}{ll} \tau\ :\ \mathbb{R} \rightarrow \mathbb{R}; & \delta\ :\ \mathbb{R} \\ \tau\ t = c \cdot t; & \delta = d\ /\ c \end{array}$$

However, the last two equations are not *explicit* definitions of y and x. Instead, they are *implicit* definitions, corresponding to the specifications

$$\begin{array}{lll} y & :\ \mathbb{R} \rightarrow \mathbb{R}; & x\ :\ \mathbb{R} \rightarrow \mathbb{R} \\ ySpec & :\ y \circ \tau\ \doteq\ \lambda t \rightarrow T\ t\ /\ T_e; & xSpec\ :\ x \circ \tau\ \doteq\ \lambda t \rightarrow S\ t\ /\ S_e \end{array}$$

and it is easy to see that the definitions

$$y\ \sigma = T\ (\sigma\ /\ c)\ /\ T_e; \qquad x\ \sigma = S\ (\sigma\ /\ c)\ /\ S_e$$

fulfil *ySpec* and *xSpec*:

$$ySpec\ t = ((y \circ \tau)\ t)\ =\{\ Refl\ \}=\ (T\ (c \cdot t\ /\ c)\ /\ T_e)\ =\{\ ?\mathbf{h_0}\ \}=\ (T\ t\ /\ T_e)\ QED$$

and similarly for *xSpec*. Filling the $?\mathbf{h_0}$ hole in the last step requires invoking congruence and a proof of $c \cdot t\ /\ c = t$. Here, the types of c and t are aliases for double-precision floating-point numbers for which such an equality needs to be postulated. The differential equations (8) for y and x follow then from the definitions of τ and δ, from the specifications *TSpec*, *ySpec*, *SSpec*, *xSpec* and from the rules for derivation.

[4] *Aufgabe*, [37].

Informally:

$$
\begin{aligned}
& y \circ \tau \doteq \lambda t \to T\ t\ /\ T_e && \Rightarrow && \text{-- congruence} \\
& D\ (y \circ \tau) \doteq D\ (\lambda t \to T\ t\ /\ T_e) && = && \text{-- D-composition, ...} \\
& \lambda t \to (D\ y)\ (\tau\ t) \cdot (D\ \tau)\ t \doteq \lambda t \to (D\ T)\ t\ /\ T_e && = && \text{-- } (D\ \tau)\ t = (const\ c)\ t = c, \mathit{TSpec} \\
& \lambda t \to (D\ y)\ (\tau\ t) \cdot c \doteq \lambda t \to c \cdot (T_e - T\ t)\ /\ T_e && = && \text{-- arithmetics} \\
& \lambda t \to (D\ y)\ (\tau\ t) \doteq \lambda t \to 1 - T\ (c \cdot t\ /\ c)\ /\ T_e && = && \text{-- } \mathit{ySpec}\text{, def. composition} \\
& \lambda t \to ((D\ y) \circ \tau)\ t \doteq \lambda t \to 1 - y\ (\tau\ t) && = && \text{-- } \beta\text{-reduction} \\
& (D\ y) \circ \tau \doteq \lambda t \to 1 - (y \circ \tau)\ t && = && \text{-- factoring on the RHS} \\
& (D\ y) \circ \tau \doteq (\lambda \sigma \to 1 - y\ \sigma) \circ \tau && = && \text{-- cancel composition (inv. } \tau) \\
& D\ y \doteq \lambda \sigma \to 1 - y\ \sigma
\end{aligned}
$$

We can turn this into a valid Idris proof but the point here is that the computation looks significantly more contrived than the straightforward (again, informal) derivation

$$
\begin{aligned}
& y = T\ /\ T_e && \Rightarrow && \text{-- chain rule} \\
& \tfrac{dy}{d\tau}\ \tfrac{d\tau}{dt} = \tfrac{dT}{dt}\ \tfrac{1}{T_e} && = && \text{-- def. of } \tau\text{, eq. (8)} \\
& \tfrac{dy}{d\tau}\ c = c\ (T_e - T)\ /\ T_e && = && \text{-- def. of } y \\
& \tfrac{dy}{d\tau} = 1 - y
\end{aligned}
$$

that is taken for granted and not even mentioned in Stommel's paper. This complication is a consequence of having insisted on explicit types and on consistent typing rules in a language that lacks important abstractions. Specifically, it lacks rules for lifting real-number operations (like multiplication, subtraction, etc.) to operations between real numbers and functions with matching co-domain.

Such complications may partly explain why modellers and mathematicians have not found much added value in dependent types and functional languages: even for derivations as simple as those of the Stommel model, these languages add a formal layer that seems to unnecessarily obfuscate computations that are nearly self-evident.

It is not difficult to make the Idris (or Agda, Rocq, etc.) type checker digest definitions like eq. (7) and simplify derivations like those of *ySpec* and *xSpec* but, unfortunately, there are more bad news: the "dimensionality" of the expressions involved in Stommel's derivation is not visible in their types! Stommel shows that by "making the equations non-dimensional" the number of parameters of eq. (6) has been reduced to just one: the "non-dimensional" ratio δ.

This is an important result. It implies that the solution of eq. (8) for a given value of δ allows one to recover a whole family of solutions of eq. (6), namely, all those corresponding to values of c and d such that $d/c = \delta$. Conversely, it demonstrates that the problem defined by eq. (6) is *inflated*.

In mathematical modelling but also in data analysis, inflated problems are ubiquitous. Recognizing that the solution of a problem can be reduced to the solution of a much simpler problem is often crucial to avoid the "curse of dimensionality", e.g. when solving large systems of partial differential equations or in training deep neural networks.

But how can one recognize that a problem is inflated? What does it mean to say that δ is *non-dimensional*? And why are eq. (6) dimensional and eq. (8) non-dimensional? The types of the expressions involved do not provide us with any clue.

Like in the case of Newton's second law and of the equation for ideal gases, we have to do with a *grammar* of properties (being a force, being a mass, being non-dimensional) that we do not grasp. As a consequence, our types are not well suited to that grammar: T and y, S and x, m and ρ have different properties and these properties are important in the domain of discourse. Yet, they all have the same types!

If we want to take mathematical physics and modelling seriously (and if we want experts in these domains to take FP seriously) we have to encode that grammar through suitable types. We discuss a minimal DSL of dimensional quantities in section 5. Before getting there, however, we need to build a better understanding of the questions raised in this section. In the next one, we start with the question of what it means for a physical quantity to *have a dimension*.

3 Dimensions, physical quantities, units of measurement

In the last section we have discussed simple examples of equations and implicit problem specifications in the context of mathematical physics and modelling. We have seen that the variables that appear in equations like Newton's second law eq. (4) or in the ideal gas law are endowed with properties, like being a force or a temperature, that we do not know how to represent through the type system.

In the case of the Stommel model, we have encountered variables that were said to "have a dimension", for example c. Other expressions were said to be "non-dimensional". But what does it mean for c to have a dimension? In a nutshell, it means two things:

(1) That c represents a *physical quantity* that can be measured in a system of *units* of measurement of a given *class*.
(2) That the numerical measure of c will change in a specific, predictable way if the system of units is changed (within the same class).

An example will help illustrate the idea. Consider the sheet of paper on which this article is printed. Assume its width to be 20 centimetres and its height to be 30 centimetres.

If we measure *lengths* in centimetres, the measures of width and height will be 20 and 30, respectively. In metres, these measures will instead be 0.2 and 0.3. A change in the units of measurement of lengths has resulted in a change in the measures of the width and of the height of the paper: we say that the width and the height have a dimension or, equivalently, that they are dimensional quantities.

By contrast, the ratio between the height and the width of the paper is 3/2 no matter whether we measure lengths in centimetres, metres or in other units: we say that the ratio is a non-dimensional quantity.

Notice that the distinction between dimensional and non-dimensional quantities crucially relies on the (implicit) assumption of measuring *both* the height and the width of the paper (more generally, all lengths) with the same units.

The dimension judgment. In strongly typed languages like Idris and Agda, the judgment $e : t$ means that expression e has type t. In the physical sciences, the judgment $[e] = d$ means that expression e (representing a physical quantity) has dimension d. At this point,

we do not know how to formalize this judgment in type theory (we discuss how to do so in section 5) but the idea is that d is a function of type $\mathbb{R}_+^n \to \mathbb{R}_+$ where the number $n : \mathbb{N}$ is domain-specific and $\mathbb{R}_+$ denotes the set of positive real numbers.

For example, in mechanics $n = 3$. In this domain, the judgment $[\![e]\!] = \lambda(L, T, M) \Rightarrow L \cdot T^{-1}$ means that e is a quantity that can be measured in a system of units of measurements, for example SI (international) or CGS (centimetre-gram-second), for *lengths*, *times* and *masses*, and that the measure of e *increases* by a factor $L \cdot T^{-1}$ when the units for lengths, times and masses are *decreased* by factors L, T and M, respectively. Another way to express the same judgment is to say that "e is a velocity" or that "e has the dimension of a velocity". The notation was originally introduced by Maxwell, see [3, Section 1.1.3], and in DA (dimensional analysis) it is common to write $[\![e]\!] = L\,T^{-1}$ as an abbreviation for $[\![e]\!] = \lambda(L, T, M) \Rightarrow L \cdot T^{-1}$.

In mechanics, physical quantities can alternatively be measured in a system of units for *lengths*, *times* and *forces*. This defines a different *class* of units in the same domain (thus $n = 3$). In plain geometry $n = 1$ and in classical mechanics with heat transfer $n = 4$: beside units for lengths, times and forces, in this domain we also need units for *temperatures*.

For the reader who finds all this very confusing and suspiciously far away from the clear and deep waters of type theory: it is! As we have seen in section 2, the standard arsenal of functional programming abstractions is not yet ready to encode the grammar of dimensions that informs the language of mathematical physics and modelling.

If we want to develop DSLs that are palatable to mathematicians, physicists and modellers, we need to spend some time wading in shallow and muddy waters. The ideas summarized in this section are discussed more authoritatively and to a greater extent in the introduction and in Section 1 of [3].

We will give a precise specification of the higher order function $[\![\ ]\!]$ and formalize the notion of physical quantity in type theory in section 5. To get there, however, we need to first get an idea of the problems addressed by the theory of *similarity* and by Buckingham's Pi theorem. This is done in the next two sections. We conclude this one with three remarks:

Remark 1: Equations relate quantities through measurements. Equations like Newton's second principle eq. (4) or the ideal gas law eq. (5) represent relationships between physical quantities (like force, mass, and acceleration). The idea is that these equations summarize empirical facts about measurements of these quantities (or put forward axioms about such measurements). These facts (or axioms) about measurements are understood to hold under a number of assumptions, typically implicit. A crucial one is that all measurements are done in the same class and system of units.

For example, eq. (5) maintains that measurements of the pressure p of an ideal gas are proportional to the product of measurements of density ρ and measurements of temperature T, the factor of proportionality being a gas-specific constant R. We come back to the idea that equations like Newton's second principle represent relationships between physical quantities and we present a more consistent interpretation of such equations in section 7.5.

Remark 2: Context matters. The result of measurements (and thus the type of the variables entering the equations) depends on a context that is not visible in the equations themselves. For example, when the measurements of pressure, density and temperature are pertinent to a homogeneous gas, eq. (5) can be interpreted as the specification

$$pSpec : \mu\ p = \mu\ \rho \cdot \mu\ R \cdot \mu\ T$$

where μ is the measure function (which will talk more about later). In a context in which p, ρ and T represent the pressure, the density and the temperature of a gas in local thermodynamical equilibrium, the same equation can be interpreted as the specification

$$pSpec\ :\ \mu\ p \doteq \lambda(x,t) \Rightarrow \mu\ (\rho\ (x,t)) \cdot \mu\ R \cdot \mu\ (T\ (x,t))$$

This is typically the case when a symbol that represents the pressure appears in the right hand side of a system of partial differential equations like the Euler or the Navier-Stokes equations of fluid mechanics [19]. In yet another context, p, ρ and T could represent probability density functions or other higher order functions. But what could be the types of p, ρ, R and T in these contexts? We answer this question in section 5.2.

Remark 3: Dimensions, functions, and derivatives. When p is a function taking inputs from, e.g., $(\mathbb{R}, \mathcal{T})$, the judgment "$p$ is a pressure" (or "p has the dimension of a pressure" or, $[p] = M\ L^{-1}\ T^{-2}$) is just an abbreviation for "$\forall\ (x,t)\ :(\mathbb{R},\mathcal{T}),\ p\ (x,t)$ is a pressure". If p is differentiable with respect to both space and time and if the space and time coordinates are dimensional (that is, $\forall\ (x,t)\ :(\mathbb{R},\mathcal{T})$, $[x] = L$ and $[t] = T$) then the partial derivatives of p with respect to space and time have the dimensions $M\ L^{-2}\ T^{-2}$ and $M\ L^{-1}\ T^{-3}$, respectively. More generally, if p is a function from a dimensional space-time set into real numbers and p has dimension d, the partial derivatives of p with respect to time and space are functions of the same type as p but with dimensions $d \cdot T^{-1}$ and $d \cdot L^{-1}$. Again, these are shortcuts for $\lambda(L,T,M) \Rightarrow d\ (L,T,M) \cdot T^{-1}$ and $\lambda(L,T,M) \Rightarrow d\ (L,T,M) \cdot L^{-1}$, respectively.

As already mentioned in section 2, the last specification for p could be written more concisely as $p \doteq \rho \cdot R \cdot T$ by introducing canonical abstractions for functions that return numerical values. To the best of our knowledge, no standard Idris library provides such abstractions although, as discussed in the introduction, there have been proposals for making types in FP languages more aware of dimensions and physical quantities, for example by Baumann et al. [4].

4 Similarity theory and the Pi theorem

The notion of dimension function informally introduced in the last section is closely related with a fundamental principle in physical sciences and with a very pragmatic question in physical modelling.

We start with the pragmatic question. At the turn of the 20th century, no computers were available for approximating numerical solutions of mathematical models of physical systems, typically in the form of partial differential equations. Thus, models of physical systems, say of a ship cruising in shallow waters or of an airplane, were themselves physical systems, for convenience often at a reduced *scale*. This raised the obvious question of the conditions under which careful measurements made on the scaled model could be "scaled up" to the real system and how this scaling up should be done.

For example, if the drag on a 1:50 model of a ship cruising in a water channel was found to be x Newton, what would be the drag of the real ship? And under which conditions is it possible to give a definite answer to this question?

At first glance, the problem seems to be one of engineering. But the theory that was developed to answer this question, similarity theory or dimensional analysis (DA), turned out to be a logical consequence of a fundamental principle: "physical laws do not depend

on arbitrarily chosen basic units of measurement", see [3, section 0.1]. This is, in turn, an instance of Galileo's principle of relativity, see [1, page 3].

The core results of DA can be summarized in Buckingham's Pi theorem and this theorem is also an answer to the question(s) of model similarity raised above. We do not need to be concerned with these answers and with specific applications of the Pi theorem in the physical sciences here. But we need to understand the notions that are at core of this theorem in order to develop a DSL that is suitable for mathematical physics and for modelling.

We introduce Buckingham's Pi theorem as it is stated in [3]. This formulation is consistent with textbook presentations and raises a number of questions. We flag these questions here and then tackle them from a functional programming perspective in section 5 where we build a minimal DSL for dimensionally consistent programming.

4.1 Buckingham's Pi theorem

The theorem is stated at page 42 of [3]:

> A physical relationship between some dimensional (generally speaking) quantity and several dimensional governing parameters can be rewritten as a relationship between some dimensionless parameter and several dimensionless products of the governing parameters; the number of dimensionless products is equal to the total number of governing parameters minus the number of governing parameters with independent dimensions.

