Relational Parametricity and Units of Measure

Andrew J. Kennedy

In *Proceedings of the 24th Annual ACM Symposium on Principles of Programming Languages*, Paris, France, January 1997.

Relational Parametricity and Units of Measure

Andrew J. Kennedy*
LIX, École Polytechnique
91128 Palaiseau cedex, France

Abstract

Type systems for programming languages with numeric types can be extended to support the checking of units of measure. Quantification over units then introduces a new kind of parametric polymorphism with a corresponding Reynolds-style representation independence principle: that the behaviour of programs is invariant under changes to the units used. We prove this 'dimensional invariance' result and describe four consequences. The first is that the type of an expression can be used to derive equations which describe its properties with respect to scaling (akin to Wadler's 'theorems for free' for System F). Secondly there are certain types which are inhabited only by trivial terms. For example, we prove that a fully polymorphic square root function cannot be written using just the usual arithmetic primitives. Thirdly we exhibit interesting isomorphisms between types and for first-order types relate these to the central theorem of classical dimensional analysis. Finally we suggest that for any expression whose behaviour is dimensionally invariant there exists some equivalent expression whose type reflects this behaviour, a consequence of which would be a full abstraction result for a model of the language.

1 Introduction

Ever since FORTRAN, programming languages have provided a numeric type suitable for modelling physical quantities in scientific computations. Typically these quantities possess units of measure (such as kilograms or pounds) which belong to some class, or *dimension* (such as mass), but existing programming languages treat all numeric values as dimensionless. In previous work the author and other researchers have shown how the type system of a language can be extended to support the compile-time checking and automatic inference of dimensions or units of measure [7, 5, 15, 12]. In this article we study the *semantics* of such a language, and in particular, how the notion of *parametricity* (due to Reynolds [11]) has an analogue here.

The paper is structured as follows. In Section 2 we introduce basic concepts from dimensional analysis and explain informally how a programming language can be extended with units of measure, giving illustrative examples in Standard ML. This motivates the formal definition in Section 3 of an explicitly-typed language whose denotational semantics is specified in the standard way using domains and continuous functions. This underlying semantics ignores the unit annotations in types; instead, units are accounted for by a binary relation over the underlying semantics defined in Section 4. Using the relation we prove the main result of this paper: a parametricity theorem which captures the idea that the behaviour of programs is independent of the units of measure used. The theorem is used in Section 5 to prove several properties of terms and types in the language, as outlined in the abstract above. In Section 6 we discuss whether it is possible to use the relation to construct a model of the language which is fully abstract with respect to the underlying cpo-based model. This notion of *relative* full abstraction is believed to be new.

2 Motivation

2.1 Units and dimensions

Physical quantities are measured with reference to a *unit of scale*. When we say that something is '6 metres long' we mean that six metre-lengths placed end-to-end would have the same length. The unit 'metre' is acting as a point of reference, not just for the purpose of *comparison* (X is longer than a metre), but also for *measurement* (X is six times as long as a metre). A single quantity can be measured in many different systems of units, some of which may even be non-linear (such as decibels) or have an origin not at zero (such as degrees Celsius). In

*Current address: Persimmon IT, The Westbrook Centre, Milton Road, Cambridge CB4 1YG, U.K., andrew@persimmon.co.uk

Appears in the Proceedings of the 24th Annual ACM Symposium on Principles of Programming Languages, Paris, France, January 1997. © ACM.

computer science terms, these can be seen as isomorphic data representations; then the notion of *dimension* is a *class* of representations.

It is usual in science to fix a set of *base dimensions* which cannot be defined in terms of each other such as mass, length and time (abbreviated to M, L and T). *Derived* dimensions are products of powers of base dimensions; the dimensions of *force* are MLT^{-2}, for example. Similarly there are base units: the SI units for mass, length and time are respectively kilograms, metres and seconds. Examples of derived units include inches (0.0254 metres) and newtons ($1.0\ \mathrm{kg\ m\ s}^{-2}$). Of course, this division of dimensions and units into base and derived is arbitrary, and one could easily work with, say, force, acceleration and velocity instead of mass, length and time.

Dimensionless quantities are common in science. Examples include refractive index, coefficient of restitution and angle. The last should properly be considered dimensionless though it is tempting to think otherwise – after all, angles are expressed in 'units' of radians. Nevertheless, it is just a dimensionless *ratio* of two lengths: the distance along an arc divided by the radius it subtends.

2.2 Dimensional analysis

The addition, subtraction or comparison of two quantities with different dimensions is invalid dimensionally, whereas their product or quotient has dimensions which are the product or quotient of the corresponding dimensions. If a formula or equation is free of dimension errors, then it is said to be *dimensionally consistent.* For scientists and engineers, dimensional consistency is a handy check on correctness, for dimensional inconsistency certainly indicates that something is amiss. Therefore the automatic checking of numerical programs for dimension errors is potentially very useful.

Why does dimension checking work? The answer lies in the assumption that physical laws are *dimensionally invariant* [9] (or *unit-free* [3]): they remain the same under changes in the units of measure used. Philosophically, this is profound: why should they have the same form at all scales? Pragmatically, it leads to a very useful technique called *dimensional analysis.* The idea is simple: when investigating some physical phenomenon, if the equations governing the phenomenon are not known but the parameters are known, one can use the dimensions of the parameters to narrow down the possible form the equations may take. For example, consider investigating the equation which determines the period of oscillation t of a simple pendulum. Possible parameters are the length of the pendulum l, the mass m, the initial angle from the vertical θ and the acceleration due to gravity g. After performing dimensional analysis it is possible to assert that the equation must be of the form $t = \sqrt{l/g}\,\phi(\theta)$ for some function ϕ of the angle θ. Of course it turns out that for small angles $\phi(\theta) \approx 2\pi$, but dimensional analysis got us a long way – in particular, the period of oscillation turned out to be independent of the mass m. In general, any dimensionally consistent equation over several variables can be reduced to an equation over a smaller number of dimensionless terms which are products of powers of the original variables. This is known (rather awkwardly) as the Pi Theorem (Π = product) [3].

2.3 Some design choices

There are a number of parameters involved in the design of a programming language that supports the prevention of dimension errors. We fix these now.

First, should numeric types be parameterised on units or dimensions? If several units with the same dimension are permitted, then parameterising on units is the sensible choice – it is even possible for the compiler to insert conversions between different units automatically. Semantically, too, it makes sense: the type of an expression which is polymorphic in its units, denoted $\forall u.\tau$, can be interpreted as meaning 'for all changes in the units of measure u'.

The second question is whether or not to allow units such as $\mathrm{kg}^{1/2}$. We take the view that if such a thing arose, it would suggest revision of the set of base units rather than the use of fractional exponents, and therefore we assume that powers of units are integers.

Static checking of units of measure is of limited value without polymorphism, since even something as simple as a squaring function must be polymorphic if it is applied to values with different units. There are many varieties of polymorphism (for example, *ad hoc*, subtype, conjunctive, parametric) but only *parametric* polymorphism suits units of measure, as was recognised even in an early proposal to extend Pascal with units and dimensions [6]. Parametric polymorphism itself comes in many flavours. All of the work on type inference cited in the introduction is based on ML-style polymorphism, in which quantifiers (over types and over units or dimensions) appear only outermost in a type, and polymorphism is introduced solely through the let construct and not in lambda abstractions. On account of these restrictions, the automatic inference of types from typeless expressions is possible – sometimes referred to as *implicit* typing. In contrast, the explicitly-typed polymorphic lambda calculus (or System F) allows quantifiers to be nested inside types; moreover, its type system is *impredicative* in that the quantification in a polymorphic type $\forall t.\tau$ ranges over all types including $\forall t.\tau$ itself. In this article we study polymorphism for units of measure in the style of System F: syntactically it is simpler (there is no need for let or a separation between simple types and type schemes), the type isomorphisms described later turn out to be definable in the system itself, and because units can appear inside types but not *vice versa*, the system is predicative. Also, it is more interesting!

2.4 Examples in Standard ML

Consider writing a function which differentiates another function numerically using the formula

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

The ML code for this accepts a function `f` and increment `h` as arguments and returns a new function which is an approximation to the derivative of `f`:

```
fun diff h f x = (f(x+h) - f(x-h)) / (2.0 * h)
```

Using parametric polymorphism over units of measure this function can be assigned the type

$$\begin{aligned}&\forall u_1.\forall u_2.\mathsf{num}\ u_1 \to (\mathsf{num}\ u_1 \to \mathsf{num}\ u_2)\\ &\quad\to (\mathsf{num}\ u_1 \to \mathsf{num}\ u_2 \cdot u_1^{-1}),\end{aligned}$$

which specifies succinctly the relationship between the units of `f` (which are arbitrary), the increment `h` and the resulting derivative. Our second example is a recursive function for finding a solution to the equation $f(x) = 0$ using Newton's method of iteration:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

The ML code is as follows:

```
fun newton f f' x xacc =
  let val dx = f x / f' x
      val x' = x - dx
  in
    if abs dx / x' < xacc then x'
    else newton f f' x' xacc
  end
```

It accepts a function `f`, its derivative `f'`, an initial estimate `x` and a relative accuracy `xacc`. Its type is

$$\begin{aligned}&\forall u_1.\forall u_2.(\mathsf{num}\ u_1 \to \mathsf{num}\ u_2)\\ &\quad\to (\mathsf{num}\ u_1 \to \mathsf{num}\ u_2 \cdot u_1^{-1})\\ &\quad\to \mathsf{num}\ u_1 \to \mathsf{num}\ \mathbf{1} \to \mathsf{num}\ u_1.\end{aligned}$$

The variable `xacc` and the constant `2.0` are both dimensionless, as indicated by the symbol $\mathbf{1}$. Note, however, that zero should be polymorphic and take any units of measure. If this were not the case, polymorphic types could not be assigned to functions such as `fn x => 0.0 - x` or the function which sums the elements of a list and returns zero for the empty list. For an analogy, consider the polymorphic type `'a list` assigned to `[]` in Standard ML.

2.5 Preview

We now preview informally some results which are given substance in later sections.

For the polymorphic lambda calculus and similar calculi, the essence of *relational parametricity* is the following: if a term e has polymorphic type $\forall t.\tau$, then for any relation between values of types τ_1 and τ_2 the behaviour of $e_{\tau_1} : \{t \mapsto \tau_1\}\tau$ and $e_{\tau_2} : \{t \mapsto \tau_2\}\tau$ is related in a corresponding way. In a sense, e is independent of the *representation* of values of type t. For example, if e has type $\forall t.t \to (t \times t)$, then for any 'change of representation' function $k : \tau_1 \to \tau_2$,

$$e_{\tau_2}(k(x)) \quad\approx\quad (k \times k)(e_{\tau_1}(x))$$

where $k \times k \overset{\text{def}}{=} \lambda y.\langle k(y), k(y)\rangle$ and $\approx$ is an appropriate notion of observational equivalence. Wadler calls such results 'theorems for free' [14] because they are derived purely from the type of a term without inspecting its definition.

Analogously, we will prove that the behaviour of a term e of type $\forall u.\tau$ is independent of the representation assigned to u, that is, the units of measure. Suppose that e has type $\forall u.\mathsf{num}\ u \to \mathsf{num}\ u^2$. It should not matter what units an argument to e has, and if they were scaled by a conversion factor k, then the result should scale by k^2. To be concrete, suppose that an argument whose units are in kilograms is converted into pounds by means of a conversion factor $k = 2.2$ lb kg^{-1}. Then

$$e_{\mathrm{lb}}(k * x) \quad\approx\quad k * k * e_{\mathrm{kg}}(x).$$

The equivalences above cut down the space of possible terms with types $\forall t.t \to (t \times t)$ and $\forall u.\mathsf{num}\, u \to \mathsf{num}\, u^2$. For some types, the space of terms can be cut down even further. For instance, we will prove that functions of type $\forall u.\mathsf{num}\ u^2 \to \mathsf{num}\ u$ cannot return a non-zero result for any argument value. Hence it is not possible to write an approximate square root function with this polymorphic type.

Even more curious is the existence of certain isomorphisms between types – that is, maps from values of one type to the other and *vice versa* which compose to give the identity. For the simply typed lambda calculus there exist isomorphisms such as $\tau_1 \times \tau_2 \cong \tau_2 \times \tau_1$ (exchanging components of a pair) and $\tau_1 \times \tau_2 \to \tau_3 \cong \tau_1 \to \tau_2 \to \tau_3$ (currying and uncurrying) [4]. For a language with polymorphism over units of measure, the following more interesting isomorphism holds (with certain restrictions):

$$\forall u.\mathsf{num}\ u \to \mathsf{num}\ u \to \mathsf{num}\ u \;\cong\; \mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathbf{1}.$$

We show later that isomorphisms like this are intimately related to the Pi Theorem from dimensional analysis described in Section 2.2, and for first-order types prove its programming language equivalent.

3 An explicitly-typed language

We are now ready to formalise the syntax, type system, and denotational semantics of a programming language that supports units of measure.

3.1 Units and types

The syntax for units (ranged over by μ) and for types (ranged over by τ) is specified as follows:

$$\begin{array}{rcll} \mu & ::= & u & \textit{unit variables} \\ & \mid & \mathbf{1} & \textit{`no units'} \\ & \mid & \mu_1 \cdot \mu_2 & \textit{product} \\ & \mid & \mu^{-1} & \textit{inverse} \end{array}$$

$$\begin{array}{rcll} \tau & ::= & \mathsf{bool} & \textit{booleans} \\ & \mid & \mathsf{num}\,\mu & \textit{numbers with units of measure} \\ & \mid & \tau_1 \to \tau_2 & \textit{functions} \\ & \mid & \forall u.\tau & \textit{polymorphic types} \end{array}$$

Unit variables (ranged over by u) are used both to stand for base units (such as kilograms, metres and seconds) and to express polymorphism through explicit quantification. We will see later that the distinction is really that of *free* and *bound* occurrences. Variables are combined using product and inverse, and the symbol $\mathbf{1}$ is used for dimensionless quantities. Types include the booleans, a numeric type parameterised by units, function types and unit-polymorphic types which are identified up to renaming of bound variables. In addition to the usual binding conventions on arrows and quantifiers, we let inverse bind tighter than product which in turn binds tighter than num.

Given a set of unit variables $\mathcal{V}$, the set of all unit expressions with variables in $\mathcal{V}$ is denoted $\text{Units}(\mathcal{V})$, and likewise $\text{Ty}(\mathcal{V})$ for types whose *free* variables are drawn from $\mathcal{V}$. A substitution $S \in \text{Subst}(\mathcal{V}, \mathcal{V}')$ is a map from $\mathcal{V}$ to $\text{Units}(\mathcal{V}')$ which extends homomorphically to maps from $\text{Units}(\mathcal{V})$ to $\text{Units}(\mathcal{V}')$ and from $\text{Ty}(\mathcal{V})$ to $\text{Ty}(\mathcal{V}')$ in the usual way, avoiding variable capture in polymorphic types by renaming bound variables where necessary. We write $S\mu$ and $S\tau$ for the application of a substitution S to unit expression μ and type τ. The particular substitution which maps u to μ is written $\{u \mapsto \mu\}$, and $i_{\mathcal{V},\mathcal{V}'} \in \text{Subst}(\mathcal{V}, \mathcal{V}')$ is the canonical inclusion for $\mathcal{V} \subseteq \mathcal{V}'$.

The most unusual aspect of the type system is that the algebraic properties of units of measure are built into the typing rules via an equational theory of types. Let E be the set of axioms which define an Abelian group of units of measure:

$$\begin{array}{rcll} u_1 \cdot u_2 & = & u_2 \cdot u_1 & \textit{commutativity} \\ (u_1 \cdot u_2) \cdot u_3 & = & u_1 \cdot (u_2 \cdot u_3) & \textit{associativity} \\ u \cdot \mathbf{1} & = & u & \textit{identity} \\ u \cdot u^{-1} & = & \mathbf{1} & \textit{inverses} \end{array}$$

These equations are extended to congruence relations $=_E$ over unit and type expressions by closure under reflexivity, symmetry, transitivity, substitution of unit expressions for unit variables, and congruence.

Let μ^n denote the raising of μ to the power $n \in \mathbb{Z}$:

$$\mu^n \overset{\text{def}}{=} \begin{cases} \mu \cdot \cdots \cdot \mu & (n\text{-fold product) if } n > 0, \\ \mathbf{1} & \text{if } n = 0, \\ \mu^{-1} \cdot \cdots \cdot \mu^{-1} & (-n\text{-fold product) if } n < 0. \end{cases}$$

With this notation, it is easily seen that any unit expression μ can be written in the form

$$\mu =_E u_1^{z_1} \cdot \cdots \cdot u_n^{z_n}$$

where $u_1, \ldots, u_n$ are distinct and $z_1, \ldots, z_n$ are non-zero. We call this the *normal form* of μ.

3.2 Terms

The language which we study is a typed lambda calculus Λ_u with constructs for explicit introduction and elimination of units of measure quantification. In the absence of type polymorphism, fixed-point and conditional constructs are built-in, with unit-polymorphic arithmetic operations and numeric constants provided by a standard environment. The syntax of terms is as follows:

$$\begin{array}{rcll} e & ::= & x & \textit{identifiers} \\ & \mid & \lambda x\colon \tau.\, e & \textit{function abstraction} \\ & \mid & e_1\, e_2 & \textit{function application} \\ & \mid & \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 & \textit{conditional} \\ & \mid & \mathsf{rec}\ x\colon \tau.\, e & \textit{recursive definition} \\ & \mid & \Lambda u.e & \textit{units abstraction} \\ & \mid & e_\mu & \textit{units application} \end{array}$$

The typing rules are shown in Figure 1. As usual, Γ ranges over *type environments*: finite maps from identifiers to types. The set of all type environments with free variables in $\mathcal{V}$ is denoted $\text{TyEnv}(\mathcal{V})$, and the equivalence relation $=_E$ and definition of substitution extend pointwise to type environments. Then a well-formed typing judgment $\mathcal{V}; \Gamma \vdash e : \tau$ means

> "Under type environment $\Gamma \in \text{TyEnv}(\mathcal{V})$ the expression e has type $\tau \in \text{Ty}(\mathcal{V})$".