This formulation comes at the end of a derivation that starts at page 39 by positing a "physical relationship" between a "dimensional quantity" a and $k+m$ "dimensional governing parameters" $a_1, \ldots, a_k$ and $b_1, \ldots, b_m$

$$a = f(a_1, \ldots, a_k, b_1, \ldots, b_m) \tag{10}$$

such that $a_1, \ldots, a_k$ "have independent dimensions, while the dimensions of parameters $b_1, \ldots, b_m$ can be expressed as products of powers of the dimensions of the parameters $a_1, \ldots, a_k$":

$$[b_i] = [a_1]^{p_{i1}} \ldots [a_k]^{p_{ik}} \quad i = 1, \ldots, m \tag{11}$$

With these premises, the conclusions are then 1) that "the dimension of a must be expressible in terms of the dimensions of the governing parameters $a_1, \ldots, a_k$":

$$[a] = [a_1]^{p_1} \ldots [a_k]^{p_k} \tag{12}$$

and 2) that the function f "possesses the property of generalized homogeneity or symmetry, i.e., it can be written in terms of a function of a smaller number of variables, and is of the following special form":

$$f(a_1, \ldots, a_k, b_1, \ldots, b_m) = a_1^{p_1} \ldots a_k^{p_k}\ \Phi(\Pi_1, \ldots, \Pi_m) \tag{13}$$

where $\Pi_i = b_i/(a_1^{p_{i1}} \ldots a_k^{p_{ik}})$ for $i = 1, \ldots, m$. On page 42ff, the author comments that the term "physical relationship" for f "is used to emphasize that it should obey the covariance principle" and, further, that the Π-theorem is "completely obvious at an intuitive level" and that "it is clear that physical laws should not depend on the choice of units" and that this "was realized long ago, and concepts from dimensional analysis were in use long before the Π-theorem had been explicitly recognized, formulated and proved formally" among others, by "Galileo, Newton, Fourier, Maxwell, Reynolds and Rayleigh". Indeed, one of the most

successful applications of the theorem was Reynolds' scaling law for fluid flows in pipes in 1883, well before Buckingham's seminal papers [16, 17].

Here we do not discuss applications of the Π-theorem, but its relevance for data analysis, parameter identification, and sensitivity analysis is obvious: the computational cost of these algorithms is typically exponential in the number of parameters. The theorem allows cost reductions that are exponential in the number of parameters with independent dimensions. In the case of f, for example, the theorem allows the cost to be reduced from N^{k+m} to N^m where N denotes a sampling size. In data-based modelling and machine learning, this can make the difference between being able to solve a problem in principle and being able to solve it in practice.

The bottom line is that, even though DA and the Π-theorem were formulated at a time in which computer based modelling was not available and were mainly motivated by the questions of model similarity mentioned above, they remain highly relevant today.

For us, the challenge is to understand how to apply dependent types to 1) systematically check the dimensional consistency of expressions and 2) assist DA.

In section 3, we have seen that what in mathematical physics and modelling are called physical quantities are equipped with a dimension function. In analogy with the judgment $e : t$ (or, *typeOf* $e = t$), we have informally introduced the judgment $[e] = d$ to denote that expression e has dimension function d. With the Π-theorem, we have seen that physical quantities may have "independent dimensions" or be dimensionally dependent. In eqs. (11) and (12) we have encountered (systems of) equations between dimension functions whose solutions play a crucial role in the Π-theorem eq. (13). In the next section we come back to the notion of dimension function of a physical quantity $[x] : \mathbb{R}_+^n \to \mathbb{R}_+$, discuss its relationship with units of measurement, and argue that $[x]$ is a power-law monomial. This property is at the core of the dependently typed formalization of DA presented in sections 5 and 6.

5 A minimal DSL for dimensionally consistent programming

In this section we introduce a minimal domain-specific language for dimensionally consistent programming in Idris and in the context of classical mechanics. We formalize the notions of dimension function, physical quantity, measurement and units of measurement from section 3 and the notion of dimensional (in)dependence which are at the core of the Pi theorem from section 4.

The DSL supports expressing dimensional judgments and dimensionally consistent programming in the domain of classical mechanics. It can be straightforwardly extended to higher dimensional domains (for example, thermodynamics) or restricted to subdomains of mechanics like kinematics or non-fractal geometry.

At the core of the DSL is the idea that any physical quantity can be associated with a dimension function. We follow a concrete, minimalist approach and encode dimension functions in terms of vectors of integers.

5.1 Dimension function

In section 3, we said that the dimension function of a physical quantity x, $[x] : \mathbb{R}_+^n \to \mathbb{R}_+$ encodes the idea that x can be measured in a system of n fundamental units of measurement and that the number $[x]\ (L_1, \ldots, L_n)$ denotes the factor by which the measure of x increases (is multiplied) when the n units are decreased (divided) by $L_1, \ldots, L_n$.

We can give a precise meaning to this idea by denoting the measure of x in the units of measurement $u_1, \ldots, u_n$ by $\mu\ (u_1, \ldots, u_n)\ x$. We will sometimes write u for the tuple $(u_1, \ldots, u_n)$ and shorten the notation to $\mu_u\ x$ for $\mu\ u\ x$. For the time being, we posit that $\mu_u\ x : \mathbb{R}$ but defer the specification of the types of x and u to sections 5.2 and 5.4. The reader should keep in mind that x could represent a velocity or a stress tensor and the result of measuring x could be a vector or a tensor of real numbers.

With these premises, we can make precise the notion of dimension function through (for any non-zero $u_1, \ldots, u_n$):

$$[x]\ (L_1, \ldots, L_n) = \frac{\mu\ (u_1\ /\ L_1, \ldots, u_n\ /\ L_n)\ x}{\mu\ (u_1, \ldots, u_n)\ x} \tag{14}$$

The specification suggests that the dimension function of x does not depend on the units of measurement. It formalizes the principle of covariance (or the relativity of measurements): that there is no privileged system of units of measurement or, in other words, that all systems are equally good:

$$\frac{\mu\ (u_1\ /\ L_1, \ldots, u_n\ /\ L_n)\ x}{\mu\ (u_1, \ldots, u_n)\ x} = \frac{\mu\ (u'_1\ /\ L_1, \ldots, u'_n\ /\ L_n)\ x}{\mu\ (u'_1, \ldots, u'_n)\ x} \tag{15}$$

for any physical quantity x, systems of units u and u' and scaling factors $L_1, \ldots, L_n$. It is easy to see that the principle 15 implies that the dimension function fulfils

$$[x]\ (L_1\ /\ L'_1, \ldots, L_n\ /\ L'_n) = \frac{[x]\ (L_1, \ldots, L_n)}{[x]\ (L'_1, \ldots, L'_n)} \tag{16}$$

by equational reasoning

$$\begin{aligned}
&[x]\ (L_1, \ldots, L_n)\ /\ [x]\ (L'_1, \ldots, L'_n) \\
&\quad = \quad \text{-- use eq. (14): def. of } [x] \text{ for units } (u_1, \ldots, u_n) \\
&\mu\ (u_1\ /\ L_1, \ldots, u_n\ /\ L_n)\ x\ /\ \mu\ (u_1\ /\ L'_1, \ldots, u_n\ /\ L'_n)\ x \\
&\quad = \quad \text{-- let } u'_1 = u_1\ /\ L'_1, \ldots, u'_n = u_n\ /\ L'_n \\
&\mu\ (u'_1\ /\ (L_1\ /\ L'_1), \ldots, u'_n\ /\ (L_n\ /\ L'_n))\ x\ /\ \mu\ (u'_1, \ldots, u'_n)\ x \\
&\quad = \quad \text{-- use eq. (14): def. of } [x] \text{ for units } (u'_1, \ldots, u'_n) \\
&[x]\ (L_1\ /\ L'_1, \ldots, L_n\ /\ L'_n)
\end{aligned}$$

From eq. (16) it follows that

$$[x]\ (1, \ldots, 1) = 1 \tag{17}$$

and that dimension functions have the form of power-law monomials

$$[x]\ (L_1, \ldots, L_n) = L_1^{d_1\ x} \cdot \ldots \cdot L_n^{d_n\ x} \tag{18}$$

The derivation of this power-law form is not straightforward and involves solving the functional equation eq. (16), see [3, section 1.1.4]. The exponents $d_1\ x, \ldots d_n\ x$ are sometimes called (perhaps confusingly) the "dimensions" of x and x is said to "have dimensions" $d_1\ x, \ldots d_n\ x$. Their value can be obtained by recalling the specification eq. (14).

For concreteness, consider the case of mechanics already discussed in section 3. Here $n = 3$ and, in a system of units of measurements for lengths, times and masses, the scaling factors L_1, L_2 and L_3 are denoted by L, T and M. For consistency, we denote the exponents $d_1\ x$, $d_2\ x$ and $d_3\ x$ by $d_L\ x$, $d_T\ x$ and $d_M\ x$, respectively.

Thus, in mechanics, eq. (14) tells us that $L^{d_L\ x} \cdot T^{d_T\ x} \cdot M^{d_M\ x}$ is the factor by which the measure of x gets multiplied when the units of measurement for lengths, times, and masses are divided by L, T and M. Therefore, when x represents a length (for example, the distance between two points or a space coordinate) we have $d_L\ x = 1$ and $d_T\ x = d_M\ x = 0$. Similarly, when x represents a mass we have $d_M\ x = 1$ and $d_L\ x = d_T\ x = 0$ and when x represents a time we have $d_T\ x = 1$ and $d_L\ x = d_M\ x = 0$. And when x represents a velocity (a distance divided by a time), the factor by which the measure of x gets multiplied when the units of measurement for lengths, times, and masses are divided by L, T and M shall be L/T and thus $d_L\ x = 1$, $d_T\ x = -1$, and $d_M\ x = 0$.

These judgments are a consequence of the notion of (direct or indirect) measurement as *counting*: when we say that the length of a pen is 20 centimetres we mean that we have to add 20 times the length of a centimetre to obtain the length of that pen.

The above analysis suggests that, in classical mechanics, the type of dimension functions is isomorphic to $\mathbb{Z}^3$, see also page 3 of McBride and Nordvall-Forsberg [41]. Thus, in this domain, we can represent a dimension function with a vector of integers of length 3:

```
namespace Mechanics
  namespace LTM
    data Units = SI | CGS
    D : Type
    D = Vec 3 ℤ
```

Here, we have embedded the datatypes *Units* (with just two codes for the *SI* and *CGS* systems of units for simplicity) and *D* in the namespaces *Mechanics* and *LTM*, the latter representing the class of units for lengths, times and masses, see section 3. Following our analysis, we can model the syntax of dimensional expressions in terms of a *DimLess* vector for dimensionless quantities, of *Length*, *Time* and *Mass* vectors for lengths, times and masses (the dimensions associated with the fundamental units of measurement in *LTM*)

```
DimLess : D;        Length : D;        Time : D;        Mass : D
DimLess = [0,0,0];  Length = [1,0,0];  Time = [0,1,0];  Mass = [0,0,1]
```

and of two combinators *Times* and *Over*:

```
Times : D → D → D;  Over : D → D → D
Times = (+);        Over = (−)
```

where $(+)$ and $(-)$ are the canonical addition and subtraction between vectors of integer numbers. These correspond to the idea that the dimensions of derived units of measurement (for example, for velocities or for energies) are obtained by multiplying or by dividing the dimensions of the fundamental units:

```
Velocity : D;                    Acceleration : D
Velocity = Length `Over` Time;   Acceleration = Velocity `Over` Time

Force : D;                       Work : D
Force = Mass `Times` Acceleration;  Work = Force `Times` Length

Energy : D
Energy = Mass `Times` (Velocity `Times` Velocity)
```

One can easily check that energy and mechanical work are equivalent, as one would expect

$$check_1 : Energy = Work$$
$$check_1 = Refl$$

that force and energy are different notions

$$check_2 : Not\ (Force = Energy)$$
$$check_2\ Refl\ impossible$$

and, we can compute the dimension functions of D-values:

$$df : D \to \mathbb{R}_+^3 \to \mathbb{R}_+$$
$$df\ d\ ls = prodReal\ (zipWith\ ipow\ ls\ d)$$

In the definition of *df*, *ipow* is the function that computes the integer power of a real number and $prodReal = foldr\ (\lambda m \to \mathbb{R})\ (\cdot)\ 1.0$. The dimension function satisfies eq. (18) by construction. One can see that it also satisfies eq. (16) (this is *dfLemma1* with $ls = (L_1 \,/\, L_1', \ldots, L_n \,/\, L_n')$, $ls' = (L_1', \ldots, L_n')$ and with the componentwise multiplication $ls \cdot ls'$), eq. (17) that is, *dfLemma2*

$$dfLemma1 : (d : D) \to (ls, ls' : \mathbb{R}_+^3) \to df\ d\ ls \cdot df\ d\ ls' = df\ d\ (ls \cdot ls')$$
$$dfLemma2 : (d : D) \to df\ d\ one3 = 1.0$$

and the following properties

$$dfLemma3 : \{d : D\} \to \{u, v, w : Units\} \to df\ d\ (fs\ u\ v) \cdot df\ d\ (fs\ v\ w) = df\ d\ (fs\ u\ w)$$
$$dfDimLessLemma : (ls : \mathbb{R}_+^3) \to df\ DimLess\ ls = 1.0$$
$$dfHomTimes : \{d_1, d_2 : D\} \to \{ls : \mathbb{R}_+^3\} \to df\ (d_1\ `Times`\ d_2)\ ls = df\ d_1\ ls \cdot df\ d_2\ ls$$
$$dfHomOver : \{d_1, d_2 : D\} \to \{ls : \mathbb{R}_+^3\} \to df\ (d_1\ `Over`\ d_2)\ ls = df\ d_1\ ls \,/\, df\ d_2\ ls$$

These follow from elementary properties of real number and integer power arithmetic (see the Idris and Agda code in the repository [10]) and are crucial for ensuring that measurements of physical quantities fulfil the covariance principle and for the results to be discussed in section 6.1.

Notice that for both D and *Units* we use datatypes of codes (just syntactic expressions) and that *df* here and *fs* later (section 5.4) are used to translate these codes to their intended semantics. The use of integers for the exponents of the dimension function makes dimensional judgments decidable, which would not hold for real numbered exponents, or functions.

We come back to the choice of types in section 7 where we put forward specifications for data types that implement dimension functions in terms of type classes. For the rest of this section, we stick to the example of mechanics and to the representation of dimension functions in terms of vectors of three integer exponents.

5.2 Physical quantities and homomorphic measurement

We are now ready to formalize the notion of physical quantity informally introduced in section 3. There, we posited that a parameter (e.g., the parameter c of the Stommel model discussed in section 2.2) represents a *dimensional* physical quantity when 1) that parameter can be measured in a system of units of a given class and 2) one can define another system of units in the same class that gives a different measurement for the parameter. By contrast, measurements of *dimensionless* physical quantities do not change when the units of measurement are re-scaled.

This suggests that a (dimensional or dimensionless) physical quantity can be represented by a value of type

$$\begin{aligned}&\textbf{data } Q : D \to \mathit{Type} \textbf{ where}\\ &\quad \mathit{Val} : \{d : D\} \to (u : \mathit{Units}) \to \mathbb{R} \to Q\ d\end{aligned}$$

effectively annotating $\mathbb{R}$ values with different dimensions and systems of units, for example

$$\begin{aligned}&x : Q\ \mathit{Length}; \quad t : Q\ \mathit{Time}; \quad m : Q\ \mathit{Mass}\\ &x = \mathit{Val\ SI}\ 3; \quad t = \mathit{Val\ CGS}\ 1; \quad m = \mathit{Val\ SI}\ 2\end{aligned}$$

What kind of combinators are required for physical quantities? As a minimum, one wants to be able to project the D-value of a physical quantity

$$\begin{aligned}&\mathit{dim} : \{d : D\} \to Q\ d \to D\\ &\mathit{dim}\ \{d\}\ \_ = d\end{aligned}$$

and thus compute its dimension function $\mathit{df} \circ \mathit{dim}$. Crucially, we need a way to measure physical quantities in different units of measurement

$$\begin{aligned}&\mu : \{d : D\} \to \mathit{Units} \to Q\ d \to \mathbb{R}\\ &\mu\ \{d\}\ u'\ (\mathit{Val}\ u\ x) = x \cdot \mathit{df}\ d\ (\mathit{fs}\ u\ u')\end{aligned}$$

In the definition of μ, *fs* is the table that returns the scaling factors between units of measurement. It fulfills

$$\begin{aligned}&\mathit{fsLemma}_1 : (u : \mathit{Units}) \to \mathit{fs}\ u\ u = [1, 1, 1]\\ &\mathit{fsLemma}_2 : (u, v : \mathit{Units}) \to \mathit{fs}\ u\ v = \mathit{invRealPos3}\ (\mathit{fs}\ v\ u)\\ &\mathit{fsLemma}_3 : (u, v, w : \mathit{Units}) \to (\mathit{fs}\ u\ v) \cdot (\mathit{fs}\ v\ w) = \mathit{fs}\ u\ w\end{aligned}$$

by construction. Remember that, as discussed in section 3, measurements of dimensionless physical quantities are to be the same in all units of measurement. In other words, μ has to fulfil

$$\mathit{measDimLessLemma} : \{u, u' : \mathit{Units}\} \to (q : Q\ \mathit{DimLess}) \to \mu_u\ q = \mu_{u'}\ q$$

This lemma is a straightforward consequence of eq. (17) that is *dfDimLessLemma*. Further, it is useful to define elementary binary operations $(+)$, $(\cdot)$, $(\triangleleft)$:

$$\begin{aligned}&(+) : \{d : D\} \to Q\ d \to Q\ d \to Q\ d\\ &q_1 + q_2 = \mathit{Val\ SI}\ (\mu_{SI}\ q_1 + \mu_{SI}\ q_2)\\ &(\cdot) : \{d_1, d_2 : D\} \to Q\ d_1 \to Q\ d_2 \to Q\ (d_1\ \text{`}\mathit{Times}\text{`}\ d_2)\\ &q_1 \cdot q_2 = \mathit{Val\ SI}\ (\mu_{SI}\ q_1 \cdot \mu_{SI}\ q_2)\\ &(\triangleleft) : \{d : D\} \to \mathbb{R} \to Q\ d \to Q\ d\\ &x \triangleleft (\mathit{Val}\ u\ y) = \mathit{Val}\ u\ (x \cdot y)\end{aligned}$$

with similar definitions for subtraction $(-)$, division $(/)$ and right scaling $(\triangleright)$ of physical quantities. Notice that addition (subtraction) is only defined for quantities of the same dimensions. This helps avoiding "adding apples and oranges" in expressions involving physical quantities. When such additions make sense, as in the example at page 6 of [3], the computations can be implemented by pattern matching, as done in the definition of μ.