It is assumed that complete programs in Λ_u are typed in the context of a type environment $\Gamma_{\text{ops}} \cup \Gamma_{\text{units}}$, where Γ_{ops} supplies the types of basic arithmetic operations

$$\text{(id)}\ \frac{}{\mathcal{V};\Gamma\cup\{x:\tau\}\vdash x:\tau}\qquad \text{(abs)}\ \frac{\mathcal{V};\Gamma\cup\{x:\tau_1\}\vdash e:\tau_2}{\mathcal{V};\Gamma\vdash(\lambda x\colon\tau_1.\,e):\tau_1\to\tau_2}\qquad \text{(app)}\ \frac{\mathcal{V};\Gamma\vdash e_1:\tau_1\to\tau_2\quad \mathcal{V};\Gamma\vdash e_2:\tau_1}{\mathcal{V};\Gamma\vdash e_1\,e_2:\tau_2}$$

$$\text{(cond)}\ \frac{\mathcal{V};\Gamma\vdash e_1:\mathsf{bool}\quad \mathcal{V};\Gamma\vdash e_2:\tau\quad \mathcal{V};\Gamma\vdash e_3:\tau}{\mathcal{V};\Gamma\vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3:\tau}\qquad \text{(rec)}\ \frac{\mathcal{V};\Gamma\cup\{x:\tau\}\vdash e:\tau}{\mathcal{V};\Gamma\vdash(\mathsf{rec}\ x\colon\tau.\,e):\tau}$$

$$\text{(eq)}\ \frac{\mathcal{V};\Gamma\vdash e:\tau_1}{\mathcal{V};\Gamma\vdash e:\tau_2}\ \tau_1=_E\tau_2\qquad (\forall\text{-intro})\ \frac{\mathcal{V}\cup\{u\};\Gamma\vdash e:\tau}{\mathcal{V};\Gamma\vdash\Lambda u.e:\forall u.\tau}\ u\notin\mathcal{V}\qquad (\forall\text{-elim})\ \frac{\mathcal{V};\Gamma\vdash e:\forall u.\tau}{\mathcal{V};\Gamma\vdash e_\mu:\{u\mapsto\mu\}\tau}$$

Figure 1: Typing rules: $\mathcal{V};\Gamma\vdash e:\tau$ well-formed if $\Gamma\in\mathrm{TyEnv}(\mathcal{V})$ and $\tau\in\mathrm{Ty}(\mathcal{V})$

$$\begin{aligned}
\Gamma_{\mathrm{ops}} = \\
\{\ \mathtt{0}\ &: \forall u.\mathsf{num}\ u,\\
\mathtt{1}\ &: \mathsf{num}\ \mathbf{1},\\
\mathtt{+}\ &: \forall u.\mathsf{num}\ u\to\mathsf{num}\ u\to\mathsf{num}\ u,\\
\mathtt{-}\ &: \forall u.\mathsf{num}\ u\to\mathsf{num}\ u\to\mathsf{num}\ u,\\
\mathtt{*}\ &: \forall u_1.\forall u_2.\mathsf{num}\ u_1\to\mathsf{num}\ u_2\to\mathsf{num}\ u_1\cdot u_2,\\
\mathtt{/}\ &: \forall u_1.\forall u_2.\mathsf{num}\ u_1\to\mathsf{num}\ u_2\to\mathsf{num}\ u_1\cdot u_2^{-1},\\
\mathtt{<}\ &: \forall u.\mathsf{num}\ u\to\mathsf{num}\ u\to\mathsf{bool}\ \}
\end{aligned}$$

$$\Gamma_{\mathrm{units}} = \{\ \mathtt{kg}:\mathsf{num}\ kg,\ \mathtt{m}:\mathsf{num}\ m,\ \mathtt{s}:\mathsf{num}\ s\ \}$$

Figure 2: Type environments Γ_{ops} and Γ_{units}

and Γ_{units} supplies types for base units as shown in Figure 2. The availability of a comparison test allows the writing of iterative functions such as `newton` from Section 2.4, and also trigonometric functions such as `sin` and `cos` of type $\mathsf{num}\ \mathbf{1}\to\mathsf{num}\ \mathbf{1}$. For simplicity we provide only the constant `1` of type $\mathsf{num}\ \mathbf{1}$; all other dimensionless rationals can be constructed from this using the arithmetic operations provided. When presenting examples, however, we will freely use constants $r\in\mathbb{Q}$ and also constructs such as $e_1 * e_2$ instead of the clumsy $\mathtt{*}_{\mu_1\ \mu_2}(e_1)(e_2)$ (assuming that e_1 and e_2 have types $\mathsf{num}\ \mu_1$ and $\mathsf{num}\ \mu_2$).

3.3 Denotational semantics

We define a call-by-name[†] denotational semantics for Λ_u in the usual way using complete partial orders (or *domains*) and continuous functions (a good reference is [16]). Our notation is standard: if D and E are domains, then $D_\perp$ is the set $\{\perp\}\cup\{\,[d]\mid d\in D\,\}$ in which $\perp$ is the new least element and $[\cdot]$ is the canonical map from D to $D_\perp$, and $D\to E$ is the set of continuous functions between D and E ordered pointwise. Then for each type τ a domain of values $[\![\tau]\!]$ is defined as follows:

$$\begin{aligned}
[\![\mathsf{bool}]\!] &= \mathbb{B}_\perp\\
[\![\mathsf{num}\ \mu]\!] &= \mathbb{Q}_\perp\\
[\![\tau_1\to\tau_2]\!] &= [\![\tau_1]\!]\to[\![\tau_2]\!]\\
[\![\forall u.\tau]\!] &= [\![\tau]\!]
\end{aligned}$$

Here $\mathbb{B}$ is the discrete cpo consisting simply of the values true and false, and $\mathbb{Q}$ is the discrete cpo of rationals. Notice how the units of measure have been ignored – the semantics of units will be captured by a logical relation defined in the next section.

A *value environment* ρ which respects some type environment Γ is a finite map from identifiers in $\mathrm{dom}(\Gamma)$ to values so that whenever $\Gamma(x)=\tau$ then $\rho(x)\in[\![\tau]\!]$. We write $[\![\Gamma]\!]$ for the set of all value environments respecting Γ. If $\mathcal{V};\Gamma\vdash e:\tau$ then the meaning of e is given by a map $[\![e]\!]:[\![\Gamma]\!]\to[\![\tau]\!]$ defined inductively in Figure 3. Again the units are ignored – this *underlying* semantics just resembles that of PCF [10] with rationals in place of integers.

4 Dimensional invariance

To formalise the ideas sketched in Section 2.5, we will define type-indexed relations between domains, parameterised on *scaling environments* which represent changes to the units of measure. For the base type $\mathsf{num}\ \mu$ the scaling will depend directly on the units μ, so that, for instance, if $r\in\mathbb{Q}$ scales at type $\mathsf{num}\ u$ to give kr for some $k\in\mathbb{Q}^+$, then r' should scale at type $\mathsf{num}\ u^2$ to give k^2r'. Thus the obvious approach (and the one taken in [8]) is to assign a scale factor $k\in\mathbb{Q}^+$ to each unit

[†]Chosen for ease of presentation; similar results should hold for a call-by-value language.

$$[\![e]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$$

$$\begin{aligned}
[\![x]\!](\rho) &= \rho(x) \\
[\![\lambda x\colon \tau.\, e]\!](\rho)(v) &= [\![e]\!](\rho[x \mapsto v]) \\
[\![e_1\, e_2]\!](\rho) &= [\![e_1]\!](\rho)([\![e_2]\!](\rho)) \\
[\![\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3]\!](\rho) &= \begin{cases} \bot & \text{if } [\![e_1]\!](\rho) = \bot \\ [\![e_2]\!](\rho) & \text{if } [\![e_1]\!](\rho) = [\mathsf{true}] \\ [\![e_3]\!](\rho) & \text{if } [\![e_1]\!](\rho) = [\mathsf{false}] \end{cases} \\
[\![\mathsf{rec}\ x\colon \tau.\, e]\!](\rho) &= \bigsqcup_{i \in \mathbb{N}} v_i \quad \begin{array}{l} \text{where } v_0 = \bot \text{ and} \\ v_{i+1} = [\![e]\!](\rho[x \mapsto v_i]) \end{array} \\
[\![\Lambda u.e]\!](\rho) &= [\![e]\!](\rho) \\
[\![e_\mu]\!](\rho) &= [\![e]\!](\rho)
\end{aligned}$$

Figure 3: Underlying semantics for $\mathcal{V}; \Gamma \vdash e : \tau$

variable, and then extend this uniquely to a homomorphism from the Abelian group of (equivalence classes of) unit expressions into the Abelian group of scale factors. However this is not sufficient for proving inhabitation results such as the square root example mentioned earlier. So, instead, to avoid anticipating any particular notion of scaling we split dimensional invariance into two parts. First, we prove a parametricity result for Λ_u which holds for a very general definition of scaling environments. Then we show that the standard interpretation of constants in Γ_{ops} imposes a notion of scaling on the scaling environments that subsumes the homomorphisms described above. These two results together constitute *dimensional invariance*.

We start by defining some standard constructions on relations. If $R \subseteq D \times E$ is a relation between domains, then $R_\bot \subseteq D_\bot \times E_\bot$ is the relation $\{(\bot, \bot)\} \cup \{\, ([d], [e]) \mid (d, e) \in R \,\}$. For a domain D, the identity relation $id_D \subseteq D \times D$ is just $\{\, (d, d) \mid d \in D \,\}$. Finally, if $R \subseteq D \times E$ and $S \subseteq D' \times E'$ then $(R \to S) \subseteq (D \to D') \times (E \to E')$ is the relation $\{\, (f, g) \mid (d, e) \in R \Rightarrow (f(d), g(e)) \in S \,\}$.