Remember that the core result of the Pi theorem is that physical laws can be expressed through products of powers of physical variables, see eq. (13). In order to formalize this result in section 6.1, we need two more combinators

$$
\begin{aligned}
&pow : \{d : D\} \to Q\ d \to (n : \mathbb{Z}) \to Q\ (d\ \text{`}Pow\text{`}\ n)\\
&pow\ (Val\ u\ x)\ n = Val\ u\ (ipow\ x\ n)
\end{aligned}
$$

$$
\begin{aligned}
&prodPows : \{n : \mathbb{N}\} \to \{ds : Vec\ n\ D\} \to Vec_{\mathrm{Q}}\ n\ ds \to (ps : Vec\ n\ \mathbb{Z}) \to Q\ (ProdPows\ ds\ ps)\\
&prodPows\ Nil\ \ \ Nil\ \ \ = Val\ SI\ 1.0\\
&prodPows\ (q :: qs)\ (p :: ps) = (pow\ q\ p) \cdot (prodPows\ qs\ ps)
\end{aligned}
$$

In the return types of *pow* and *prodPows* we have applied integer powers of *D*-values *Pow* and its generalization *ProdPows* to products of integer powers. *Pow* fulfils the specification

$$
\begin{aligned}
&Pow\ d\ 0\ \ \ \ \ \ = DimLess\\
&Pow\ d\ (n+1) = Pow\ d\ n\ \text{`}Times\text{`}\ d\\
&Pow\ d\ (n-1) = Pow\ d\ n\ \text{`}Over\text{`}\ d
\end{aligned}
$$

and *ProdPows* is a fold after a zip:

$$
\begin{aligned}
&ProdPows : \{n : \mathbb{N}\} \to Vec\ n\ D \to Vec\ n\ \mathbb{Z} \to D\\
&ProdPows\ Nil\ \ \ Nil\ \ \ = DimLess\\
&ProdPows\ (d :: ds)\ (p :: ps) = Pow\ d\ p\ \text{`}Times\text{`}\ ProdPows\ ds\ ps
\end{aligned}
$$

Note that the definition of *prodPows* has the same structure as the definition of its type (index) *ProdPows*. This is an example of type-driven development [14].

Because of the definition of *DimLess*, *Times* and *Over* from section 5.1, *Pow* and *ProdPows* also fulfil

$$
\begin{aligned}
&Pow\ d\ n = n \triangleleft d\\
&ProdPows\ ds\ ps = foldr\ (+)\ DimLess\ (zipWith\ (\triangleleft)\ ps\ ds)
\end{aligned}
$$

where $(\triangleleft)$ is generic scaling for vectors of numerical types:

$$
\begin{aligned}
&(\triangleleft) : \{n : \mathbb{N}\} \to \{T : Type\} \to Num\ T \Rightarrow T \to Vec\ n\ T \to Vec\ n\ T\\
&x \triangleleft v = map\ (x\cdot)\ v
\end{aligned}
$$

Finally, the data type Vec_{Q} in the signature of *prodPows* represents vectors of physical quantities of potentially different dimensions: $Vec_{\mathrm{Q}}\ n\ ds = All\ Q\ ds$.

Physical quantities and "phantom" types. We have introduced physical quantities through the "phantom[5]" data type $Q : D \to Type$. The type parameter *D* allows one to *specify* physical quantities of given dimensions as in $x : Q\ Length$. When *defining* x one has to provide its measure in a system of units as in $x = Val\ SI\ 3$. An alternative design could be to make also the units of measurement visible in the type of physical quantities:

$$
\begin{aligned}
&\textbf{data}\ Q_1 : D \to Units \to Type\ \textbf{where}\\
&\quad Val_1 : \{d : D\} \to (u : Units) \to \mathbb{R} \to Q_1\ d\ u
\end{aligned}
$$

This would allow one to avoid the introduction of an apparently distinguished system of units in the definition of the elementary binary operations, *SI* in our implementation. With Q_1, addition would read

$$
\begin{aligned}
&(+) : \{d : D\} \to \{u, v, w : Units\} \to Q_1\ d\ u \to Q_1\ d\ v \to Q_1\ d\ w\\
&(+)\ q_1\ q_2 = Val_1\ w\ (\mu\ w\ q_1 + \mu\ w\ q_2)
\end{aligned}
$$

[5] See https://wiki.haskell.org/Phantom_type.

A practical drawback of that approach is the proliferation of implicit parameters. More importantly, exposing the units of measurement in the type of physical quantities is conceptually unsatisfactory: whether x has been defined to be a length of one meter or 1.093613 yards does not really matter and we argue that this choice shall not be visible in its type.

The covariance principle for elementary operations. Perhaps not surprisingly, μ_u is a homomorphism from $(+)$, $(\cdot)$, $(\triangleleft)$, $(-)$, $(/)$ and $(\triangleright)$ between physical quantities and the corresponding binary operations in $\mathbb{R}$. For example

$$
\begin{aligned}
&\mu HomMult : (u : Units) \to \{d_1, d_2 : D\} \to \\
&\qquad (q_1 : Q\ d_1) \to (q_2 : Q\ d_2) \to \mu_u\ (q_1 \cdot q_2) = \mu_u\ q_1 \cdot \mu_u\ q_2 \\
&\mu HomMult\ u\ \{d_1\}\ \{d_2\}\ (Val\ u_1\ x_1)\ (Val\ u_2\ x_2) = \\
&\quad \textbf{let} \quad \text{-- Factors to convert from local units to SI} \\
&\qquad f_1 = df\ d_1\ (fs\ u_1\ SI) \\
&\qquad f_2 = df\ d_2\ (fs\ u_2\ SI) \\
&\qquad\quad \text{-- Factors to convert from SI to target unit u} \\
&\qquad g_1 = df\ d_1\ (fs\ SI\ u) \\
&\qquad g_2 = df\ d_2\ (fs\ SI\ u) \\
&\qquad g_{12} = df\ (d_1\ \text{`}Times\text{`}\ d_2)\ (fs\ SI\ u) \\
&\quad \textbf{in}\ \ \mu_u\ (Val\ u_1\ x_1 \cdot Val\ u_2\ x_2) \\
&\qquad\quad =\{\ Refl\ \}= \\
&\qquad ((x_1 \cdot f_1) \cdot (x_2 \cdot f_2)) \cdot g_{12}) \\
&\qquad\quad =\{\ cong\ \{f = \lambda y \Rightarrow ((x_1 \cdot f_1) \cdot (x_2 \cdot f_2)) \cdot y\}\ dfHomTimes\ \}= \\
&\qquad (((x_1 \cdot f_1) \cdot (x_2 \cdot f_2)) \cdot (g_1 \cdot g_2)) \\
&\qquad\quad =\{\ lemma\mathbb{R}_4\ x_1\ f_1\ x_2\ f_2\ g_1\ g_2\ \}= \\
&\qquad ((x_1 \cdot (f_1 \cdot g_1)) \cdot (x_2 \cdot (f_2 \cdot g_2))) \\
&\qquad\quad =\{\ cong\ \{f = \lambda y \Rightarrow (x_1 \cdot y) \cdot (x_2 \cdot (f_2 \cdot g_2))\}\ dfLemma3\ \}= \\
&\qquad ((x_1 \cdot df\ d_1\ (fs\ u_1\ u)) \cdot (x_2 \cdot (f_2 \cdot g_2))) \\
&\qquad\quad =\{\ cong\ \{f = \lambda y \Rightarrow (x_1 \cdot df\ d_1\ (fs\ u_1\ u)) \cdot (x_2 \cdot y)\}\ dfLemma3\ \}= \\
&\qquad ((x_1 \cdot df\ d_1\ (fs\ u_1\ u)) \cdot (x_2 \cdot df\ d_2\ (fs\ u_2\ u))) \\
&\qquad\quad =\{\ Refl\ \}= \\
&\qquad (\mu_u\ (Val\ u_1\ x_1) \cdot \mu_u\ (Val\ u_2\ x_2)) \\
&\qquad\quad QED
\end{aligned}
$$

The proofs require postulating commutativity and associativity of $(\cdot)$, distributivity of $(\cdot)$ over $(+)$ and $(-)$ in $\mathbb{R}$ (packaged up in $lemma\mathbb{R}_4$) and, crucially, the properties of dimension functions *dfLemma3*, *dfHomTimes* and *dfHomTimes* discussed in section 5.1.

The fact that μ_u is a homomorphism between elementary binary operations on physical quantities and the corresponding operations in $\mathbb{R}$ is a special form of the covariance principle. We discuss a general form of this principle in section 6.

5.3 Dimensional judgments, dimensional consistent programming

The infrastructure introduced so far allows one to define new physical quantities from existing ones (remember that x and t were declared to be a length and a time in section 5.2)

$$
\begin{aligned}
&v : Q\ Velocity \\
&v = (2 \triangleleft x)\ /\ t
\end{aligned}
$$

and implement dimensional judgments for verified programming like

$$check_3 : dim\ (x\ /\ (t \cdot t)) = Acceleration$$
$$check_3 = Refl$$

Here, a value of type $dim\ (x\ /\ (t \cdot t)) = Acceleration$ serves as proof that $x\ /\ (t \cdot t)$ is an acceleration. We have seen many examples of this kind of dimensional judgments in previous sections: f is a force, m is a mass, T is a temperature, p is a pressure, etc. We can express these judgments through the idiom

$$Is : \{d : D\} \to Q\ d \to D \to Type$$
$$Is\ q\ d = (dim\ q = d)$$

and write $Is\ q\ d$ instead of $dim\ q = d$:

$$check_4 : Is\ (m \cdot x\ /\ (t \cdot t))\ Force$$
$$check_4 = Refl$$

Similarly, we can assess whether a physical quantity is dimensionless or not

$$IsDimLess : \{d : D\} \to Q\ d \to Type$$
$$IsDimLess\ q = (dim\ q = DimLess)$$
$$check_5 : IsDimLess\ ((x + x)\ /\ x)$$
$$check_5 = Refl$$

As one would expect, dimensionless quantities are invariant under re-scaling of the units of measurement

$$\lambda\sqcap >\ \ \mu\ SI\ ((x + x)\ /\ x)$$
$$2.0 : Double$$
$$\lambda\sqcap >\ \ \mu\ CGS\ ((x + x)\ /\ x)$$
$$2.0 : Double$$

and the dimension function fulfils the specification (14): by the definition of μ, measurements only depend on the scaling factors between the units of measurement, not on the units themselves.

Notice, however, that a direct encoding of the perhaps most discussed application of dimensional analysis does not work out. If we try to compute the period τ of small-angle oscillation of a simple pendulum of length l in a gravitational field of strength g

$$l : Q\ Length; \quad g : Q\ Acceleration; \quad \pi : Q\ DimLess$$
$$l = Val\ SI\ 0.5; \quad g = Val\ SI\ 9.81; \quad \pi = Val\ SI\ 3.14$$

as

$$\tau : Q\ Time$$
$$\tau = 2 \triangleleft \pi \cdot sqrt\ (l\ /\ g)$$

we get an error. This is because we have represented dimension functions through vectors of *integer* rather than *rational* numbers and therefore we cannot define a *sqrt* function for arbitrary physical variables.

Representing dimension functions through integer vectors is what makes our DSL "minimal". We motivate this choice in section 5.5 and discuss possible generalizations in section 7.

5.4 Units of measurement

We have introduced the notion of units of measurement through a datatype *Units* with just two data constructors: *SI* and *CGS*. This is also the approach followed by Barenblatt et al. [3] and reflects the idea that units of measurement are just annotations. There is nothing wrong with this approach and *Units* can readily be extended to accommodate more systems of units by enumeration.

Notice, however, that this requires defining *fs*, the table that returns the scaling factors between units of measurement, for all possible pairs of data constructors. For the definition of *Units* above, for example, *fs* was defined as:

$$
\begin{aligned}
&fs : Units \to Units \to \mathbb{R}^3_+ \\
&fs\ SI\ \ SI\ \ = [\,1,\ \ 1, 1\ \ \ ] \\
&fs\ SI\ \ CGS = [\,100,\ 1, 1000\ ] \\
&fs\ CGS\ SI\ \ = [\,0.01, 1, 0.001\,] \\
&fs\ CGS\ CGS = [\,1,\ \ 1, 1\ \ \ ]
\end{aligned}
$$

The table fulfils $fsLemma_1$, $fsLemma_2$ and $fsLemma_3$ by construction which can be exploited to reduce the amount of boiler-plate code in the definition of *fs*.

Still, defining *Units* through enumeration is perhaps not completely satisfactory from a conceptual point of view: first, it does not explicitly encode the idea that units of measurement are distinguished properties of *reference* physical quantities. For example, the standard metre is defined as the length of the path traveled by light in vacuum during a time interval of 1/299792458 of a second[6].

Second, a datatype with only two (or even a countable number of) inhabitants is perhaps a bit too small to encode a principle (that there is no privileged system of units of measurement) that must hold for an uncountable number of systems of units.

One can avoid these shortcomings by introducing a data constructor for systems of units that takes as arguments strictly positive but otherwise arbitrary reference lengths, times and masses. For example

```
mutual
  data Units : Type where
    SI :  Units
    Ref : (l : Q Length) → {auto p : (0.0 < val l) = True} →
          (t : Q Time)   → {auto q : (0.0 < val t) = True} →
          (m : Q Mass)   → {auto r : (0.0 < val m) = True} → Units
  data Q : D → Type where
    Val : {d : D} → (u : Units) → ℝ → Q d
  val : {d : D} → Q d → ℝ
  val (Val u x) = x
```

As one would expect from the notion of measurement as counting, negative values and the neutral elements of (+) for lengths, times and masses are not suitable reference quantities: in building an arbitrary system of units in the LTM class with the *Ref* constructor, these conditions are checked with the "internal" *val* helper.

[6] See https://en.wikipedia.org/wiki/Metre.

Perhaps not surprisingly, the approach requires a mutual definition of units of measurement and physical quantities but otherwise presents no difficulties. The table that returns the scaling factors between units of measurement *fs* and the measure function μ also have to be defined mutually:

```
partial fs : Units → Units → ℝ³₊
fs  SI          SI          = [1, 1, 1]
fs  SI          (Ref l t m) = [1.0 / (μ SI l), 1.0 / (μ SI t), 1.0 / (μ SI m)]
fs (Ref l t m)  SI          = [μ SI l, μ SI t, μ SI m]
fs (Ref l₁ t₁ m₁) (Ref l₂ t₂ m₂) = [μ SI l₁ / μ SI l₂, μ SI t₁ / μ SI t₂, μ SI m₁ / μ SI m₂]

partial μ : {d : D} → Units → Q d → ℝ
μ {d} u' (Val u x) = x · df d (fs u u')
```

The mutual definition prevents the (necessarily conservative) Idris termination checker to establish the totality of *fs* and μ which is why these functions are marked as partial.

Apart from these technicalities, the implementation of *dim* and the definition of new systems of units and of physical quantities in terms of arbitrary reference lengths, times and masses are straightforward, see the literate Idris code that generates this document [10].

A variation on the mutual approach sketched above could be to avoid introducing *SI* as an outstanding system of units altogether and instead equip the data type of physical quantities with three reference constructors for a length, a time and a mass. For the rest of this and of the next section, we stick to the minimal DSL with just two codes for systems of units.