Let a *scaling environment* ψ be a map from unit expressions to binary relations on $\mathbb{Q}$ which respects unit equivalence (if $\mu_1 =_E \mu_2$ then $\psi(\mu_1) = \psi(\mu_2)$) and which preserves the identity element (that is, $\psi(\mathbf{1}) = id_{\mathbb{Q}}$).[†] A family $\mathcal{E}$ of sets of scaling environments provides for each set of variables $\mathcal{V}$ a set $\mathcal{E}(\mathcal{V})$ consisting of scaling environments over Units($\mathcal{V}$), with the following closure

[†]Strictly speaking this latter property is not required by the parametricity theorem but its inclusion simplifies the presentation, and clearly will be required later to scale dimensionless constants to themselves.

$$\begin{aligned}
R^\psi_\tau &\subseteq [\![\tau]\!] \times [\![\tau]\!] \text{ for } \tau \in \mathrm{Ty}(\mathcal{V}), \psi \in \mathcal{E}(\mathcal{V}) \\
\\
R^\psi_{\mathsf{bool}} &= (id_{\mathbb{B}})_\bot \\
R^\psi_{\mathsf{num}\ \mu} &= \psi(\mu)_\bot \\
R^\psi_{\tau_1 \to \tau_2} &= R^\psi_{\tau_1} \to R^\psi_{\tau_2} \\
R^\psi_{\forall u.\tau} &= \bigcap \left\{ R^{\chi(\psi)}_\tau \mid \chi \in \mathrm{Ext}_{\mathcal{E}}(\mathcal{V}, \mathcal{V} \cup \{u\}) \right\}
\end{aligned}$$

Figure 4: Family of relations induced by $\mathcal{E}$

property: that if $\psi \in \mathcal{E}(\mathcal{V}')$ and $S \in \mathrm{Subst}(\mathcal{V}, \mathcal{V}')$ then $\psi \circ S \in \mathcal{E}(\mathcal{V})$. Given $\mathcal{V} \subseteq \mathcal{V}'$, an *extension* of scaling environments is a map $\chi : \mathcal{E}(\mathcal{V}) \to \mathcal{E}(\mathcal{V}')$ with the property that $\chi(\psi) \circ i_{\mathcal{V},\mathcal{V}'} = \psi$ for all $\psi \in \mathcal{E}(\mathcal{V})$. We write $\mathrm{Ext}_{\mathcal{E}}(\mathcal{V}, \mathcal{V}')$ for the set of extensions from $\mathcal{E}(\mathcal{V})$ to $\mathcal{E}(\mathcal{V}')$.

Using these constructions, a family $\mathcal{E}$ of scaling environments induces a family $\{R^\psi_\tau \subseteq [\![\tau]\!] \times [\![\tau]\!]\}$ of *scaling relations* parameterised on a type $\tau \in \mathrm{Ty}(\mathcal{V})$ and a scaling environment $\psi \in \mathcal{E}(\mathcal{V})$. This is shown in Figure 4. For booleans it is the identity relation, for functions it carries related arguments to related results (this makes R^ψ_τ a *logical* relation), and for polymorphic types it requires that values be related at all possible extensions of the scaling environment on the quantified unit variable. The relations on values extend pointwise to relations on environments $\{R^\psi_\Gamma \subseteq [\![\Gamma]\!] \times [\![\Gamma]\!]\}$ for $\Gamma \in \mathrm{TyEnv}(\mathcal{V})$ and $\psi \in \mathcal{E}(\mathcal{V})$. We omit the scaling environment ψ when τ or Γ is closed (and $\mathcal{V} = \emptyset$), writing R_τ and R_Γ respectively (as there is only one choice of scaling environment, namely that which maps $\mathbf{1}$ to the identity relation).

In order for relations to preserve the fixed-point interpretation of recursive definitions, they must be *strict* and *complete*, that is, preserve $\bot$ and least upper bounds of chains.

Lemma 1. *Let $\mathcal{E}$ induce a family of relations $\{R^\psi_\tau\}$. For any $\tau \in \mathrm{Ty}(\mathcal{V})$ and $\psi \in \mathcal{E}(\mathcal{V})$, the following hold:*

1. $(\bot, \bot) \in R^\psi_\tau$.
2. *If $\{v_i\}$ and $\{v'_i\}$ are chains for which $(v_i, v'_i) \in R^\psi_\tau$ for all $i \in \mathbb{N}$, then $(\bigsqcup\{v_i\}, \bigsqcup\{v'_i\}) \in R^\psi_\tau$.*

Proof. By induction on the structure of τ. □

The interaction of substitutions with scaling relations is captured by the following lemma.

Lemma 2. *Let $\mathcal{E}$ induce a family of relations $\{R^\psi_\tau\}$. Then for any $\tau \in \mathrm{Ty}(\mathcal{V})$, $S \in \mathrm{Subst}(\mathcal{V}, \mathcal{V}')$ and $\psi \in \mathcal{E}(\mathcal{V}')$ it is the case that $R^{\psi \circ S}_\tau = R^\psi_{S\tau}$.*

Proof. By induction on the structure of τ. □

We are now ready to prove the parametricity result. In essence this states that if two value environments are related with respect to some scaling environment, then the interpretations of an expression under those environments are related in a corresponding way. Formally, we write $\mathcal{V};\Gamma \models_{\mathcal{E}} e : \tau$ if for any scaling environment $\psi \in \mathcal{E}(\mathcal{V})$ and pair of related value environments $(\rho, \rho') \in R^{\psi}_{\Gamma}$ it follows that $([\![e]\!](\rho), [\![e]\!](\rho')) \in R^{\psi}_{\tau}$.

Theorem 1 (Parametricity).

$$\mathcal{V};\Gamma \vdash e : \tau \;\Rightarrow\; \mathcal{V};\Gamma \models_{\mathcal{E}} e : \tau.$$

Proof. By induction on the typing derivation. The cases for (abs) and (app) follow as usual for logical relations, the identity relation on booleans is required for (cond), the case for (eq) uses the fact that ψ respects $=_E$, and Lemma 1 is used in (rec). We present the remaining cases in detail.

($\forall$-intro). We need to show

$$\mathcal{V} \cup \{u\};\Gamma \models_{\mathcal{E}} e : \tau \quad\Rightarrow\quad \mathcal{V};\Gamma \models_{\mathcal{E}} \Lambda u.e : \forall u.\tau.$$

Suppose that $\psi \in \mathcal{E}(\mathcal{V})$ and $(\rho, \rho') \in R^{\psi}_{\Gamma}$. Now pick any $\chi \in \mathrm{Ext}_{\mathcal{E}}(\mathcal{V}, \mathcal{V} \cup \{u\})$. Then $(\rho, \rho') \in R^{\chi(\psi)}_{\Gamma}$ because $u \notin \mathcal{V}$, and for the antecedent of the above implication to hold it must be the case that $([\![e]\!](\rho), [\![e]\!](\rho')) \in R^{\chi(\psi)}_{\tau}$. From the semantics $[\![\Lambda u.e]\!] = [\![e]\!]$ so given the definition of the relation at polymorphic type we have just shown that $([\![\Lambda u.e]\!](\rho), [\![\Lambda u.e]\!](\rho')) \in R^{\psi}_{\forall u.\tau}$.

($\forall$-elim). We need to show

$$\mathcal{V};\Gamma \models_{\mathcal{E}} e : \forall u.\tau \quad\Rightarrow\quad \mathcal{V};\Gamma \models_{\mathcal{E}} e_{\mu} : \{u \mapsto \mu\}\tau.$$

Suppose that $\psi \in \mathcal{E}(\mathcal{V})$ and $(\rho, \rho') \in R^{\psi}_{\Gamma}$. Let $S = \{u \mapsto \mu\} \in \mathrm{Subst}(\mathcal{V} \cup \{u\}, \mathcal{V})$. It is easy to see that the map $\psi \mapsto \psi \circ S$ is an element of $\mathrm{Ext}_{\mathcal{E}}(\mathcal{V}, \mathcal{V} \cup \{u\})$ so when the antecedent of the above implication holds it is the case that $([\![e]\!](\rho), [\![e]\!](\rho')) \in R^{\psi\circ S}_{\tau}$. By Lemma 2 we know that $R^{\psi\circ S}_{\tau} = R^{\psi}_{S\tau}$, and because $[\![e_{\mu}]\!] = [\![e]\!]$ the required result is reached. $\square$

In the sections which follow we will apply Theorem 1 to terms which have been typed under the type environment Γ_{ops} and interpreted in the presence of a value environment $\rho_{\mathrm{ops}} \in [\![\Gamma_{\mathrm{ops}}]\!]$. We assume that ρ_{ops} assigns to `0`, `1`, `+`, `-`, `*`, `/` and `<` their usual arithmetic meanings (with division by zero leading to divergence). To get the most out of Theorem 1 we now find the *largest* collection of scaling environments $\mathcal{E}$ such that $(\rho_{\mathrm{ops}}, \rho_{\mathrm{ops}}) \in R_{\Gamma_{\mathrm{ops}}}$ for the relation $R_{\Gamma_{\mathrm{ops}}}$ induced by $\mathcal{E}$.