5.5 Dimensional (in)dependence

Dependence. Remember that the Pi theorem is about two properties of a generic "physical relationship" f between a "dimensional quantity" a and $k + m$ "dimensional governing parameters" $a_1, \ldots, a_k$ and $b_1, \ldots, b_m$.

One property (conclusion of the theorem) is that "the dimension of a can be expressed as a product of the powers of the dimensions of the parameters $a_1, \ldots, a_k$" as formulated in eq. (12). We have just seen examples of physical quantities whose dimension functions can be expressed as products of powers of the dimension functions of other physical quantities:

$$[l] = [g][\tau]^2, \quad [g] = [l][\tau]^{-2}, \quad \text{or} \quad [\tau] = [l]^{1/2}[g]^{-1/2}$$

In the *D*-language, we can express and assert the first equality with

```
check₆ : Is l (dim g `Times` (dim τ `Times` dim τ))
check₆ = Refl
```

or, equivalently

```
check₇ : Is l (dim g `Times` Pow (dim τ) 2)
check₇ = Refl
```

and similarly for the second equality.

As already mentioned, formulating $[\tau] = [l]^{1/2}[g]^{-1/2}$ would require fractional exponents and extending our representation of dimension functions to vectors of rational numbers. In other words: the exponents in eq. (11) (and those in eq. (12)) are, in general, rational numbers and representing dimension functions in terms of vectors of rational numbers is perhaps the most natural setting for formulating the Pi theorem in type theory.

The drawback of this approach is that it requires implementing rational numbers in type theory. This is not a problem in principle, but integers have a simpler algebraic structure (initial ring) and more efficient implementations. Also, formulating the Pi theorem in terms of rational exponents does not seem strictly necessary: if dimensional analysis with rational exponents allows one to deduce that the period τ of a simple pendulum scales with $(l/g)^{\frac{1}{2}}$, integer-based dimensional analysis should be enough to deduce that τ^2 scales with l/g! We explore this possibility in the next section.

Formalizing dependence. To this end, let's formalize the notion that $d : D$ is dependent on $ds : Vec\ k\ D$ iff a non-zero integer power of d can be expressed as a product of integer powers of ds:

```
IsDep : {k : ℕ} → (d : D) → (ds : Vec k D) → Type
IsDep {k} d ds = Exists (ℤ, Vec k ℤ) (λ(p, ps) → (Not (p = 0), Pow d p = ProdPows ds ps))
```

Therefore, in the *D*-language, dependence between dimension functions boils down to linear dependence between their representations, as one would expect. By extension, we say that a physical quantity q is *dimensionally dependent* on a vector of physical quantities qs if $dim\ q$ is dependent on the dimensions of qs:

```
IsDimDep : {d : D} → {k : ℕ} → {ds : Vec k D} → Q d → Vec_Q k ds → Type
IsDimDep {d} {ds} q qs = IsDep d ds
```

Notice that $IsDimDep\ a\ as$ is an existential type. In order to assess that a is dimensionally dependent on as, one has to provide suitable integer exponents and an equality proof. For the simple pendulum, for example:

```
check₈ : IsDimDep τ [l, g]
check₈ = Evidence (2, [1, −1]) (not2eq0, Refl)
```

where *not2eq0* is a proof that 2 is not equal to 0. This evidence (*Evidence* is the data constructor of *Exists*, a data type representing dependent pairs first used in the definition of *IsDep*) is just another way of asserting the equality

```
check₉ : Pow (dim τ) 2 = Pow (dim l) 1 `Times` Pow (dim g) (−1)
check₉ = Refl
```

For the simple pendulum example, it allows one to deduce that, under quite general assumptions[7], the period of oscillation τ is proportional to the square root of l/g.

Forming dimensionless products. Given a physical quantity which is dimensionally dependent on other physical quantities, one can make it dimensionless like the "Π" quantities of the Pi theorem:

```
makeDimLess : {d : D} → {k : ℕ} → {ds : Vec k D} →
                (q : Q d) → (qs : Vec_Q k ds) → IsDimDep q qs → Q DimLess
makeDimLess {d} {ds} q qs (Evidence (p, ps) prf) =
  let comp = pow q p / prodPows qs ps
  in replace (dimMakeDimLessCompIsDimLess d ds (Evidence (p, ps) prf)) comp
```

[7]The assumptions are that the period of oscillation of the pendulum only depends on its mass, its length, the acceleration of gravity and the initial angle, see for example the introduction of [3].

In *makeDimLess*, we have applied the function *dimMakeDimLessCompIsDimLess*. This takes a dimension d, a vector of dimensions ds and evidence $e : IsDep\ d\ ds$. It returns a proof that $comp = pow\ q\ p \mid prodPows\ qs\ ps$ is indeed dimensionless. Implementing this proof requires proving that d ‘*Over*‘ d is equal to *DimLess* for arbitrary d:

$$dodIsDimLess : \{d : D\} \to d\ \text{`}Over\text{`}\ d = DimLess$$

This is straightforward for our definition of D and suggests that D-values form group. See McBride and Nordvall-Forsberg [41] and section 7 for a discussion of the algebraic structure of D.

Independence. Remember that a condition for the function f of the Pi theorem to be reducible to the form of eq. (13) is that the parameters $a_1, \ldots, a_k$ “have independent dimensions”. Perhaps not surprisingly, the idea is that $qs : Vec_{\mathbb{Q}}\ k\ ds$ are dimensionally independent iff expressing *DimLess* as a product of powers of their dimension functions requires all exponents to be zero. This is equivalent to saying the vectors associated with their dimensions are linearly independent, see [3, Section 1.1.5]:

$$\begin{aligned}
&AreDimIndep : \{k : \mathbb{N}\} \to \{ds : Vec\ k\ D\} \to Vec_{\mathbb{Q}}\ k\ ds \to Type\\
&AreDimIndep\ \{ds\}\ _ = AreIndep\ ds
\end{aligned}$$

In the definition of *AreDimIndep* we have applied the predicate *AreIndep*. In mechanics, $n = 3$ and $D = Vec\ 3\ \mathbb{Z}$, so that *AreIndep* is decidable

$$\begin{aligned}
&AreIndep : \{k : \mathbb{N}\} \to Vec\ k\ (Vec\ 3\ \mathbb{Z}) \to Type\\
&AreIndep\ vs = areLinIndep\ vs = True
\end{aligned}$$

and a decision procedure can be implemented by pattern matching on the the length of vs

$$\begin{aligned}
&areLinIndep : \{k : \mathbb{N}\} \to Vec\ k\ (Vec\ 3\ \mathbb{Z}) \to Bool\\
&areLinIndep\ Nil &&= True\\
&areLinIndep\ (v_1 :: Nil) &&= \neg\ (v_1 == [0,0,0])\\
&areLinIndep\ (v_1 :: v_2 :: Nil) &&= \neg\ (areCollinear\ v_1\ v_2)\\
&areLinIndep\ (v_1 :: v_2 :: v_3 :: Nil) &&= \neg\ (det\ v_1\ v_2\ v_3 == 0)\\
&areLinIndep\ (v_1 :: v_2 :: v_3 :: v_4 :: vs) &&= False
\end{aligned}$$

where *areCollinear* and *det* are defined in terms of the standard dot and cross products in $Vec\ 3\ \mathbb{Z}$. Thus, *AreDimIndep* can be applied to assess the dimensional independence of physical quantities:

$$\begin{aligned}
&check_{11} : AreDimIndep\ [l, g]\\
&check_{11} = Refl\\
&check_{12} : Not\ (AreDimIndep\ [\tau, l, g])\\
&check_{12}\ Refl\ impossible
\end{aligned}$$

We have encoded the notions of dimension function, physical quantity, measurement, units of measurement and dimensional (in)dependence for classical mechanics (and its subdomains) in Idris and Agda[8]. With this minimal DSL, we can express dimensional judgments and implement dimensionally consistent programs.

[8] See `DSL1.agda` in the `agda` folder of [10].

In the next section, we apply the DSL to formulate the covariance principle and the Pi theorem in type theory, discuss its non-constructive nature and present a method for applying the theorem to compute functions that fulfil the covariance principle by construction.

6 The covariance principle and verified applications of the Pi theorem

We start by formalizing the covariance principle and the Pi theorem as formulated in section 4. This strictly follows Barenblatt et al. [3] which, in turn, follows those of Bridgman [15], Buckingham [16], Rayleigh [49] and of standard DA textbooks. Then, in section 6.2 we discuss the non-constructive nature of classical proofs and, in section 6.3, we exploit the formalization of section 6.1 to derive a method for "applying" the Pi theorem (as this is routinely done in mathematical physics and modelling) in a verified type theoretical setting.

6.1 The covariance principle and the Pi theorem

Following the principle that types can encode logical propositions [29], our first step is to define a type, say *Pi*, such that values of type *Pi* represent proofs of the Pi theorem.

In section 4.1 we saw that the Pi theorem asserts that a "physical relationship" f between a "dimensional quantity" a and $k+m$ "dimensional governing parameters" $a_1, \ldots, a_k$ and $b_1, \ldots, b_m$ that fulfil eqs. (10) and (11) satisfies eqs. (12) and (13). Thus the theorem entails two conclusions, both quantified over functions between physical quantities. We account for this through two higher-order function types $Pi_{12}, Pi_{13} : Type$

$$Pi_{12} = (f : Vec_{\mathrm{Q}}\ k\ ds \to Vec_{\mathrm{Q}}\ m\ ds' \to Q\ d) \to \ldots$$

and similarly for Pi_{13} with implicit parameters $k, m : \mathbb{N}$, $ds : Vec\ k\ D$, $ds' : Vec\ m\ D$ and $d : D$. As discussed in section 4.1, the term "physical relationship" is used by Barenblatt et al. [3] to denote a function that fulfils the "covariance principle".

We have seen in section 5.1 that the covariance principle (or principle of relativity of measurements) posits that there is no privileged system of units of measurement or, equivalently, that all systems are equally good. So far, we have formalized the notion of covariance for dimension functions (through the specification 14) and we have argued that, for elementary binary operations on physical quantities, the covariance principle boils down to the requirement that μ_u is a homomorphism, see section 5.2.

In general, a function $f : Vec_{\mathrm{Q}}\ m\ ds \to Q\ d$ fulfils the covariance principle iff there exists a representation $\rho_f : Vec\ m\ \mathbb{R} \to \mathbb{R}$ such that the diagram in fig. 1 commutes. Here $map_{\mathrm{Q}}\ \mu_u$

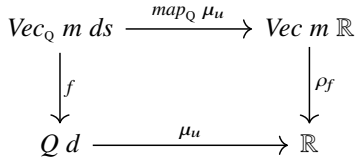

Fig. 1. The covariance principle (or principle of relativity of measurements) for a function between physical quantities.

is the function that applies μ_u to the physical quantities of a Vec_{Q} and u is an arbitrary system of units of measurement. We can specialize this notion to binary operations between physical quantities, for example multiplication

$$\begin{aligned}
&\mathit{isCovariantMult} : \mathit{Exists}\ (\mathbb{R} \to \mathbb{R} \to \mathbb{R}) \\
&\qquad (\lambda\rho \Rightarrow (u : \mathit{Units}) \to \{d_1, d_2 : D\} \to \\
&\qquad\quad (q_1 : Q\ d_1) \to (q_2 : Q\ d_2) \to \\
&\qquad\quad \mu_u\ (q_1 \cdot q_2) = \rho\ (\mu_u\ q_1)\ (\mu_u\ q_2))
\end{aligned}$$

and apply the results from section 5.2 to show that multiplication between physical quantities is indeed covariant:

$$\mathit{isCovariantMult} = \mathit{Evidence}\ (\cdot)\ \mu\mathit{HomMult}$$

Similarly, we can encode the requirement that the physical relationship f of the Pi theorem fulfils the covariance principle in terms of a predicate

$$\begin{aligned}
&\mathit{IsCovariant} : \{k, m : \mathbb{N}\} \to \{ds : \mathit{Vec}\ k\ D\} \to \{ds' : \mathit{Vec}\ m\ D\} \to \{d : D\} \to \\
&\qquad (f : \mathit{Vec}_{\mathrm{Q}}\ k\ ds \to \mathit{Vec}_{\mathrm{Q}}\ m\ ds' \to Q\ d) \to \mathit{Type} \\
&\mathit{IsCovariant}\ \{k\}\ \{m\}\ \{ds\}\ \{ds'\}\ \{d\}\ f = \\
&\quad \mathit{Exists}\ (\mathit{Vec}\ k\ \mathbb{R} \to \mathit{Vec}\ m\ \mathbb{R} \to \mathbb{R}) \\
&\qquad (\lambda\rho_f \Rightarrow (u : \mathit{Units}) \to (as : \mathit{Vec}_{\mathrm{Q}}\ k\ ds) \to (bs : \mathit{Vec}_{\mathrm{Q}}\ m\ ds') \to \\
&\qquad\quad \mu_u\ (f\ as\ bs) = \rho_f\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as)\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ bs))
\end{aligned}$$

and apply *IsCovariant* to encode the first assumption of the theorem:

$$\mathit{Pi}_{12} = (f : \mathit{Vec}_{\mathrm{Q}}\ k\ ds \to \mathit{Vec}_{\mathrm{Q}}\ m\ ds' \to Q\ d) \to (h_1 : \mathit{IsCovariant}\ f) \to \ldots$$

Next, we need to formalize the two assumptions eqs. (10) and (11) about the arguments of f. The first one states that $a_1, \ldots, a_k$ "have independent dimensions". We have seen how to formalize this assumption in section 5.5:

$$\begin{aligned}
&\mathit{Pi}_{12} = (f : \mathit{Vec}_{\mathrm{Q}}\ k\ ds \to \mathit{Vec}_{\mathrm{Q}}\ m\ ds' \to Q\ d) \to (h_1 : \mathit{IsCovariant}\ f) \to \\
&\qquad (h_2 : \mathit{AreIndep}\ ds) \to \ldots
\end{aligned}$$

The second assumption of the Pi theorem, eq. (11), specifies m equalities between dimension functions. We have seen that equality between dimension functions boils down to equality in $\mathbb{Z}^n$ (in mechanics $n = 3$) and is thus decidable.

In section 5.5, we have also seen that the exponents in eq. (11) are rational numbers and that we can rewrite these equalities as

$$[b_i]^{p_i} = [a_1]^{p_{i1}} \ldots [a_k]^{p_{ik}} \quad i = 1, \ldots, m \tag{19}$$

with integers $p_i, p_{i,1}, \ldots, p_{i,k}$ as we have done for the simple pendulum example. This states that the dimension functions of the physical quantities of the second argument bs of f can be expressed as products of powers of the dimension functions of the physical quantities of the first argument

$$\begin{aligned}
&\mathit{Pi}_{12} = (f : \mathit{Vec}_{\mathrm{Q}}\ k\ ds \to \mathit{Vec}_{\mathrm{Q}}\ m\ ds' \to Q\ d) \to (h_1 : \mathit{IsCovariant}\ f) \to \\
&\qquad (h_2 : \mathit{AreIndep}\ ds) \to (h_3 : \mathit{AreDep}\ ds'\ ds) \to \ldots
\end{aligned}$$

where $h_3 : \mathit{AreDep}\ ds'\ ds$ is a vector of *IsDep* proofs, one for each element of ds':

$$\begin{aligned}
&\mathit{AreDep} : \{m, k : \mathbb{N}\} \to (ds' : \mathit{Vec}\ m\ D) \to (ds : \mathit{Vec}\ k\ D) \to \mathit{Type} \\
&\mathit{AreDep}\ ds'\ ds = \mathit{All}\ (\lambda d' \Rightarrow \mathit{IsDep}\ d'\ ds)\ ds'
\end{aligned}$$

In the *Pi* theorem, the parameters that are turned into dimensionless quantities are $b_1, \ldots, b_m$ (with dimensions ds'). They are dimensionally dependent on the dimensionally independent

ones $a_1, \ldots, a_k$ (with dimensions *ds*). From this angle, the two hypotheses h_2 : *AreIndep ds* and h_3 : *AreDep ds′ ds* completely determine how the $k + m$ parameters are split. In general there can be several choices, and the user of the *Pi* theorem decides.

With these premises, the Pi theorem warrants the existence of exponents $p_1, \ldots, p_k$ and of a function Φ such that the equalities (12) and (13) hold. As for eq. (19), these are rational numbers but we can reformulate eq. (12) as:

$$[a]^p = [a_1]^{p_1} \ldots [a_k]^{p_k} \tag{20}$$

with integers exponents $p, p_1, \ldots, p_k$ and the first conclusion of the Pi theorem (with all its implicit arguments) as

$$\begin{aligned}
&Pi_{12} \ : \ Type \\
&Pi_{12} = \{k, m \ : \ \mathbb{N}\} \to \{ds \ : \ Vec\ k\ D\} \to \{ds' \ : \ Vec\ m\ D\} \to \{d \ : \ D\} \to \\
&\qquad (f \ : \ Vec_{\mathrm{Q}}\ k\ ds \to Vec_{\mathrm{Q}}\ m\ ds' \to Q\ d) \to (h_1 \ : \ IsCovariant\ f) \to \\
&\qquad (h_2 \ : \ AreIndep\ ds) \to (h_3 \ : \ AreDep\ ds'\ ds) \to IsDep\ d\ ds
\end{aligned}$$

The second conclusion of the Pi theorem is eq. (13). This states the existence of a function Φ that allows one to express *f as bs* to the power of *p* as a product of powers of the *as* times Φ applied to the dimensionless "Π" fractions of eq. (13):

$$\begin{aligned}
&Pi_{13} \ : \ Type \\
&Pi_{13} = \{k, m \ : \ \mathbb{N}\} \to \{ds \ : \ Vec\ k\ D\} \to \{ds' \ : \ Vec\ m\ D\} \to \{d \ : \ D\} \to \\
&\qquad (f \ : \ Vec_{\mathrm{Q}}\ k\ ds \to Vec_{\mathrm{Q}}\ m\ ds' \to Q\ d) \to (h_1 \ : \ IsCovariant\ f) \to \\
&\qquad (h_2 \ : \ AreIndep\ ds) \to (h_3 \ : \ AreDep\ ds'\ ds) \to \\
&\qquad Exists\ (Vec_{\mathrm{Q}}\ m\ (replicate\ m\ DimLess) \to Q\ DimLess) \\
&\qquad\qquad (\lambda\Phi \Rightarrow (as \ : \ Vec_{\mathrm{Q}}\ k\ ds) \to (bs \ : \ Vec_{\mathrm{Q}}\ m\ ds') \to \\
&\qquad\qquad\qquad \mathbf{let}\ (p, ps) = fst\ (\pi_{12}\ f\ h_1\ h_2\ h_3) \\
&\qquad\qquad\qquad\quad \Pi s \quad\ = makeAllDimLess\ bs\ as\ h_3 \\
&\qquad\qquad\qquad \mathbf{in}\ \ pow\ (f\ as\ bs)\ p = prodPows\ as\ ps \cdot \Phi\ \Pi s)
\end{aligned}$$

Thus, both conclusions of the Pi theorem are existential types (remember the definition of *IsDep d ds* from section 5.5) and the second depends on the first through the integer exponents $p, p_1, \ldots, p_k$ denoted, in the code as (p, ps). The "Π" fractions Πs are computed by mapping *makeDimLess* from section 5.5 on $b_1, \ldots, b_m$ alias *bs*, see the Idris and Agda code in [10]. Alternatively, one could combine the two conclusions in a single, nested existential type.

6.2 The non-constructive nature of the Pi theorem

Before discussing the non-constructive nature of the Pi theorem, it is worth pointing out two difficulties of the textbook formulation discussed above. The first one is technical. In the equality at the last line of the definition of Pi_{13}

$$pow\ (f\ as\ bs)\ p = prodPows\ as\ ps \cdot \Phi\ \Pi s$$

the left and the right hand side have different types. The left-hand side has type $Q\ (d\ `Pow`\ p)$ where d is the return type of f. The right hand side has type $Q\ (ProdPows\ ds\ ps\ `Times`\ DimLess)$. The second projection of the second projection of $\pi_{12}\ f\ h_1\ h_2\ h_3$ is a witness of the equality between these two types which needs to be applied in order to formulate the Pi theorem in homogeneous implementations of identity types [40]. For example, in Agda with homogeneous equality the above formalization of the Pi theorem reads

$$
\begin{aligned}
&Pi_{13} \;:\; Set \quad \text{-- Agda version}\\
&Pi_{13} = \{k\ m : \mathbb{N}\} \to \{ds : Vec\ D\ k\} \to \{ds' : Vec\ D\ m\} \to \{d : D\} \to\\
&\quad (f : Vec_{\mathrm{Q}}\ k\ ds \to Vec_{\mathrm{Q}}\ m\ ds' \to Q\ d) \to (h_1 : IsCovariant\ f) \to\\
&\quad (h_2 : AreIndep\ ds) \to (h_3 : AreDep\ ds'\ ds) \to\\
&\quad \Sigma\ (Vec_{\mathrm{Q}}\ m\ (replicate\ DimLess) \to Q\ DimLess)\\
&\qquad (\lambda\Phi \to (as : Vec_{\mathrm{Q}}\ k\ ds) \to (bs : Vec_{\mathrm{Q}}\ m\ ds') \to\\
&\qquad\quad \mathbf{let}\ ddds : IsDep\ d\ ds\\
&\qquad\qquad ddds = \pi_{12}\ f\ h_1\ h_2\ h_3\\
&\qquad\qquad p \;\; : \mathbb{Z}\\
&\qquad\qquad p \;\; = \pi_1\ (\pi_1\ ddds)\\
&\qquad\qquad ps \;\; : Vec\ \mathbb{Z}\ k\\
&\qquad\qquad ps \;\; = \pi_2\ (\pi_1\ ddds)\\
&\qquad\qquad prf_1 \;\; : Pow\ d\ p \equiv ProdPows\ ds\ ps\\
&\qquad\qquad prf_1 \;\; = \pi_2\ (\pi_2\ ddds)\\
&\qquad\qquad prf_2 \;\; : Pow\ d\ p \equiv ProdPows\ ds\ ps\ Times\ DimLess\\
&\qquad\qquad prf_2 \;\; = trans\ prf_1\ (sym\ (dTimesDimLessIsd\ (ProdPows\ ds\ ps)))\\
&\qquad\qquad \Pi s = makeAllDimLess\ bs\ as\ h_3\\
&\qquad\qquad \mathbf{in}\ pow\ (f\ as\ bs)\ p\\
&\qquad\qquad\quad \equiv\\
&\qquad\qquad\quad transport\ \{P = Q\}\ (sym\ prf_2)\ (prodPows\ as\ ps\ Q \cdot Q\ \Phi\ \Pi s))
\end{aligned}
$$

This is in principle not a problem. But it suggests that a one-to-one translation (modulo integer arithmetic) of the textbook formulation eq. (13) of the Pi theorem is perhaps inadequate.

The second difficulty is conceptual. The Pi theorem starts from the assumption that a "physical relationship" f that fulfills the covariance principle exists between certain physical variables. In the context of applications of the Pi theorem (Galileo, Newton, Fourier, Maxwell, Reynolds and Rayleigh) until its formulation in 1914 [16, 17] and until today, the existence of f has never been understood to be evidential: f is a sought physical law (like Newton's second law or like the ideal gas law from section 2.2) that needs to be identified by empirical experiments or from first principles.

In this context the Pi theorem merely states that *if* f would exist, it would have to satisfy certain dimensional requirements and its form would be constrained as expressed by eq. (13). In other words: even if we were given the p and ps and evidence that d `Pow` p is equal to $ProdPows\ ds\ ps$ `Times` $DimLess$, that is, evidence $\pi_{12} : Pi_{12}$, it would make little sense to "apply" the Pi theorem to "compute" the shape function Φ because the input f, in fact, is not known.

This observation, too, suggests that applying the Pi theorem in type theory should not be understood as the act of implementing Pi_{12} and Pi_{13} and then apply these functions to suitable arguments: at least one of these arguments, namely the function f is what we are trying to identify.

Indeed, proofs of the Pi theorem are typically non-constructive and thus Pi_{12} is not implementable. For example, the one sketched in Section 1 of [3] starts from the linear algebra lemma

Lemma 1. *Let* $as = a_1, \ldots, a_k$ *have independent dimensions. Then for any system of units* u*, any positive real number* r *and any* $1 \leqslant j \leqslant k$ *there exists a system of units* u_j *such that* $\mu\ u_j\ a_j = r \cdot \mu\ u\ a_j$ *and* $\mu\ u_j\ a_i = \mu\ u\ a_i$ *for all* $1 \leqslant i \leqslant k$, $i \neq j$.

and goes on as follows. Let $bs = b_1, \ldots, b_m$. Assume $f\ (as, bs)$ and as are dimensionally independent and take $r \neq 1$. Then one has

$$\mu_{u_0}\ (f\ (as, bs)) = r \cdot \mu_u\ (f\ (as, bs))$$

by lemma 1 and also

$$\mu_{u_0}\ (f\ (as, bs)) = \rho_f\ (map_{\mathrm{Q}}\ \mu_{u_0}\ as, map_{\mathrm{Q}}\ \mu_{u_0}\ bs)$$

by the covariance principle. But by lemma 1 one also has $map_{\mathrm{Q}}\ \mu_{u_0}\ as = map_{\mathrm{Q}}\ \mu_u\ as$, and also $map_{\mathrm{Q}}\ \mu_{u_0}\ bs = map_{\mathrm{Q}}\ \mu_u\ bs$ (because bs are dimensionally dependent on as) and thus

$$\mu_{u_0}\ (f\ (as, bs)) = \rho_f\ (map_{\mathrm{Q}}\ \mu_u\ as, map_{\mathrm{Q}}\ \mu_u\ bs)$$

by congruence and, again by the covariance principle $\mu_{u_0}\ (f\ (as, bs)) = \mu_u\ (f\ (as, bs))$ and thus $r = 1$. Contradiction. Thus, it is not the case that $f\ (as, bs)$ and as have independent dimensions.

Even if lemma 1 is implementable, the proof sketched by Barenblatt et al. [3] is non-constructive. One can derive a contradiction from the assumption that $f\ (as, bs)$ and as have independent dimensions and thus implement a function of type $\neg\ (IsDep\ d\ ds) \to Void$.

But such a function does not allow one to produce a value of type $IsDep\ d\ ds$. What is missing is double negation elimination or, equivalently, the excluded middle.

6.3 *Verified applications of the Pi theorem*

The lack of implementable formulations of the Pi theorem should neither be surprising nor concerning. In order to support applying the theorem as this is routinely done in mathematical physics and modelling (remember the discussion about Stommel's paper in section 2.2) in a safe type theoretical setting, one does not need to be able to implement Pi_{12}, not to mention Pi_{13}.

We only need to recognize that the Pi theorem, while not implementable in type theory, encodes important restrictions on the return type and on the form of functions of physical quantities and provide users with a verified method for enforcing these restrictions.

A way of doing so that does not require modifications of the host language can be defined in two steps. First, encode the type of functions of physical quantities as constrained by the Pi theorem through a data type:

$$\begin{aligned}
&\textbf{data}\ PhysFunType : (k, m : \mathbb{N}) \to Vec\ k\ D \to Vec\ m\ D \to D \to Type\ \textbf{where}\\
&\quad PFT : \{k : \mathbb{N}\} \to \{ds : Vec\ k\ D\} \to \{m : \mathbb{N}\} \to \{ds' : Vec\ m\ D\} \to \{d : D\} \to\\
&\qquad AreIndep\ ds \to AreDep\ ds'\ ds \to IsDep\ d\ ds \to PhysFunType\ k\ m\ ds\ ds'\ d
\end{aligned}$$

For the second step, it is useful to define the projections

$$\begin{aligned}
&areDep : \{k : \mathbb{N}\} \to \{ds : Vec\ k\ D\} \to \{m : \mathbb{N}\} \to \{ds' : Vec\ m\ D\} \to \{d : D\} \to\\
&\qquad PhysFunType\ k\ m\ ds\ ds'\ d \to AreDep\ ds'\ ds\\
&areDep\ (PFT\ h_2\ h_3\ \pi_{12}) = h_3
\end{aligned}$$

$$\begin{aligned}
&isDep : \{k : \mathbb{N}\} \to \{ds : Vec\ k\ D\} \to \{m : \mathbb{N}\} \to \{ds' : Vec\ m\ D\} \to \{d : D\} \to\\
&\qquad PhysFunType\ k\ m\ ds\ ds'\ d \to IsDep\ d\ ds\\
&isDep\ (PFT\ h_2\ h_3\ \pi_{12}) = \pi_{12}
\end{aligned}$$

and a function that computes the return type of functions of physical quantities as encoded by *PhysFunType* values:

$$
\begin{aligned}
&\mathit{RetType} : \{k : \mathbb{N}\} \to \{ds : \mathit{Vec}\ k\ D\} \to \{m : \mathbb{N}\} \to \{ds' : \mathit{Vec}\ m\ D\} \to \{d : D\} \to \\
&\qquad \mathit{PhysFunType}\ k\ m\ ds\ ds'\ d \to \mathit{Type} \\
&\mathit{RetType}\ \{ds\}\ \mathit{pft} = Q\ (\mathit{ProdPows}\ ds\ (\mathit{snd}\ (\mathit{fst}\ (\mathit{isDep}\ \mathit{pft}))))
\end{aligned}
$$

With these computations in place, we define an *applyPi* function that constructs functions of physical quantities that are consistent with the Pi theorem from arbitrary shape functions:

$$
\begin{aligned}
&\mathit{applyPi} : \{k : \mathbb{N}\} \to \{ds : \mathit{Vec}\ k\ D\} \to \{m : \mathbb{N}\} \to \{ds' : \mathit{Vec}\ m\ D\} \to \{d : D\} \to \\
&\qquad (\mathit{pft} : \mathit{PhysFunType}\ k\ m\ ds\ ds'\ d) \to (\mathit{Vec}\ m\ \mathbb{R} \to \mathbb{R}) \to \\
&\qquad (\mathit{Vec}_{\mathrm{Q}}\ k\ ds \to \mathit{Vec}_{\mathrm{Q}}\ m\ ds' \to \mathit{RetType}\ \mathit{pft}) \\
&\mathit{applyPi}\ \{k\}\ \{ds\}\ \{m\}\ \{ds'\}\ \{d\}\ \mathit{pft}\ \varphi\ \mathit{as}\ \mathit{bs} = \\
&\quad \textbf{let}\ \mathit{ps} = \mathit{snd}\ (\mathit{fst}\ (\mathit{isDep}\ \mathit{pft})) \\
&\qquad \mathit{pis} = \mathit{map}_{\mathrm{Q}}\ \mu_{SI}\ (\mathit{makeAllDimLess}\ \mathit{bs}\ \mathit{as}\ (\mathit{areDep}\ \mathit{pft})) \\
&\quad \textbf{in}\ \mathit{prodPows}\ \mathit{as}\ \mathit{ps} \triangleright \varphi\ \mathit{pis}
\end{aligned}
$$

The construction follows directly from the formalization of the Pi theorem of section 6.1, see the definition of Pi_{13}.

Notice that, because the minimal DSL from section 5 represents dimension functions in terms of integer exponents, the physical quantity $\mathit{applyPi}\ (\mathit{PFT}\ h_2\ h_3\ \pi_{12})\ \varphi\ \mathit{as}\ \mathit{bs}$ is actually the $\mathit{fst}\ (\mathit{fst}\ h_2)$ power of $f\ \mathit{as}\ \mathit{bs}$ as one can see from the definition of Pi_{13}.

Notice also that the first argument taken by the data constructor *PFT*, a proof that the physical quantities *as* in *applyPi pft* φ *as bs* are dimensionally independent, is not used in the definition of *applyPi*. Its role is to prevent applications of the Pi theorem that yield functions of physical quantities that are not consistent with the Pi theorem.

The second argument of *applyPi* is a "shape function" φ. Its type is $\mathit{Vec}\ m\ \mathbb{R} \to \mathbb{R}$ rather than $\mathit{Vec}_{\mathrm{Q}}\ m\ (\mathit{replicate}\ m\ \mathit{DimLess}) \to Q\ \mathit{DimLess}$ as in the Pi theorem, see Pi_{13}. This is because of two reasons: the first one is that functions of type $\mathit{Vec}_{\mathrm{Q}}\ m\ (\mathit{replicate}\ m\ \mathit{DimLess}) \to Q\ \mathit{DimLess}$ that respect the covariance principle are in fact just functions of type $\mathit{Vec}\ m\ \mathbb{R} \to \mathbb{R}$. The second reason is that, in applications of the Pi theorem, the shape function is what needs to be identified from first principles or approximated from empirical data, often via statistical methods, approximation theory or machine learning. These methodologies yield functions of real variables, not of physical quantities.