First observe that the polymorphism of zero forces $(0, 0)$ to be in every relation $\psi(\mu) \in \mathcal{E}(\mathcal{V})$ for any $\mathcal{V}$ and $\mu \in \mathrm{Units}(\mathcal{V})$. It also turns out that if any pair *not* of the form $(0, 0)$ is present in a relation $\psi(\mu)$, then the relation

$$\mathcal{E}_{\mathrm{ops}}(\mathcal{V}) = \left\{ \;\psi_{G,h} \;\middle|\; \begin{array}{l} G \text{ is a subgroup of Units}(\mathcal{V}), \\ h \in \mathrm{hom}(G, \mathbb{Q}^{+}) \end{array} \right\}$$

where

$$\psi_{G,h}(\mu) \;\stackrel{\mathrm{def}}{=}\; \begin{cases} \{\,(r, h(\mu)r) \mid r \in \mathbb{Q}\,\} & \text{if } \mu \in G, \\ \{\,(0,0)\,\} & \text{otherwise.} \end{cases}$$

Figure 5: Sets of scaling environments preserving ρ_{ops}

is a bijection of the form $\{\,(r, kr) \mid r \in \mathbb{Q}\,\}$ for some $k \in \mathbb{Q}^{+}$ (for details of the proof, see Appendix A). This still leaves the possibility of singleton relations $\{(0, 0)\}$, and indeed these do occur. The end result is the family $\mathcal{E}_{\mathrm{ops}}$ shown in Figure 5. In this definition, G is a subgroup of the Abelian group $\mathrm{Units}(\mathcal{V})$, that is, it is a subset that is closed under all unit-forming constructions.[†] Then $h \in \mathrm{hom}(G, \mathbb{Q}^{+})$ is a homomorphism from G into the Abelian group of positive rationals with product as the group operation (that is, $h(\mu_1 \cdot \mu_2) = h(\mu_1)h(\mu_2)$, $h(\mu^{-1}) = 1/h(\mu)$ and $h(\mathbf{1}) = 1$). For example, consider the set of unit expressions generated from $u_1^2 u_2^3$ and u_2^2. Homomorphisms from this into $\mathbb{Q}^{+}$ take the form $u_1^{2m} \cdot u_2^{3m+2n} \mapsto k_1^m k_2^n$ for $k_1, k_2 \in \mathbb{Q}^{+}$.

The essence of the theorem which follows is that $\mathcal{E}_{\mathrm{ops}}$ consists of exactly those scaling environments which preserve all the constants in ρ_{ops}. By a slight abuse of notation we write $\mathcal{E} \subseteq \mathcal{E}'$ if $\mathcal{E}(\mathcal{V}) \subseteq \mathcal{E}'(\mathcal{V})$ for all $\mathcal{V}$.

Theorem 2 (Completeness of $\mathcal{E}_{\mathrm{ops}}$). *Suppose that $\mathcal{E}$ induces $R_{\Gamma_{\mathrm{ops}}}$. Then*

$$(\rho_{\mathrm{ops}}, \rho_{\mathrm{ops}}) \in R_{\Gamma_{\mathrm{ops}}} \iff \mathcal{E} \subseteq \mathcal{E}_{\mathrm{ops}}.$$

Proof. See Appendix A. $\square$

Sieber proves a similar result in his application of logical relations to PCF to remove elements that destroy full abstraction [13]. He shows that certain *sequential* relations are exactly the logical relations under which all the constants of PCF are invariant.

Before presenting sophisticated applications of Theorems 1 and 2 we observe the important special case that the behaviour of a program is independent of the values assigned to the base units. Suppose that a boolean-valued expression e is typed under the type environments of Figure 2:

$$\{kg, m, s\};\Gamma_{\mathrm{ops}} \cup \Gamma_{\mathrm{units}} \vdash e : \mathsf{bool}.$$

[†]To be really precise, we should write $G \subseteq \mathrm{Units}(\mathcal{V})/{=_E}$ and distinguish a unit expression $\mu \in \mathrm{Units}(\mathcal{V})$ from its equivalence class $[\mu]_{=_E} \in \mathrm{Units}(\mathcal{V})/{=_E}$. For simplicity we avoid this.

Given the environment $\rho_{\text{ops}} \in [\![\Gamma_{\text{ops}}]\!]$ described above, it is the case that $[\![e]\!](\rho_{\text{ops}} \cup \rho_{\text{units}})$ has the same value for any $\rho_{\text{units}} \in [\![\Gamma_{\text{units}}]\!]$, with one proviso: the values chosen for ρ_{units} must be positive. This is due to the possibility that e could test their sign using the comparison function <. Intuitively, it makes no sense for units of measure to be negative or zero.

In the rest of this paper, we assume that scaling relations R^{ψ}_{τ} are induced by $\mathcal{E}_{\text{ops}}$ and that $\mathcal{V}; \Gamma \models e : \tau$ is shorthand for $\mathcal{V}; \Gamma \models_{\mathcal{E}_{\text{ops}}} e : \tau$.

5 Dimensional invariance applied

5.1 Theorems for free

Theorems 1 and 2 can be used to derive equivalences of the kind that Wadler calls 'theorems for free' [14]. We give examples informally; the formal reasoning involved is tedious but trivial, and always involves picking scaling environments of the form

$$\psi(u_1^{z_1} \cdots u_n^{z_n}) = \{\ (r, k_1^{z_1} \cdots k_n^{z_n} r) \ \mid \ r \in \mathbb{Q}\ \}$$

for some scale factors $k_1, \ldots, k_n \in \mathbb{Q}^+$. For clarity we omit unit information from expressions, as would be the case for an implicitly-typed language such as ML.

Example (Powers). Consider an expression e with the following type for some $n \in \mathbb{Z}$:

$$\Gamma_{\text{ops}} \vdash e : \forall u.\mathsf{num}\ u \to \mathsf{num}\ u^n.$$

Then for any $k \in \mathbb{Q}^+$ the following equivalence holds:

$$e(k * x) \ \approx \ k^n * e(x).$$

□

Example (Differentiation). Suppose that e has the type of the differentiation function from Section 2.4:

$$\begin{aligned}\Gamma_{\text{ops}} \vdash e : \forall u_1.\forall u_2.&\mathsf{real}\ u_1 \to (\mathsf{num}\ u_1 \to \mathsf{num}\ u_2) \\ &\to (\mathsf{num}\ u_1 \to \mathsf{num}\ u_2 \cdot u_1^{-1}).\end{aligned}$$

Then for any $k_1, k_2 \in \mathbb{Q}^+$ the following equivalence holds:

$$e\ h\ f\ x \ \approx \ \frac{k_2}{k_1} * e\left(\frac{h}{k_1}\right)\left(\lambda x.\frac{f(x * k_1)}{k_2}\right)\left(\frac{x}{k_1}\right).$$

□

5.2 Type inhabitation

Conventional parametricity can be used to characterise all terms with a particular type. For example, there is *no* term in System F with type $\forall t.t$, and for a fragment of ML the type $\forall t.t \to t$ contains only the identity function and the always-divergent function [2]. Analogous results can be obtained for Λ_u.

Example (Square root). Consider the typing

$$\Gamma_{\text{ops}} \vdash e : \forall u.\mathsf{num}\ u^2 \to \mathsf{num}\ u.$$

Let $f = [\![e]\!](\rho_{\text{ops}})$. Then by Theorems 1 and 2, for any scaling environment $\psi \in \mathcal{E}_{\text{ops}}(\{u\})$, if $(v, v') \in \psi(u^2)_\perp$ then $(f(v), f(v')) \in \psi(u)_\perp$. Take ψ to be

$$\psi(u^n) \quad = \quad \{\ (r, k^n r) \ \mid \ r \in \mathbb{Q}\ \}$$

for some scale factor $k \in \mathbb{Q}^+$. First observe that if $f(\perp) = [r]$ then $r = 0$, so it follows from monotonicity of functions that $f(v) = [0]$ for any other argument v. In other words, if f is not strict then it must be the constant zero function. Now if $f([r]) = \perp$ then $f([k^2 r]) = \perp$, and if $f([r]) = [r']$ then $f([k^2 r]) = [kr']$. This cuts down the range of possible functions somewhat. If f diverges on *any* value without a rational root then it must diverge on all such values. Also, if it returns zero for any value without a rational root then it must return zero for all such values. So we have the intriguing possibility of a function which finds rational roots when they exist, but which otherwise returns zero or just loops. Amusingly, it is actually possible to write such a function simply by enumerating all rationals until reaching the root but looping if none exists. However, this function must be assigned the dimensionless type $\mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathbf{1}$; to prove formally that it cannot be polymorphic we use scaling environments defined by

$$\begin{aligned}\psi(u^{2n}) \quad &= \quad \{\ (r, k^n r) \ \mid \ r \in \mathbb{Q}\ \} \\ \psi(u^{2n+1}) \quad &= \quad \{\ (0, 0)\ \}\end{aligned}$$

for $k \in \mathbb{Q}^+$. Under such an environment, either $f([r]) = f([kr]) = \perp$ or $f([r]) = f([kr]) = [0]$. Hence if f diverges for any positive argument then it must diverge for all positive arguments, and if f returns zero for any positive argument then it must return zero for all positive arguments. Extending this to all arguments, it follows that f is characterised by

$$f([r]) = \begin{cases} v_1 & \text{if } r < 0, \\ v_2 & \text{if } r = 0, \\ v_3 & \text{if } r > 0, \end{cases}$$

where each of v_1, v_2 and v_3 is either $\perp$ or $[0]$. With the constant zero function included this yields just nine possible functions. □

We can make sense of this result in two ways. First, consider the operation of a root-finding function. It must start with an initial estimate for the root (type: $\mathsf{num}\ u$) and yet it cannot generate this estimate from its argument (type: $\mathsf{num}\ u^2$) using only the built-in arithmetic operations. The only value of type $\mathsf{num}\ u$ which it can construct is zero, and this is useless. A more semantic explanation is the following. The final estimate of the root is just that – an estimate – and the degree of error will scale with respect to the degree of error in the initial estimate. If this initial estimate is some fixed number then the error will depend on the units in which the argument is measured, and hence the function cannot be uniformly polymorphic in units of measure.

We are led to the conclusion that in order to write polymorphic root-finding functions it is necessary to provide an initial estimate for the root as an additional argument. For example, it is possible to write an approximating square root function with the type $\forall u.\mathsf{num}\ u \rightarrow \mathsf{num}\ u^2 \rightarrow \mathsf{num}\ u$. It is also interesting to note that one can write a unit-polymorphic function that accepts two numbers a and b and returns an approximation to $\sqrt{a^2+b^2}$. This function has the type $\forall u.\mathsf{num}\ u \rightarrow \mathsf{num}\ u \rightarrow \mathsf{num}\ u$, and would use some linear combination of a and b as its initial estimate for the root.

For the next example we formalise the use of the term *non-trivial*.

- A value $r \in \mathbb{Q}$ is non-trivial if $r \neq 0$.
- A value $f \in D \rightarrow E$ is non-trivial if $f(d)$ is non-trivial for some $d \in D$.
- A value $d' \in D_\perp$ is non-trivial if $d' = [d]$ for some non-trivial $d \in D$.

Example (First-order types). Consider the typing

$$\Gamma_{\text{ops}} \vdash e : \forall u_1 \dots \forall u_m.\mathsf{num}\ \mu_1 \rightarrow \cdots \rightarrow \mathsf{num}\ \mu_n \rightarrow \mathsf{num}\ \mu$$

where $\mu_j = \mathsf{num}(u_1^{a_{1j}} \cdots u_m^{a_{mj}})$ and $\mu = \mathsf{num}(u_1^{b_1} \cdots u_m^{b_m})$ for $a_{11}, \dots, a_{mn}, b_1, \dots, b_m \in \mathbb{Z}$. There exists an expression e with this typing with non-trivial meaning $f = [\![e]\!](\rho_{\text{ops}})$ if and only if there is a solution in integers $z_1, \dots, z_n$ to the equations

$$\begin{array}{rcl} a_{11}z_1 + \cdots + a_{1n}z_n &=& b_1 \\ &\vdots& \\ a_{m1}z_1 + \cdots + a_{mn}z_n &=& b_m. \end{array}$$

Proof. For the (if) part, we exhibit the term

$$\begin{array}{l}\Lambda u_1 \dots u_m. \\ \quad \lambda x_1 \colon \mathsf{num}\ \mu_1. \dots \lambda x_n \colon \mathsf{num}\ \mu_n.\ x_1^{z_1} * \cdots * x_n^{z_n}.\end{array}$$

For (only if), use the scaling environment

$$\psi(\mu) = \begin{cases} id_{\mathbb{Q}} & \text{if } \mu =_E \mu_1^{z_1} \cdots \mu_n^{z_n} \\ & \text{for some } z_1, \dots, z_n \in \mathbb{Z}, \\ \{(0,0)\} & \text{otherwise.} \end{cases}$$

□

Our final example is the type $\forall u_1.\forall u_2.(\mathsf{num}\ u_1 \rightarrow \mathsf{num}\ u_2) \rightarrow \mathsf{num}\ u_1 \cdot u_2$. Perhaps a function that calculates the area under a curve might have this type – intuitively, though, such a thing is impossible as there are no arguments representing bounds (a, b in $\int_a^b f(x)\,dx$) or increment values.

Example (Integration). There is no term e with typing

$$\Gamma_{\text{ops}} \vdash e : \forall u_1.\forall u_2.(\mathsf{num}\ u_1 \rightarrow \mathsf{num}\ u_2) \rightarrow \mathsf{num}\ u_1 \cdot u_2$$

and non-trivial meaning $f = [\![e]\!](\rho_{\text{ops}})$.

Proof. From Theorems 1 and 2 we know that for any $\psi \in \mathcal{E}_{\text{ops}}(\{u_1, u_2\})$ and $g, g' \in [\![\mathsf{num}\ u_1 \rightarrow \mathsf{num}\ u_2]\!]$,

$$(g, g') \in R^{\psi}_{\mathsf{num}\ u_1 \rightarrow \mathsf{num}\ u_2} \ \Rightarrow\ (f(g), f(g')) \in R^{\psi}_{\mathsf{num}\ u_1 \cdot u_2}.$$

Suppose that ψ is the following:

$$\psi(u_1^m \cdot u_2^n) \ \overset{\text{def}}{=}\ \begin{cases} id_{\mathbb{Q}} & \text{if } m = 0, \\ \{\ (0,0)\ \} & \text{otherwise.} \end{cases}$$

If we set $g' = g$ then $(g, g) \in R^{\psi}_{\mathsf{num}\ u_1 \rightarrow \mathsf{num}\ u_2}$, hence $(f(g), f(g)) \in R^{\psi}_{\mathsf{num}\ u_1 \cdot u_2}$ and so $f(g) \in \{\perp, [0]\}$. □

5.3 Type isomorphisms

Consider the types

$$\begin{array}{rcl} \tau_1 & \overset{\text{def}}{=} & \forall u.\mathsf{num}\ u \rightarrow \mathsf{num}\ u \rightarrow \mathsf{num}\ u \\ \text{and } \tau_2 & \overset{\text{def}}{=} & \mathsf{num}\ \mathbf{1} \rightarrow \mathsf{num}\ \mathbf{1}. \end{array}$$

If arguments are restricted to positive values, it can be shown that τ_1 is *isomorphic* to τ_2, that is, there exists a map i from values of type τ_1 to values of type τ_2, and a map j from τ_2 to τ_1 such that $i \circ j$ and $j \circ i$ are identities. Furthermore, these maps are *definable* in Λ_u:

$$\begin{array}{rcl} i & \overset{\text{def}}{=} & \lambda f \colon \tau_1.\ \lambda y \colon \mathsf{num}\ \mathbf{1}.\ f_{\mathbf{1}}(1)(y) \\ j & \overset{\text{def}}{=} & \lambda g \colon \tau_2.\ \Lambda u.\lambda x \colon \mathsf{num}\ u.\ \lambda y \colon \mathsf{num}\ u.\ x * g(y/x) \end{array}$$

Intuitively, a function of type τ_1 can be reduced to a function of type τ_2 if one of its arguments is considered to be the ‘units’ by which the other argument is measured.

Let us be more precise. The type τ of a function with n numeric arguments and a numeric result has a domain of values

$$[\![\tau]\!] = \mathbb{Q}_\perp \rightarrow \overset{n}{\dots} \rightarrow \mathbb{Q}_\perp \rightarrow \mathbb{Q}_\perp.$$

Let τ_1 and τ_2 be such types with m and n arguments respectively. Then τ_1 and τ_2 are isomorphic for positive values, written $\tau_1 \cong^+ \tau_2$, if there are definable functions $\Gamma_{\text{ops}} \vdash i : \tau_1 \to \tau_2$ and $\Gamma_{\text{ops}} \vdash j : \tau_2 \to \tau_1$ such that for any expressions $\Gamma_{\text{ops}} \vdash f : \tau_1$ and $\Gamma_{\text{ops}} \vdash g : \tau_2$, the equivalences

$$\begin{aligned} [\![j(i(f))]\!](\rho_{\text{ops}})(v_1)\cdots(v_m) &= [\![f]\!](\rho_{\text{ops}})(v_1)\cdots(v_m) \\ [\![i(j(g))]\!](\rho_{\text{ops}})(v_1)\cdots(v_n) &= [\![g]\!](\rho_{\text{ops}})(v_1)\cdots(v_n) \end{aligned}$$

hold for all positive or looping values v_i.

For our example above, the second equation follows directly from the underlying semantics, whereas the first relies crucially on dimensional invariance. This is similar to Wadler's demonstration that in System F the isomorphism $\tau \cong \forall t.(\tau \to t) \to t$ holds for any type τ, where parametricity is required to prove one direction of the isomorphism [14].

Why the restriction to positive values? Intuitively, there are more functions of type $\forall u.\mathsf{num}\ u \to \mathsf{num}\ u \to \mathsf{num}\ u$ than of type $\mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathbf{1}$ because the former can use the *signs* of two arguments to determine different courses of action, whereas the latter has the sign of only one argument available.

In Section 2.2 we outlined the technique of *dimensional analysis*: the reduction of an equation involving dimensioned quantities to an equation over a smaller number of dimensionless ones. We have just done something very similar, reducing a dimensionally-invariant function of two arguments to a dimensionless function of just one. In order to obtain a precise connection between these two ideas, we first state the central theorem of dimensional analysis.

Pi Theorem. *Fix a set of m base dimensions and let $x_1, \ldots, x_n$ be positive variables with the dimensions of x_i given by the i'th column of an $m \times n$ matrix A of dimension exponents. Then any dimensionally-invariant relation of the form*

$$f(x_1, \ldots, x_n) = 0$$

is equivalent to a relation

$$f'(\Pi_1, \ldots, \Pi_{n-r}) = 0$$

where r is the rank of the matrix A and $\Pi_1, \ldots, \Pi_{n-r}$ are dimensionless power-products of $x_1, \ldots, x_n$.

Proof. See Birkhoff [3]. □

We prove the following analogous result for first-order types in Λ_u.