The idea of the approach encoded by *PhysFunType-applyPi* is that it should yield *verified* functions of physical quantities

$$
\begin{aligned}
&\mathit{applyPiLemma} : \{k : \mathbb{N}\} \to \{ds : \mathit{Vec}\ k\ D\} \to \{m : \mathbb{N}\} \to \{ds' : \mathit{Vec}\ m\ D\} \to \{d : D\} \to \\
&\qquad (\mathit{pft} : \mathit{PhysFunType}\ k\ m\ ds\ ds'\ d) \to (\varphi : \mathit{Vec}\ m\ \mathbb{R} \to \mathbb{R}) \to \\
&\qquad \mathit{IsCovariant}\ (\mathit{applyPi}\ \mathit{pft}\ \varphi)
\end{aligned}
$$

Proving *applyPiLemma* is tedious but conceptually straightforward: we have seen in section 6.1 that multiplication between physical variables fulfills the covariance principle: *isCovariantMult*. The same holds for division. Integer powers and products of integer powers are just iterated multiplication and division. Making a physical quantity dimensionless is again division (of that quantity) by a suitable products of integer powers (of other physical quantities).

Thus, the whole computation $\mathit{prodPows}\ \mathit{as}\ \mathit{ps} \triangleright \varphi\ \mathit{pis}$ is a combination of multiplication and division between physical quantities and scaling and in section 5.2 we have seen that these operations fulfil the covariance principle. Indeed, it is not difficult to prove

$$
\begin{aligned}
&\mathit{powIsCovariant} : (p : \mathbb{Z}) \to \\
&\quad \mathit{Exists}\ (\mathbb{R} \to \mathbb{R})\ (\lambda\rho \Rightarrow (u : \mathit{Units}) \to \{d : D\} \to (q : Q\ d) \to \mu_u\ (\mathit{pow}\ q\ p) = \rho\ (\mu_u\ q)) \\
&\mathit{prodPowsIsCovariant} : \{k : \mathbb{N}\} \to (ps : \mathit{Vec}\ k\ \mathbb{Z}) \to \\
&\quad \mathit{Exists}\ (\mathit{Vec}\ k\ \mathbb{R} \to \mathbb{R})\ (\lambda\rho \Rightarrow (u : \mathit{Units}) \to \{ds : \mathit{Vec}\ k\ D\} \to (as : \mathit{Vec}_{\mathrm{Q}}\ k\ ds) \to \\
&\qquad\qquad \mu_u\ (\mathit{prodPows}\ as\ ps) = \rho\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as)) \\
&\mathit{makeAllDimLessIsCovariant} : \{k : \mathbb{N}\} \to \{ds : \mathit{Vec}\ k\ D\} \to \\
&\quad \{m : \mathbb{N}\} \to \{ds' : \mathit{Vec}\ m\ D\} \to (\mathit{areDep} : \mathit{AreDep}\ ds'\ ds) \to \\
&\quad \mathit{Exists}\ (\mathit{Vec}\ k\ \mathbb{R} \to \mathit{Vec}\ m\ \mathbb{R} \to \mathit{Vec}\ m\ \mathbb{R}) \\
&\qquad (\lambda\rho \Rightarrow (u : \mathit{Units}) \to (as : \mathit{Vec}_{\mathrm{Q}}\ k\ ds) \to (bs : \mathit{Vec}_{\mathrm{Q}}\ m\ ds') \to \\
&\qquad\quad \mathit{map}_{\mathrm{Q}}\ \mu_u\ (\mathit{makeAllDimLess}\ bs\ as\ \mathit{areDep}) = \rho\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as)\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ bs))
\end{aligned}
$$

The proofs depend on four results from section 5: *powdfLemma*, *ipowHomMult*, *μHomMult*, *dfDimLessLemma*. The first two are postulated in Idris and fully implemented in Agda.

In implementing *makeAllDimLessIsCovariant* in Idris we have run into an issue with the type checker but the Agda proof is fully implemented. It is perhaps worth mentioning that, in Agda with homogeneous Martin-Löf equality, implementing *makeAllDimLessIsCovariant* involves proving

$$\mu\ \{d = \mathit{DimLess}\}\ u\ (\mathit{transport}\ \{P = Q\}\ \mathit{dmdd}\ \mathit{comp}) \equiv \mu\ \{d = \mathit{dmd}\}\ u\ \mathit{comp}$$

where $\mathit{dmdd} : \mathit{dmd} \equiv \mathit{DimLess}$, and $\mathit{comp} = \mathit{pow}\ b\ p\ /\ \mathit{prodPows}\ as\ ps$ is of type $Q\ \mathit{dmd}$ (with variables interpreted as usual). This step requires congruence modulo transport

$$(p : a_1 \equiv a_2) \to f\ a_1\ Pa_1 \equiv f\ a_2\ (\mathit{transport}\ p\ Pa_1)$$

which can be implemented without UIP. Finally, *prodPowsIsCovariant* (abbreviated to *pPCov* in the code) and *makeAllDimLessIsCovariant* (abbreviated to *mADLCov*) are combined to prove that functions constructed with *applyPi* are in fact covariant:

$$
\begin{aligned}
&\mathit{applyPiLemma}\ \{k\}\ \{ds\}\ \{m\}\ \{ds'\}\ \mathit{pft}\ \varphi = \\
&\quad \mathbf{let}\ f = \mathit{applyPi}\ \mathit{pft}\ \varphi; \qquad\qquad ps = \mathit{snd}\ (\mathit{fst}\ (\mathit{isDep}\ \mathit{pft})) \\
&\qquad \rho_1 = \mathit{fst}\ (\mathit{pPCov}\ ps); \qquad\qquad \mathit{prf}_1 = \mathit{snd}\ (\mathit{pPCov}\ ps) \\
&\qquad \rho_3 = \mathit{fst}\ (\mathit{mADLCov}\ (\mathit{areDep}\ \mathit{pft})); \quad \mathit{prf}_3 = \mathit{snd}\ (\mathit{mADLCov}\ (\mathit{areDep}\ \mathit{pft})) \\
&\qquad \rho = \lambda xs \Rightarrow \lambda ys \Rightarrow \rho_1\ xs \cdot \varphi\ (\rho_3\ xs\ ys); \\
&\qquad \mathit{prf} : ((u : \mathit{Units}) \to (as : \mathit{Vec}_{\mathrm{Q}}\ k\ ds) \to (bs : \mathit{Vec}_{\mathrm{Q}}\ m\ ds') \to \\
&\qquad\qquad \mu_u\ (f\ as\ bs) = \rho\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as)\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ bs) \qquad\qquad) \\
&\qquad\quad = \lambda u \Rightarrow \lambda as \Rightarrow \lambda bs \Rightarrow \\
&\qquad\quad \mathbf{let}\ \mathit{comp} = \mathit{makeAllDimLess}\ bs\ as\ (\mathit{areDep}\ \mathit{pft}); \quad pis = \mathit{map}_{\mathrm{Q}}\ \mu_{SI}\ \mathit{comp} \\
&\qquad\quad \mathbf{in}\ \mu_u\ (f\ as\ bs) \\
&\qquad\qquad\quad =\{\ \mu\mathit{HomScalR}\ u\ (\mathit{prodPows}\ as\ ps)\ (\varphi\ pis)\ \}= \\
&\qquad\qquad (\mu_u\ (\mathit{prodPows}\ as\ ps) \cdot \varphi\ pis) \\
&\qquad\qquad\quad =\{\ \mathit{cong}\ \{f = \lambda x \Rightarrow x \cdot \varphi\ pis\}\ (\mathit{prf}_1\ u\ as)\ \}= \\
&\qquad\qquad (\rho_1\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as) \cdot \varphi\ pis) \\
&\qquad\qquad\quad =\{\ \mathit{cong}\ \{f = \lambda x \Rightarrow \rho_1\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as) \cdot \varphi\ x\}\ (\mathit{measVectDimLessLemma}\ \mathit{comp})\ \}= \\
&\qquad\qquad (\rho_1\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as) \cdot \varphi\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ \mathit{comp})) \\
&\qquad\qquad\quad =\{\ \mathit{cong}\ \{f = \lambda x \Rightarrow \rho_1\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as) \cdot \varphi\ x\}\ (\mathit{prf}_3\ u\ as\ bs)\ \}= \\
&\qquad\qquad (\rho\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ as)\ (\mathit{map}_{\mathrm{Q}}\ \mu_u\ bs)) \\
&\qquad\qquad\quad \mathit{QED} \\
&\quad \mathbf{in}\ \mathit{Evidence}\ \rho\ \mathit{prf}
\end{aligned}
$$

Notice that, in order to apply *applyPi* and construct a physical law that fulfills the covariance principle, one needs to implement a value of type *PhysFunType*.

Doing so, requires computing evidence of *AreIndep ds*, *AreDep ds′ ds* and *IsDep d ds* for the dimensions of the first and of the second argument *ds* and *ds′* of *f* and for its return type *d*. This is equivalent to proving the linear independence of *ds* and to solving $m + 1$ systems of linear equations for $\mathbb{Z}$ values. Formalizing the fragment of linear algebra necessary to accomplish this task goes well beyond the scope of this paper but we point the interested reader to [22].

A more powerful approach towards applying the Pi theorem and constructing functions of physical quantities that fulfil the covariance principle would require modifications of the host language. We discuss this possibility in section 7.3.

We conclude this section by applying the *PhysFunType* and *applyPi* approach outlined above to the pendulum example from section 5.5.

First we need to encode the type of functions that computes how the second power of the period of a pendulum depends on its mass, its length and on the acceleration of gravity:

$$
\begin{aligned}
&\mathit{tau2t} \;:\; \mathit{PhysFunType}\ 3\ 0\ [\mathit{Mass}, \mathit{Length}, \mathit{Acceleration}]\ [\,]\ \mathit{Time}\\
&\mathit{tau2t} \;=\; \mathit{PFT}\ h_2\ h_3\ \pi_{12}\ \textbf{where}\\
&\quad h_2 \;:\; \mathit{AreIndep}\ [\mathit{Mass}, \mathit{Length}, \mathit{Acceleration}]\\
&\quad h_2 \;=\; \mathit{Refl}\\
&\quad h_3 \;:\; \mathit{AreDep}\ [\,]\ [\mathit{Mass}, \mathit{Length}, \mathit{Acceleration}]\\
&\quad h_3 \;=\; [\,]\\
&\quad \pi_{12} \;:\; \mathit{IsDep}\ \mathit{Time}\ [\mathit{Mass}, \mathit{Length}, \mathit{Acceleration}]\\
&\quad \pi_{12} \;=\; \mathit{Evidence}\ (2, [0, 1, -1])\ (\mathit{not2eq0}, \mathit{Refl})
\end{aligned}
$$

Then we apply the Pi theorem and construct a function that computes the second power of the period of a pendulum:

$$
\begin{aligned}
&\mathit{tau2} \;:\; \mathit{Vec}_{\mathbb{Q}}\ 3\ [\mathit{Mass}, \mathit{Length}, \mathit{Acceleration}] \to \mathit{Vec}_{\mathbb{Q}}\ 0\ [\,] \to \mathit{RetType}\ \mathit{tau2t}\\
&\mathit{tau2} \;=\; \mathit{applyPi}\ \mathit{tau2t}\ \varphi\ \textbf{where}\\
&\quad \varphi \;:\; \mathit{Vec}\ 0\ \mathbb{R} \to \mathbb{R}\\
&\quad \varphi\ [\,] \;=\; \mathit{pow}\ (2 \cdot \pi)\ 2.0
\end{aligned}
$$

One can check that measurements of *tau2 m l g* do not depend on the mass *m* of the pendulum, are proportional to its length *l* and inversely proportional to the gravity *g*.

The minimal DSL presented in section 5, the formulations of the covariance principle and of the Pi theorem and the two-steps approach towards constructing functions of physical quantities that respect this principle are the main contributions of this paper towards making mathematical physics (functional programming) more accessible to computer scientists (modelers and physicists).

In the next section we discuss possible generalizations and desiderata that go beyond the scope of this manuscript.

7 Possible generalizations and extension

We discuss possible generalizations and extensions of the DSL for dimensionally consistent programming presented in section 5. Some of these extensions (section 7.2) can be implemented straightforwardly, while some (section 7.1) come with substantial disadvantages and

are perhaps not worth being pursued. Other extensions (section 7.3) go beyond the scope of this paper.

7.1 Dimensions and physical quantities

In section 5 we have introduced the concrete data types D and Q and encoded the notions of dimensions and of physical quantities in the domain of mechanics and for the *LTE* (lengths, times and masses) class of units of measurement. This has allowed us to formalize basic notions of DA in type theory and to apply dependent types to ensure the dimensional consistency of expressions involving physical quantities (section 5) and assist the formulation of relationships between physical quantities that provably satisfy the covariance principle (section 6). In doing so, we have exploited a number of properties that values of type D fulfilled by definition. Most prominently, the fact that equality of dimensions is decidable. In section 5.5 we also suggested that D together with the binary operation *Times* form a group.

The algebraic structure of dimensions. It seems natural to generalize the approach of section 5 by putting forward the algebraic structure of dimensions. As we will see, this has both advantages and disadvantages. Again, we first discuss the generalization in the domain of mechanics and for the *LTE* class of units of measurement.

As done in [9] for the notions of functor and monad, we discuss the operations required for a type to be a dimension as well as their laws through an Idris type class. Encoding the algebraic structure of dimensions through type classes and attempting generic implementations of *df* and of dimensional judgments requires introducing a few language-specific details but is a well-established method for discovering the potential drawbacks of more abstract approaches than the one proposed in section 5. In Idris, type classes are introduced through the **interface** keyword. For example

```
interface DecEq t where
  decEq : (x1 : t) → (x2 : t) → Dec (x1 = x2)
```

explains what it means for a type t to be in *DecEq*, the class of types for which propositional equality is decidable. The data constructor *Dec* in the definition of *DecEq* is defined as

```
data Dec : Type → Type where
  Yes : (prf : prop) → Dec prop
  No  : (contra : prop → Void) → Dec prop
```

A value of type *Dec prop* can only be constructed in two ways: either by providing a proof of *prop* (a value of type *prop*) or by providing a proof of *Not prop* (a function that maps values of type *prop* to values of the empty type, that is, a contradiction). Thus, a value of type *Dec* $(x_1 = x_2)$ is either a proof of $x_1 = x_2$ or a proof of *Not* $(x_1 = x_2)$ which is what it means for the equality to be decidable.