Theorem 3 (Pi Theorem for Λ_u). *Let τ be a closed type of the form*

$$\forall u_1 \ldots \forall u_m.\mathsf{num}\ \mu_1 \to \cdots \to \mathsf{num}\ \mu_n \to \mathsf{num}\ \mu_0.$$

Let A be the $m \times n$ matrix of unit exponents in $\mu_1, \ldots, \mu_n$, and B the m-vector of unit exponents in μ_0. If the equation $AX = B$ is solvable for integer variables in X, then

$$\tau \cong^+ \mathsf{num}\ \mathbf{1} \to \overset{n-r}{\cdots} \to \mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathbf{1}$$

where r is the rank of A.

Proof. See Appendix A. □

Example (Pendulum). Suppose that the square of the period of a pendulum is determined by a function

$$\begin{aligned} \Gamma_{\text{ops}} \vdash e : \forall \mathrm{M}.\forall \mathrm{L}.\forall \mathrm{T}.\mathsf{num}\ \mathrm{M} &\to \mathsf{num}\ \mathrm{L} \\ &\to \mathsf{num}\ \mathrm{L} \cdot \mathrm{T}^{-2} \to \mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathrm{T}^2 \end{aligned}$$

whose arguments represent the mass and length of the pendulum, the acceleration due to gravity and the angle of swing. Then for positive argument values $\mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathbf{1}$ is an isomorphic type. □

It is possible to prove similar results for higher-order types, for example that the type of the differentiation function of Section 2.4 is isomorphic for positive values to $(\mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathbf{1}) \to (\mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathbf{1})$. A general result in the style of Theorem 3 is the subject of further research.

5.4 Relative definability

If Λ_u is extended with lists then one can define a recursive function to calculate the arithmetic mean of a list of n numbers according to the formula $\left(\sum_{i=1}^{n} a_i\right)/n$. The function would have the polymorphic type $\forall u.\mathsf{list}\ (\mathsf{num}\ u) \to \mathsf{num}\ u$. Now try writing a function *gm* to calculate the *geometric* mean according to the formula $\left(\prod_{i=1}^{n} a_i\right)^{1/n}$. At a first attempt, one immediately runs up against the problem that the product of the elements of the list has units which depend upon the length of the list. Although geometric mean has *behaviour* consistent with the polymorphic type $\forall u.\mathsf{list}\ (\mathsf{num}\ u) \to \mathsf{num}\ u$, its natural definition receives the typing

$$\Gamma_{\text{ops}} \vdash gm : \mathsf{list}\ (\mathsf{num}\ \mathbf{1}) \to \mathsf{num}\ \mathbf{1}.$$

But by a trick due to Rittri [12] one can turn this into a polymorphic function, as follows:

$$\begin{aligned} &\Lambda u.\lambda xs : \mathsf{list}\ (\mathsf{num}\ u). \\ &\quad abs(hd\ xs) * gm(map\ (\lambda x : \mathsf{num}\ u.\ x/abs(hd\ xs))\ xs) \end{aligned}$$

The trick is to use the magnitude of the first element of the list ($abs(hd\ xs)$) as a 'unit' by which to scale the whole list, pass this dimensionless list to *gm*, and then scale back again by the first element. The dimensional invariance of *gm* ensures that its semantics is preserved by the translation.

To formalise this idea, which we call *relative definability*, let $\tau^\star$ denote the *unit-erasure* of τ: the removal of all quantifiers and the replacement of $\mathsf{num}\,\mu$ with $\mathsf{num}\,\mathbf{1}$. Then given a closed type τ and a term e with typing $\Gamma_{\text{ops}} \vdash e : \tau^\star$ but behaviour $\Gamma_{\text{ops}} \models e : \tau$, we seek another another term e' with typing $\Gamma_{\text{ops}} \vdash e' : \tau$ such that $[\![e]\!](\rho_{\text{ops}}) = [\![e']\!](\rho_{\text{ops}})$.

Example. Suppose that a term has typing and behaviour specified by

$$\begin{aligned}&\Gamma_{\text{ops}} \vdash e : \mathsf{num}\,\mathbf{1} \to \mathsf{num}\,\mathbf{1}\\ &\Gamma_{\text{ops}} \models e : \forall u.\mathsf{num}\,u \to \mathsf{num}\,u^2.\end{aligned}$$

Let e' be the term

$$\begin{aligned}&\Lambda u.\lambda x\colon \mathsf{num}\,u.\\ &\quad e(\mathsf{if}\ x=0\ \mathsf{then}\ 0\ \mathsf{else}\ x/abs(x)) * abs(x) * abs(x).\end{aligned}$$

with the typing $\Gamma_{\text{ops}} \vdash e' : \forall u.\mathsf{num}\,u \to \mathsf{num}\,u^2$. Then $[\![e]\!](\rho_{\text{ops}}) = [\![e']\!](\rho_{\text{ops}})$.

Proof. Let $f = [\![e]\!](\rho_{\text{ops}})$ and $f' = [\![e']\!](\rho_{\text{ops}})$. Using Figure 3 and the usual interpretation of arithmetic constants it is easily shown that

$$f'(v) = \begin{cases} f(v) & \text{if } v = \bot \text{ or } v = [0]\,,\\ \bot & \text{if } v = [r_1] \text{ and } f([r_1/|r_1|]) = \bot,\\ \left[r_2|r_1|^2\right] & \text{if } v = [r_1] \text{ and } f([r_1/|r_1|]) = [r_2]\,.\end{cases}$$

From the dimensional invariance of e we know that for any $\psi \in \mathcal{E}_{\text{ops}}(\{u\})$ it is the case that

$$(v, v') \in \psi(u)_\bot \;\Rightarrow\; (f(v), f(v')) \in \psi(u^2)_\bot.$$

By setting $\psi(u^m) = \{\, (r, |r_1|^m r) \mid r \in \mathbb{Q} \,\}$ it follows that $f = f'$ as required. ☐

6 Full abstraction

It is well-known that standard cpo models of PCF are not *fully abstract* [10]; that is, denotational equality ($[\![e_1]\!] = [\![e_2]\!]$) does not coincide with observational equivalence ($e_1 \approx e_2$, which formulated denotationally states that $[\![\mathcal{C}[e_1]]\!] = [\![\mathcal{C}[e_2]]\!]$ for every program context $\mathcal{C}[\cdot]$). This is because there are certain elements in the model which are not definable by any term in the language, and these elements can be used to distinguish the denotations of observationally-equivalent terms. Naturally enough, these elements exist in the underlying semantics of Λ_u; moreover, they can even exhibit unit-polymorphic behaviour. For example, consider a function $p \in \mathbb{Q}_\bot \to \mathbb{Q}_\bot \to \mathbb{Q}_\bot$:

$$p(v_1)(v_2) = \begin{cases}[0] & \text{if } v_1 \neq \bot \text{ or } v_2 \neq \bot,\\ \bot & \text{otherwise.}\end{cases}$$

This is a version of the classic ‘parallel or’ function whose implementation requires parallel evaluation of its arguments. It is continuous, and furthermore is preserved by the scaling relation R_τ induced by $\mathcal{E}_{\text{ops}}$ for $\tau = \forall u.\mathsf{num}\,u \to \mathsf{num}\,u \to \mathsf{num}\,u$, but there is no expression e of type τ whose meaning is p. The function p can be used to distinguish the denotations of the following observationally-equivalent expressions (where Ω is a divergent term of type $\mathsf{num}\,\mathbf{1}$):

$$\begin{aligned} e_1 &\overset{\text{def}}{=} \lambda y\colon \tau.\, y_{\mathbf{1}}(\Omega)(0) + y_{\mathbf{1}}(0)(\Omega)\\ \text{and } e_2 &\overset{\text{def}}{=} \lambda y\colon \tau.\, y_{\mathbf{1}}(\Omega)(\Omega).\end{aligned}$$

Now consider the expressions

$$\begin{aligned} e_3 &\overset{\text{def}}{=} \lambda y\colon (\forall u.\mathsf{num}\,u \to \mathsf{num}\,u).\, y_{\mathbf{1}}(2)\\ \text{and } e_4 &\overset{\text{def}}{=} \lambda y\colon (\forall u.\mathsf{num}\,u \to \mathsf{num}\,u).\, 2 * y_{\mathbf{1}}(1).\end{aligned}$$

Although the underlying semantics assigns different meanings to e_3 and e_4, we know from dimensional invariance that they must be observationally equivalent. Similarly, because functions of type $\forall u.\mathsf{num}\,u^2 \to \mathsf{num}\,u$ cannot distinguish between different positive arguments, the expressions

$$\begin{aligned} e_5 &\overset{\text{def}}{=} \lambda y\colon (\forall u.\mathsf{num}\,u^2 \to \mathsf{num}\,u).\, y_{\mathbf{1}}(3)\\ \text{and } e_6 &\overset{\text{def}}{=} \lambda y\colon (\forall u.\mathsf{num}\,u^2 \to \mathsf{num}\,u).\, y_{\mathbf{1}}(5)\end{aligned}$$

are observationally-equivalent.

The scaling relation R^ψ_τ has proved to be remarkably powerful in a variety of applications. The ultimate test of its power would be to construct a model of the language in which the only incorrect distinctions between terms are due to the failure of full abstraction in the underlying semantics – so it distinguishes e_1 and e_2 but identifies e_3 with e_4 and e_5 with e_6.