Similarly, we can explain what it means for a type D to encode the notion of dimension through a *Dimension* interface. As discussed in section 5.1, we need dimensional judgments or, more precisely, equality in D, to be decidable. This can be expressed by introducing *Dimension* as a *refinement* of *DecEq*:

```
interface DecEq D ⇒ Dimension D where
```

Perhaps confusingly, this says that *Dimension D* implies *DecEq D* or, in other words, that being in *DecEq* is a necessary condition for being in *Dimension*. This condition is certainly

not sufficient. We have seen in section 5.1 that, as a minimum, we need to be able to define dimensionless physical quantities and the 3 fundamental dimensions of the *LTE* class:

$$\mathit{DimLess} : D; \quad \mathit{Length} : D; \quad \mathit{Time} : D; \quad \mathit{Mass} : D$$

Further, we need the *Times* and *Over* combinators

$$\begin{aligned}&\mathit{Times} : D \to D \to D\\ &\mathit{Over} \ : D \to D \to D\end{aligned}$$

It is time to put forward some axioms. In section 5.5 we mentioned that d '*Over*' d is equal to *DimLess* (for any $d : D$) and that D is a group. The idea is that D together with the *Times* operation is the free Abelian group generated by the fundamental dimensions (which are also required to be not equal). Thus, writing $(\cdot)$ for *Times*, $(/)$ for *Over*, and 1 for *DimLess* we have

$$\begin{aligned}&\mathit{isCommutativeTimes} && : \{d_1, d_2 : D\} \to d_1 \cdot d_2 = d_2 \cdot d_1\\ &\mathit{isAssociativeTimes} && : \{d_1, d_2, d_3 : D\} \to (d_1 \cdot d_2) \cdot d_3 = d_1 \cdot (d_2 \cdot d_3)\\ &\mathit{isLeftIdentityDimLess} && : \{d : D\} \to 1 \cdot d = d\\ &\mathit{isRightIdentityDimLess} && : \{d : D\} \to d \cdot 1 = d\\ &\mathit{isLeftInverseDimLessOver} && : \{d : D\} \to (1/d) \cdot d = 1\\ &\mathit{isRightInverseDimLessOver} && : \{d : D\} \to d \cdot (1/d) = 1\end{aligned}$$

In order to derive $d/d = 1$ one also needs *Times* to associate with *Over* [26]:

$$\mathit{noPrec} : \{d_1, d_2, d_3 : D\} \to (d_1 \cdot d_2)/d_3 = d_1 \cdot (d_2/d_3)$$

With *DimLess*, *Times* and *Over* one can implement the functions *Pow* and *ProdPows* from section 5.5 generically

$$\begin{aligned}&\mathit{Pow} \quad : \{D : \mathit{Type}\} \to \mathit{Dimension}\ D \Rightarrow D \to \mathbb{Z} \to D\\ &\mathit{Pow}\ d\ n = \mathit{pow}\ d\ (\mathit{integerRec}\ n)\ \textbf{where}\\ &\quad \mathit{pow} : \{n : \mathbb{Z}\} \to D \to \mathit{IntegerRec}\ n \to D\\ &\quad \mathit{pow}\ d\ \mathit{IntegerZ} \quad\quad\ = 1\\ &\quad \mathit{pow}\ d\ (\mathit{IntegerSucc}\ m) = \mathit{pow}\ d\ m \cdot d\\ &\quad \mathit{pow}\ d\ (\mathit{IntegerPred}\ m) = \mathit{pow}\ d\ m/d\\ &\mathit{ProdPows} : \{n : \mathbb{N}\} \to \{D : \mathit{Type}\} \to \mathit{Dimension}\ D \Rightarrow \mathit{Vec}\ n\ D \to \mathit{Vec}\ n\ \mathbb{Z} \to D\\ &\mathit{ProdPows}\ \mathit{Nil} \quad\ \mathit{Nil} \quad\ = 1\\ &\mathit{ProdPows}\ (d :: ds)\ (p :: ps) = \mathit{Pow}\ d\ p \cdot \mathit{ProdPows}\ ds\ ps\end{aligned}$$

and define derived dimensions as we did in section 5.1:

$$\begin{aligned}&\mathit{Velocity} : \{D : \mathit{Type}\} \to \mathit{Dimension}\ D \Rightarrow D\\ &\mathit{Velocity} = \mathit{Length}/\mathit{Time}\\ \\ &\mathit{Acceleration} : \{D : \mathit{Type}\} \to \mathit{Dimension}\ D \Rightarrow D\\ &\mathit{Acceleration} = \mathit{Velocity}/\mathit{Time}\\ &\mathit{Force} \quad\quad : \{D : \mathit{Type}\} \to \mathit{Dimension}\ D \Rightarrow D\\ &\mathit{Force} \quad\quad = \mathit{Mass} \cdot \mathit{Acceleration}\\ &\mathit{Energy} \quad\quad : \{D : \mathit{Type}\} \to \mathit{Dimension}\ D \Rightarrow D\end{aligned}$$

$$
\begin{aligned}
&\mathit{Energy} && = \mathit{Mass} \cdot (\mathit{Velocity} \cdot \mathit{Velocity})\\
&\mathit{Work} && : \{D : \mathit{Type}\} \to \mathit{Dimension}\ D \Rightarrow D\\
&\mathit{Work} && = \mathit{Force} \cdot \mathit{Length}
\end{aligned}
$$

Notice, however, that the type of *Velocity*, *Acceleration*, etc. is generic rather than specific. As a consequence, proving elementary dimensional equalities requires some more work, as one would expect. For example, in section 5.1, we could assess the equivalence between energy and mechanical work simply by

$$
\begin{aligned}
&\mathit{check}_1 : \mathit{Energy} = \mathit{Work}\\
&\mathit{check}_1 = \mathit{Refl}
\end{aligned}
$$

because the type of *Energy* and *Work* was fully *defined*. A similar proof based on the *specification Dimension D* would look like

$$
\begin{aligned}
&\mathit{check}_1 : \{D : \mathit{Type}\} \to \mathit{Dimension}\ D \Rightarrow (=)\ \{A = D\}\ \{B = D\}\ \mathit{Energy}\ \mathit{Work}\\
&\mathit{check}_1 = && (\mathit{Energy})\\
&\quad =\{\ \mathit{Refl}\ \}= && (\mathit{Mass} \cdot (\mathit{Velocity} \cdot \mathit{Velocity}))\\
&\quad =\{\ \mathit{Refl}\ \}= && (\mathit{Mass} \cdot ((\mathit{Length}/\mathit{Time}) \cdot (\mathit{Length}/\mathit{Time})))\\
&\quad =\{\ ?\mathbf{h_0}\ \}= && ((\mathit{Mass} \cdot ((\mathit{Length}/\mathit{Time})/\mathit{Time})) \cdot \mathit{Length})\\
&\quad =\{\ \mathit{Refl}\ \}= && ((\mathit{Mass} \cdot (\mathit{Velocity}/\mathit{Time})) \cdot \mathit{Length})\\
&\quad =\{\ \mathit{Refl}\ \}= && ((\mathit{Mass} \cdot \mathit{Acceleration}) \cdot \mathit{Length})\\
&\quad =\{\ \mathit{Refl}\ \}= && (\mathit{Force} \cdot \mathit{Length})\\
&\quad =\{\ \mathit{Refl}\ \}= && (\mathit{Work})\\
&\quad \mathit{QED}
\end{aligned}
$$

We can omit all *Refl* steps, which are only there to guide the reader (not the type checker) and perhaps make the type of check_1 more readable. However, filling in the $?\mathbf{h_0}$ hole and completing the proof requires invoking the axioms of *Dimension*, see the literate Idris code that generates this document [10]. Alas, we know that implementing generic proofs can be awkward!

Thus, a DSL for dimensionally consistent programming that does not rely on a concrete representation of *D* like the one put forward in section 5, would have to provide a library of proofs of elementary equalities like the one between energy and work. Perhaps more importantly, it would also have to provide proofs of elementary inequalities, for example that $\mathit{Not}\ (\mathit{Force} = \mathit{Energy})$. In section 5.1, we could assess this inequality by

$$
\begin{aligned}
&\mathit{check}_2 : \mathit{Not}\ (\mathit{Force} = \mathit{Energy})\\
&\mathit{check}_2\ \mathit{Refl}\ \mathit{impossible}
\end{aligned}
$$

Implementing a generic proof on the only basis that the type of *Force* is equal to the type of *Energy* and that such type is in *Dimension* would not be as easy. As a minimum, it would require extending the *Dimension* interface with axioms that guarantee that the generators are not equal.

Besides providing the basic grammar of the *D*-language, a data type in *Dimension* also needs to provide a dimension function. There are (at least) two ways of encoding this requirement. One is to require *Dimension* to be equipped with a dimension function method

$$
\mathit{df} : D \to (\mathbb{R}_+^3 \to \mathbb{R}_+)
$$

that fulfils the specifications corresponding to *dfLemma1*, *dfLemma2* and *dfLemma3* from section 5.1:

$$dfSpec_1 : (d : D) \to (ls, ls' : \mathbb{R}_+^3) \to df\ d\ ls \cdot df\ d\ ls' = df\ d\ (ls \cdot ls')$$

$$dfSpec_2 : (d : D) \to df\ d\ one3 = 1.0$$

$$dfSpec_3 : \{d : D\} \to \{u, v, w : Units\} \to df\ d\ (fs\ u\ v) \cdot df\ d\ (fs\ v\ w) = df\ d\ (fs\ u\ w)$$

But instantiating *Dimension* with $dfSpec_1$, $dfSpec_2$ and $dfSpec_3$ would have to rely on non-implementable assumptions (if $\mathbb{R}_+$ is just an alias for floating-point numbers) or on a formalization of real numbers. One way to circumvent this difficulty would be to restrict the type of $df\ d$ to $\mathbb{Q}_+^3 \to \mathbb{Q}_+$. This is awkward and conceptually unsatisfactory.

Alternatively, one could require *Dimension* to expose the integer exponents of the dimension function eq. (18):

$$ds : D \to Vec\ 3\ \mathbb{Z}$$

One could then define the dimension function associated with a D type in *Dimension* on the basis of such exponents, as done in section 5.2. For example

```
df : {D : Type} → Dimension D ⇒ D → Vec 3 ℝ₊ → ℝ₊
df d ls = foldr (·) 1.0 (zipWith pow ls rds) where
  rds : Vec 3 ℝ
  rds = map fromInt (ds d)
```

The discussion above suggests that a DSL for DA and dimensionally consistent programming should be based on a concrete implementation of D like the one discussed in section 5.1. We argue that this conclusion holds even if we define *Dimension* as a refinement of a *Group* type class. By a similar token, we argue that a DSL for DA and dimensionally consistent programming should also be based on a concrete implementation of the data type Q for physical quantities, as proposed in section 5.2.

Beyond mechanics and the LTE class. Other parameters in which it is natural to generalize the DSL from section 5 are the number of fundamental dimensions and the class of units of measurement: we have introduced data types for dimensions, physical quantities, etc. in the specific domain of mechanics ($n = 3$ fundamental dimensions) and for the *LTE* (lengths, times and masses) class of units of measurement. But the Pi theorem holds for an arbitrary number of fundamental dimensions and, perhaps more importantly, for arbitrary classes of units. It would be nice to have a general theory that is parameterized on n and on the class of units and that can be easily instantiated to other domains. The major obstacle towards such a generalization is, as already mentioned, the need to formalize a significant fraction of linear algebra. In section 5 we have defined the predicates *IsDep*, *AreDep*, *AreIndep* on D-values (and the corresponding ones for physical quantities) for the specific case $n = 3$. This is straightforward but implementing these predicates for a generic n can only be done on the top of a library that formalizes the basic notions of linear algebra. Building such a library is certainly not trivial, see [22] for a prototype implementation.

At the same time, it is probably worth keeping in mind the potential danger associated with generalizations of DA: while the theory can be a powerful methodology to reduce complexity and, up to a certain extent, obtain physical laws "for free", it is not a "generic" tool. As Bridgman has made very clear in Chapter 5 of his 1931 "Dimensional Analysis"

book, applying DA to a domain whose fundamental laws have not yet been formulated in a form independent of the size of the fundamental units can be potentially very dangerous. At this early stage, we feel that the best that we can do is to build small prototypes of consistent "grammars" of dimensions for specific domains and test how they work.

Thus, our preliminary conclusion is that generalizing the approach of section 5 is useful to understand the algebraic structure of dimension functions and the role of the number of fundamental dimensions in the notions of dimensional (in)dependence that are at the core of the Pi theorem but also comes with a number of practical disadvantages. This is not really surprising if one keeps in mind that we still do not have an established methodology for encoding fragments of well-understood theories (for example, game theory, optimal control, linear algebra) in dependently typed languages.

7.2 Functions and their dimensions

As we have seen in sections 2 to 4, most computations in mathematical physics involve operations on functions between physical quantities. For example, the following function that describes the position of a body moving in a constant gravitational field:

$$pos : Q\ Time \to Q\ Length$$
$$pos\ t = 1\ /\ 2 \triangleleft (g \cdot pow\ t\ 2)$$

Standard arithmetic operations between such functions can be defined straightforwardly by lifting the corresponding operations on Q-values. For example:

$$(+) : \{d_1, d_2 : D\} \to (Q\ d_1 \to Q\ d_2) \to (Q\ d_1 \to Q\ d_2) \to (Q\ d_1 \to Q\ d_2)$$
$$(+)\ f_1\ f_2 = \lambda q \to f_1\ q + f_2\ q$$

Other operations, however, require some more care. Let pos' represent the first derivative of pos. What should be the type of pos'? The discussion at the end of section 3 suggests that this must be $Q\ Time \to Q\ Velocity$. We can build on the DSL of section 5 and specify types for dimensionally consistent differentiation, for example

$$derivative : \{d_0, d_1 : D\} \to (Q\ d_0 \to Q\ d_1) \to Q\ d_0 \to Q\ (d_1\ `Over`\ d_0)$$

This is enough to support elementary dimensional judgments

$$pos'' : typeOf\ (derivative\ (derivative\ pos))$$
$$pos''\ t = g$$
$$check_{13} : dimCodomain\ pos'' = Acceleration$$
$$check_{13} = Refl$$

and reject definitions that are dimensionally inconsistent like $pos''\ t = x\ /\ t$. As one would expect, actually implementing *derivative* (and dimensionally consistent operations for partial differentiation, integration, "nabla" operators etc.) requires developing a small DSL of elementary calculus for functions of Q-variables (for example, as discussed in [33, Chapter 3] for functions of real variables) and thus involves making a number of non-trivial decisions.

7.3 More advanced features: DA driven program derivation and data analysis

Beside supporting program specification and verified programming, dependently typed languages are also powerful tools for type-driven program development [14]. For example, the Idris system can be queried interactively and asked to assist filling in holes like ?$\mathbf{h_0}$. This

suggests that, in principle, one should be able to exploit the two-step construction discussed in section 6.3 to make the type checker fit for assisting the implementation of physical relationships that fulfil the Pi theorem.

For example, coming back to the simple pendulum example from sections 5.2 and 6.3, we may want to implement a function that computes the length $penLen\ \alpha\ g\ m\ \tau$ of a pendulum given the amplitude α of the oscillations, the acceleration of gravity g, its mass m, and its period of oscillations τ:

$$penLen\ :\ Q\ DimLess \to Q\ Acceleration \to Q\ Mass \to Q\ Time \to Q\ Length$$

As a first step, we assess that *Acceleration*, *Mass* and *Time* are independent and that *DimLess* depends on these three dimensions. This can be done straightforwardly:

$$check_{14}\ :\ AreIndep\ [Acceleration, Mass, Time]$$
$$check_{14} = Refl$$

$$check_{15}\ :\ IsDep\ DimLess\ [Acceleration, Mass, Time]$$
$$check_{15} = Evidence\ (1, [0, 0, 0])\ (not1eq0, Refl)$$

Then we define $penLen\ \alpha\ g\ m\ \tau$ as a product of powers of g, m and τ, consistently with the Pi theorem

$$penLen\ \alpha\ g\ m\ \tau = pow\ g\ ?\mathbf{h_1} \cdot pow\ m\ ?\mathbf{h_2} \cdot pow\ \tau\ ?\mathbf{h_3} \cdot Psi\ \alpha$$

and fill in the holes with exponents that match the type of *penLen*: 1, 0 and 2. The function $Psi\ :\ Q\ DimLess \to Q\ DimLess$ remains undefined, but is the only part left to deduce from experiments. The type checker will not accept other implementations of *penLen* but notice that we are solving the system of equations $?\mathbf{h_1} = 1$; $-2 \cdot ?\mathbf{h_1} + ?\mathbf{h_3} = 0$; and $?\mathbf{h_2} = 0$ by hand, with little help from the type checker!

A better approach would be to ask the type checker to solve the system for us, e.g., by searching for suitable values of $?\mathbf{h_1}$, $?\mathbf{h_2}$ and $?\mathbf{h_3}$ in certain ranges. Perhaps more importantly, one would like the type checker to detect situations in which the system has no solutions and recommend possible decompositions of the arguments of physical relationships into lists of dimensionally independent and dimensionally dependent components: the *as* and the *bs* parameters of the Pi theorem. As discussed in sections 5.2 and 6.3, this requires formulating a fragment of linear algebra in type theory and modifications to the Idris type checker. But we think that it is an effort that would be worth pursuing: it would provide domain experts with alternative, dimensionally consistent views of data sets and help practitioners reduce the complexity of data-based studies.

7.4 Related and future work

The best account we have found of the Pi theorem from a linear algebra perspective is by Curtis et al. [21]. In a compact 10-page paper they both explain informally, and prove more formally, the Pi theorem in a classical mathematical style. Unfortunately, there is no connection to programming languages or types, only vector spaces, linear transformations, and how to model dimensional analysis in this setting.

In the key reference on the programming languages side, Kennedy [36] describes one way of combining relational parametricity and units of measure. Our paper shares some of the main ideas: the use of a typed functional programming language, integers as exponents of the base dimensions, and parametric polymorphism. But there are also differences: where their

types are indexed by units, ours are indexed by dimensions; and where they prove results using parametricity, we formulate the Pi theorem using dependent types. An interesting avenue for future work could be to combine these approaches and perhaps use parametricity for dependent types [5] to gain further understanding of the interplay between dimension analysis and strongly typed functional programming.

Atkey et al. [2] provide a categorical framework for semantic models of a type theory that has special types for physical quantities. Their starting point is [36] and they provide a general notion programming language with physical dimension types. They provide dimension polymorphism, but not regular System-F-style polymorphism. The paper provides an impressive collection of definitions, theorems, and examples which explain how Abelian groups, fibrations, and groupoid actions can be used to build models for languages with dimensions. Even though the setting is different (and much more general), their key type *Quantity* (X) is basically the same as our $Q\ d$. They briefly mention an instance of the Pi theorem, but do not formalize it.

The most recent related work is that by McBride and Nordvall-Forsberg [41] which uses dependent types to extend type systems for units of measure from scalars to matrices. Their approach is more algebraic, with graded semirings over a group of physical dimensions, and it would be interesting to further extend this to tensor calculus, a domain where we are also developing DSLs [6].

7.5 Physical laws revisited

We conclude this section by going back to section 3 where we have argued that equations like Newton's second principle, eq. (4), or the ideal gas law, eq. (5), summarize empirical facts about measurements (or put forward axioms about such measurements) of physical quantities. Specifically, we have argued that eq. (4) posits that measurements of F (force) are equal to the product of measurements of m (mass) and measurements of a (acceleration). In section 5.2 we have formalized the notions of physical quantity and measurement and in section 6 we have applied these notions to formulate the covariance principle for a generic function between physical quantities, see fig. 1.

With this understanding, we can now give a clearer meaning to equations eqs. (4) and (5) from section 2 and, more generally, to equations that represent physical laws. The idea is that these equations represent both functions between physical quantities, for example

$$F\ :\ Q\ Mass \to Q\ Acceleration \to Q\ Force$$
$$F\ m\ a = m \cdot a$$

and also instances of the covariance principle as encoded generically in fig. 1

$$\mu_u\ (m \cdot a) = \mu_u\ m \cdot \mu_u\ a$$

or, with $\rho_F\ :\ \mathbb{R} \to \mathbb{R} \to \mathbb{R},\ \ \rho_F = (\cdot)$

$$\mu_u\ (F\ m\ a) = \rho_F\ (\mu_u\ m)\ (\mu_u\ a)$$

In this special form the covariance principle can actually be proved, as discussed in section 5.2.

8 Conclusions

Specialization and the pervasive usage of computer-based modelling and simulation in the physical sciences have widened the gap between the languages of mathematical physics and

modelling and those of mathematics and functional programming. This gap is a major obstacle to fruitful communication and to interdisciplinary collaborations: computer-based modelling critically needs precise specifications and dependently typed programming languages have enough expressive power to support formulating such specifications. But dependently typed programming languages are not (yet) well equipped for encoding the "grammar of dimensions" which rules the languages of mathematical physics and modelling. Our contribution is a first step towards making FP more suitable for developing applications in these domains.

We have studied the role of equations, laws and dimensions on the basis of established examples from the physical sciences and from seminal works in modelling. We have analyzed the notions of dimension function, physical quantity and units of measurement and we have provided an account of the theory of physical similarity and of Buckingham's Pi theorem from the point of view of computer science and FP. Finally, we have proposed a small DSL that encodes these notions in Idris, supports dimensional judgments, and leverages the type system of the host language to provide tests of dimensional consistency, dependence, and independence to ensure the consistency of expressions involving physical quantities.

We have formalized the covariance principle (the requirement that physical laws be independent of the chosen system of units) as a homomorphism between the algebra of physical quantities and the algebra of real numbers (in fig. 1). We have proved that elementary arithmetic operations preserve this property, providing a rigorous type-theoretic foundation for dimensionally consistent programming.

The DSL also supports classical, non-implementable formulations of Buckingham's Pi theorem and we have derived one such formulation. In section 6.3, we have introduced a constructive direction. This allows us to go beyond merely checking existing equations: it enables the definition of functions that fulfil the covariance principle by construction. Consequently, we have shown that dependently typed languages can support DA-driven program derivation, where the type checker assists in identifying valid physical laws from empirical data constraints.

From this perspective, our work is also a contribution towards understanding relativity principles through formalization. In the physical sciences these principles are well understood and appreciated. They have led to important applications in engineering and data science. But it is not clear how relativity principles could be formulated in the economic or biological sciences, and thus also in climate science. We believe that type theory and FP can contribute towards answering this question.

Acknowledgments

We are grateful to Prof. Jeremy Gibbons, Dr. Julian Newman, the JFP editors and the anonymous reviewers whose comments and questions have led to significant improvements of the original manuscript. Guilherme da Silva contributed to the initial discussions and developed a prototype Agda library for linear algebra. The work presented in this paper heavily relies on free software, among others on Idris, Agda, Rocq, GHC, git, vi, Emacs, LaTeX and on the FreeBSD and Debian GNU/Linux operating systems. It is our pleasure to thank all developers of these excellent products. This is TiPES contribution No 231. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 820970.

References

[1] V.I. Arnold. 1989. *Mathematical methods of classical mechanics*. Springer.

[2] Robert Atkey, Neil Ghani, Fredrik Nordvall Forsberg, Timothy Revell, and Sam Staton. 2015. Models for Polymorphism over Physical Dimensions. In *13th International Conference on Typed Lambda Calculi and Applications (TLCA'15) (Leibniz International Proceedings in Informatics (LIPIcs))*, Thorsten Altenkirch (Ed.). 45–59. https://doi.org/10.4230/LIPIcs.TLCA.2015.45

[3] G.I. Barenblatt et al. 1996. *Scaling, Self-similarity, and Intermediate Asymptotics: Dimensional Analysis and Intermediate Asymptotics*. CUP. https://books.google.de/books?id=r-Az53e-MTYC

[4] Baumann et al. 2019. quantities: Type-safe physical computations and unit conversions in Idris. (2019). https://github.com/timjb/quantities

[5] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for free — Parametricity for dependent types. *J. Funct. Program.* 22, 2 (2012), 107–152. https://doi.org/10.1017/S0956796812000056

[6] Jean-Philippe Bernardy and Patrik Jansson. 2025. Domain-specific tensor languages. *Journal of Functional Programming* 35 (2025), e9. https://doi.org/10.1017/S0956796825000048

[7] R. Bird. 2014. *Thinking Functionally with Haskell*. Cambridge University Press. https://books.google.de/books?id=B4RxBAAAQBAJ

[8] Nicola Botta, Nuria Brede, Michel Crucifix, Cezar Ionescu, Patrik Jansson, Zheng Li, Marina Martínez, and Tim Richter. 2023. Responsibility Under Uncertainty: Which Climate Decisions Matter Most? *Environmental Modeling & Assessment* (2023). https://doi.org/10.1007/s10666-022-09867-w

[9] Nicola Botta, Nuria Brede, Patrik Jansson, and Tim Richter. 2021. Extensional equality preservation and verified generic programming. *Journal of Functional Programming* 31 (2021), e24. https://doi.org/10.1017/S0956796821000204

[10] Nicola Botta, Patrik Jansson, and Guilherme da Silva. 2025. Associated code for the paper "Types, equations, dimensions and the Pi theorem". (2025). https://gitlab.pik-potsdam.de/botta/papers/-/tree/master/2023.Types,%20equations,%20dimensions%20and%20the%20Pi%20theorem See the `.lidr` literate Idris files and the Idris and Agda files in the `idris` and `agda` folders of the repository.

[11] Nicola Botta, Patrik Jansson, and Cezar Ionescu. 2017. Contributions to a computational theory of policy advice and avoidability. *Journal of Functional Programming* 27 (2017), 1–52. https://doi.org/10.1017/S0956796817000156

[12] N. Botta, P. Jansson, and C. Ionescu. 2018. The impact of uncertainty on optimal emission policies. *Earth System Dynamics* 9, 2 (2018), 525–542. https://doi.org/10.5194/esd-9-525-2018

[13] N. Botta, A. Mandel, C. Ionescu, M. Hofmann, D. Lincke, S. Schupp, and C. Jaeger. 2011. A functional framework for agent-based models of exchange. *Appl. Math. Comput.* 218, 8 (December 2011), 4025–4040.

[14] Edwin Brady. 2017. *Type-Driven Development With Idris*. Manning. http://www.worldcat.org/isbn/9781617293023

[15] P. W. Bridgman. 1922. *Dimensional Analysis*. Yale University Press.

[16] Edgar Buckingham. 1914. On Physically Similar Systems; Illustrations of the Use of Dimensional Equations. *Phys . Rev.* 4, 4 (1914), 345–376.

[17] Edgar Buckingham. 1915. Model experiments and the form of physical equations. *Transactions of The American Society of Mechanical Engineers* 37 (1915), 263–296.

[18] Bjorn Buckwalter. 2006-2022. dimensional: Statically checked physical dimensions. (2006-2022). https://hackage.haskell.org/package/dimensional

[19] A.J. Chorin and J.E. Marsden. 2000. *A Mathematical Introduction to Fluid Mechanics*. Springer New York. https://books.google.de/books?id=0Iglq1WA5PQC

[20] Richard Courant and David Hilbert. 1989. *Methods of Mathematical Physics*. Vol. 1. Wiley, New York.

[21] W.D. Curtis, J. David Logan, and W. A. Parker. 1982. Dimensional analysis and the pi theorem. *Linear Algebra Appl.* 47 (1982), 117–126. https://doi.org/10.1016/0024-3795(82)90229-4

[22] Guilherme da Silva. 2025. Agda linear algebra. (2025). https://github.com/guilhermehas/agda-linear-algebra

[23] Richard Eisenberg and Takayuki Muranushi. 2022. units: A domain-specific type system for dimensional analysis. https://hackage.haskell.org/package/units. (2022). https://hackage.haskell.org/package/units

[24] Galassi et al. 1996-2023. GSL - GNU Scientific Library. (1996-2023). https://www.gnu.org/software/gsl/

[25] J. C. Gibbings. 2011. *Dimensional analysis*. Springer, London.

[26] Jeremy Gibbons. 1991. *Algebras for Tree Algorithms*. D. Phil. thesis. Programming Research Group, Oxford University. Available as Technical Monograph PRG-94. ISBN 0-902928-72-4.

[27] Mathias Hoppe, Ola Embreus, and Tünde Fülöp. 2021. DREAM: A fluid-kinetic framework for tokamak disruption runaway electron simulations. *Comput. Phys. Commun.* 268 (2021), 108098. https://doi.org/10.1016/j.cpc.2021.108098

[28] R. T. House. 1983. A Proposal for an Extended Form of Type Checking of Expressions. *Comput. J.* 26, 4 (nov 1983), 366–374. https://doi.org/10.1093/comjnl/26.4.366

[29] W. A. Howard. 1969. The formulae-as-types notion of construction. (1969). https://www.cs.cmu.edu/~crary/819-f09/Howard80.pdf

[30] Cezar Ionescu and Patrik Jansson. 2013. Testing versus proving in climate impact research. In *Proc. TYPES 2011 (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 41–54. https://doi.org/10.4230/LIPIcs.TYPES.2011.41

[31] Cezar Ionescu and Patrik Jansson. 2016. Domain-Specific Languages of Mathematics: Presenting Mathematical Analysis Using Functional Programming. *Electronic Proceedings in Theoretical Computer Science* 230 (Nov 2016), 1–15. https://doi.org/10.4204/eptcs.230.1

[32] Cezar Ionescu, Patrik Jansson, and Nicola Botta. 2018. Type Theory as a Framework for Modelling and Programming. In *ISoLA 2018 LNCS Proceedings*. Springer, 119–133. https://doi.org/10.1007/978-3-030-03418-4_8

[33] Patrik Jansson, Cezar Ionescu, and Jean-Philippe Bernardy. 2022. *Domain-Specific Languages of Mathematics*. Texts in Computing, Vol. 24. College Publications. 268 pages.

[34] Dan Jonsson. 2014. Dimensional Analysis: A Centenary Update. (2014). arXiv:math.HO/1411.2798

[35] Keller et al. 2023. Unitful: Physical quantities with arbitrary units. (2023). https://github.com/PainterQubits/Unitful.jl

[36] Andrew J. Kennedy. 1997. Relational Parametricity and Units of Measure. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '97)*. ACM, 442–455. https://doi.org/10.1145/263699.263761

[37] A. Kolmogoroff. 1932. Zur Deutung der intuitionistischen Logik. *Math Z* 35 (1932), 58–65. https://doi.org/10.1007/BF01186549

[38] Yuri A. Kuznetsov. 1998. *Elements of Applied Bifurcation Theory* (2. ed.). Springer, Berlin.

[39] V. Lucarini. 2004. Towards a definition of climate science. *ArXiv Physics e-prints* (Aug 2004). arXiv:physics/0408038

[40] Per Martin-Löf and Giovanni Sambin. 1984. *Intuitionistic type theory*. Vol. 9. Bibliopolis Naples.

[41] Conor McBride and Fredrik Nordvall-Forsberg. 2022. Type systems for programs respecting dimensions. In *Advanced Mathematical and Computational Tools in Metrology and Testing XII*, F Pavese, A B Forbes, N F Zhang, and A G Chunovkina (Eds.). World Scientific, 331–345. https://doi.org/10.1142/9789811242380_0020

[42] Carroll Morgan. 1994. *Programming from specifications, 2nd Edition*. Prentice Hall.

[43] Julian Newman. 2011. A more "complete" version of the Pi-theorem: DRAFT. (2011). https://doi.org/10.48550/ARXIV.1107.4520

[44] Ulf Norell. 2007. *Towards a practical programming language based on dependent type theory*. Ph.D. Dissertation. Chalmers University of Technology. https://doi.org/10.1.1.436.7331

[45] OpenCFD. 2004-2023. OpenFOAM. (2004-2023). https://www.openfoam.com/

[46] Pint Developers. 2012-2023. Pint: makes units easy. (2012-2023). https://pint.readthedocs.io/en/stable/

[47] Istvan Pusztai, Ida Ekmark, Hannes Bergström, Peter Halldestam, Patrik Jansson, Mathias Hoppe, Oskar Vallhagen, and Tünde Fülöp. 2023. Bayesian optimization of massive material injection for disruption mitigation in tokamaks. *Journal of Plasma Physics* 89, 2 (2023), 905890204. https://doi.org/10.1017/S0022377823000193

[48] Wilhelm Quade. 1961. Über die algebraische Struktur des Größenkalküls der Physik. *Abhandlungen der BWG* (1961). https://doi.org/10.24355/dbbs.084-201301141605-0

[49] J. W. S. Rayleigh. 1915. The Principle of Similitude. *Nature* (1915), 66–68. https://doi.org/10.1038/095066c0

[50] Matthias Christian Schabel and Steven Watanabe. 2003-2010. Boost.Units. (2003-2010). https://www.boost.org/doc/libs/1_77_0/doc/html/boost_units.html

[51] Henry Stommel. 1961. Thermohaline Convection with Two Stable Regimes of Flow. *Tellus* 13, 2 (1961), 224–230. https://doi.org/10.3402/tellusa.v13i2.9491

[52] The Astropy Developers. 2011-2023. astropy: Units and Quantities. (2011-2023). https://docs.astropy.org/en/stable/units/

[53] The Numerical Algorithms Group (NAG). 1970-2023. The NAG Library. (1970-2023). https://www.nag.com/content/nag-library

[54] The Python community. 2005-2023. Numerical computing with Python. (2005-2023). https://numpy.org/

[55] The Rocq Development Team. 2025. The Rocq Prover. (Sept. 2025). https://doi.org/10.5281/zenodo.15149628

[56] M. Y. Verbitsky. 2022. Inarticulate past: similarity properties of the ice–climate system and their implications for paleo-record attribution. *Earth System Dynamics* 13, 2 (2022), 879–884. https://doi.org/10.5194/esd-13-879-2022

[57] M. Y. Verbitsky and M. Crucifix. 2020. π-theorem generalization of the ice-age theory. *Earth System Dynamics* 11, 1 (2020), 281–289. https://doi.org/10.5194/esd-11-281-2020

[58] M. Y. Verbitsky and M. Crucifix. 2021. ESD Ideas: The Peclet number is a cornerstone of the orbital and millennial Pleistocene variability. *Earth System Dynamics* 12, 1 (2021), 63–67. https://doi.org/10.5194/esd-12-63-2021

[59] M. Y. Verbitsky and M. Crucifix. 2023. Do phenomenological dynamical paleoclimate models have physical similarity with Nature? Seemingly, not all of them do. *Climate of the Past Discussions* 2023 (2023), 1–15. https://doi.org/10.5194/cp-2023-30

[60] Philip Wadler. 2015. Propositions as types. *Commun. ACM* 58, 12 (2015), 75–84. https://doi.org/10.1145/2699407

[61] Hassler Whitney. 1968a. The Mathematics of Physical Quantities: Part I: Mathematical Models for Measurement. *The American Mathematical Monthly* 75, 2 (1968a), 115–138. http://www.jstor.org/stable/2315883

[62] Hassler Whitney. 1968b. The Mathematics of Physical Quantities: Part II: Quantity Structures and Dimensional Analysis. *The American Mathematical Monthly* 75, 3 (1968b), 227–256. http://www.jstor.org/stable/2314953

[63] Weiqun Zhang et al. 2019. AMReX: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software* 4, 37 (May 2019), 1370. https://doi.org/10.21105/joss.01370