To formalise this notion of *relative full abstraction*, first extend the notion of unit-erasure from Section 5.4 to terms: let $e^\star$ denote the removal of all unit abstractions and unit applications from e and the unit-erasure of type annotations. Because the underlying semantics of Figure 3 ignores unit annotations, it is easy to see that $[\![\tau]\!] = [\![\tau^\star]\!]$ and $[\![e]\!] = [\![e^\star]\!]$. We can now define two kinds of observational equivalence: unit-respecting equivalence ($e_1 \approx e_2$ iff $[\![\mathcal{C}[e_1]]\!] = [\![\mathcal{C}[e_2]]\!]$ for every program context $\mathcal{C}[\cdot]$) and underlying equivalence ($e_1 \approx^\star e_2$ iff $[\![\mathcal{C}^\star[{e_1}^\star]]\!] = [\![\mathcal{C}^\star[{e_2}^\star]]\!]$ for every dimensionless program context $\mathcal{C}^\star[\cdot]$). Consider the expressions studied above: we have $e_1 \approx e_2$ and $e_1 \approx^\star e_2$; $e_3 \approx e_4$ but $e_3 \not\approx^\star e_4$; $e_5 \approx e_6$ but $e_5 \not\approx^\star e_6$.

Now let $\{\![e]\!\}$ stand for the meaning of a term e in some more abstract model, and assume that the two models identify exactly the same unit-erased terms. Then we say that $\{\!|\cdot|\!\}$ is fully abstract *relative* to $[\![\cdot]\!]$ if

$$\begin{array}{rl} & \{\!|e_1|\!\} \neq \{\!|e_2|\!\} \text{ and } e_1 \approx e_2 \\ \text{implies} & [\![e_1]\!] \neq [\![e_2]\!] \text{ and } e_1 \approx^\star e_2. \end{array}$$

Can such a model be constructed? Space constraints prevent us giving details of a proposed model here, but the basic idea is to use the scaling relation to *quotient* the underlying semantics into equivalence classes. This is possible because for closed types τ, the relation R_τ is symmetric and transitive, making it a *partial equivalence relation*, that is, an equivalence relation on a subset of $[\![\tau]\!]$. By extending this idea to open types and open terms in an appropriate way, a PER can be constructed on the underlying meanings in $[\![\Gamma]\!] \rightarrow [\![\tau]\!]$. It is then possible to show that *if* the relative definability property of Section 5.4 holds for all types (an open problem), *then* the quotienting of the underlying semantics by this PER is fully abstract relative to the underlying semantics in the sense described above.

7 Conclusion

We have presented a novel application of Reynolds' notion of parametricity: the dimensional invariance of terms which are polymorphic in units of measure. As with conventional parametricity, this has allowed us to prove certain observational equivalences, to give conditions on the terms which inhabit a type, and to demonstrate isomorphisms between types. Furthermore it suggests a means of constructing an abstract model of the language that validates all the equivalences introduced by units of measure.

There is an apparent dependence of our results on the particular properties of the constants in ρ_{ops}. However, Theorem 1 itself is independent of these constants, and the idea of completeness captured by Theorem 2 adapts well to other constants. For example, if the comparison operation `<` is replaced by a function which compares the *magnitudes* of two values, then the Abelian group $\mathbb{Q}^+$ in Figure 5 is replaced by the Abelian group $\mathbb{Q} \setminus \{0\}$. Similarly, in a programming language with 'computable reals', a built-in square root function can be accommodated whilst still ruling out non-trivial functions of type $\forall u.\mathsf{num}\ u^3 \rightarrow \mathsf{num}\ u$.

Reynolds suggests that parametricity should not be limited to computation [11]. In this paper we have furnished one such instance: the invariance of physical laws under changes of scale. In general they are also invariant under changes in the *coordinate system*, given by a translation or rotation of the axes. Perhaps this too can be supported by the type system of a programming language.

Acknowledgements

Part of this work was carried out at the University of Cambridge Computer Laboratory with funding provided by an EPSRC studentship. I am extremely grateful for many productive discussions there with Nick Benton, Francis Davey, Alan Mycroft and Ian Stark and for comments on this paper from Simon Gay and Ian Stark.

References

[1] W. A. Adkins and S. H. Weintraub. *Algebra: An Approach via Module Theory.* Springer-Verlag, 1992.

[2] P. N. Benton. *Strictness Analysis of Lazy Functional Programs.* PhD thesis, University of Cambridge Computer Laboratory, August 1993. Technical Report 309.

[3] G. Birkhoff. *Hydrodynamics: A Study in Logic, Fact and Similitude.* Princeton University Press, Revised edition, 1960.

[4] Kim B. Bruce, Roberto Di Cosmo, and Giuseppe Long. Provable isomorphisms of types. *Mathematical Structures in Computer Science*, 2(2):231–247, June 1992.

[5] J. Goubault. Inférence d'unités physiques en ML. In P. Cointe, C. Queinnec, and B. Serpette, editors, *Journées Francophones des Langages Applicatifs, Noirmoutier*, pages 3–20. INRIA, 1994.

[6] R. T. House. A proposal for an extended form of type checking of expressions. *The Computer Journal*, 26(4):366–374, 1983.

[7] A. J. Kennedy. Dimension types. In *Proceedings of the 5th European Symposium on Programming*, volume 788 of *Lecture Notes in Computer Science*, pages 348–362. Springer-Verlag, 1994.

[8] A. J. Kennedy. *Programming Languages and Dimensions.* PhD thesis, Computer Laboratory, University of Cambridge, 1995. Available as Technical Report No. 391.

[9] D. H. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of Measurement*, volume I, III. Academic Press, 1971, 1990.

[10] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[11] J. C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523, Amsterdam, 1983. Elsevier Science Publishers B. V. (North-Holland).

[12] M. Rittri. Dimension inference under polymorphic recursion. In *7th ACM Conf. on Functional Programming Languages and Computer Architecture*, pages 147–159. ACM Press, June 1995.

[13] K. Sieber. Reasoning about sequential functions via logical relations. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science: Proceedings of the LMS Symposium, Durham, 1991.* Cambridge University Press, 1992. LMS Lecture Notes Series, 177.

[14] P. Wadler. Theorems for free! In *Proceedings of the 4th International Symposium on Functional Programming Languages and Computer Architecture*, 1989.

[15] M. Wand and P. M. O'Keefe. Automatic dimensional inference. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 479–486. MIT Press, 1991.

[16] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction.* The MIT Press, 1993.

A Omitted proofs

Theorem 2 (Completeness of $\mathcal{E}_{\mathrm{ops}}$). *Suppose that $\mathcal{E}$ induces $R_{\Gamma_{\mathrm{ops}}}$. Then*

$$(\rho_{\mathrm{ops}}, \rho_{\mathrm{ops}}) \in R_{\Gamma_{\mathrm{ops}}} \iff \mathcal{E} \subseteq \mathcal{E}_{\mathrm{ops}}.$$

Proof.
($\Leftarrow$). The verification that ρ_{ops} is invariant under $R_{\Gamma_{\mathrm{ops}}}$ is straightforward.
($\Rightarrow$). We can deduce from the antecedent that for any $\mathcal{V}$, $\psi \in \mathcal{E}(\mathcal{V})$, and $\mu, \mu_1, \mu_2 \in \mathrm{Units}(\mathcal{V})$:

0: $(0,0) \in \psi(\mu)$.

1: $(1,1) \in \psi(\mathbf{1})$.

+: If $(r_1, r_1') \in \psi(\mu)$ and $(r_2, r_2') \in \psi(\mu)$ then $(r_1 + r_2, r_1' + r_2') \in \psi(\mu)$.

-: If $(r_1, r_1') \in \psi(\mu)$ and $(r_2, r_2') \in \psi(\mu)$ then $(r_1 - r_2, r_1' - r_2') \in \psi(\mu)$.

*: If $(r_1, r_1') \in \psi(\mu_1)$ and $(r_2, r_2') \in \psi(\mu_2)$ then $(r_1 r_2, r_1' r_2') \in \psi(\mu_1 \cdot \mu_2)$.

/: If $(r_1, r_1') \in \psi(\mu_1)$ and $(r_2 \neq 0, r_2' \neq 0) \in \psi(\mu_2)$ then $(r_1/r_2, r_1'/r_2') \in \psi(\mu_1 \cdot \mu_2^{-1})$.

<: If $(r_1, r_1') \in \psi(\mu)$ and $(r_2, r_2') \in \psi(\mu)$ then $r_1 < r_2$ iff $r_1' < r_2'$.

We proceed to the consequent in two steps.

1. We show that

$$\psi(\mu) = \begin{array}{l} \textit{either}\ \{ (0,0) \} \\ \textit{or}\ \{ (r, kr) \mid r \in \mathbb{Q} \}\ \text{for some } k \in \mathbb{Q}^+. \end{array}$$

We already know that $(0,0) \in \psi(\mu)$ for any μ. Now consider $(r, r') \in \psi(\mu)$ for r and r' not both zero. From the interpretation of < we can deduce that $r < 0$ iff $r' < 0$ and that $0 < r$ iff $0 < r'$. Hence $r \neq 0$ and $r' = kr$ for some $k \in \mathbb{Q}^+$.

Now pick any two pairs $(r, kr), (r', k'r') \in \psi(\mu)$. From the interpretation of / we know that $(r/r', (kr)/(k'r')) \in \psi(\mathbf{1})$. As $\psi(\mathbf{1}) = id_{\mathbb{Q}}$ we can deduce that $r/r' = (kr)/(k'r')$ so $k = k'$ giving the bijection as required.

2. We show that if the constants in ρ_{ops} are invariant under the scaling relation $R_{\Gamma_{\mathrm{ops}}}$ induced by $\mathcal{E}$ then for any $\psi \in \mathcal{E}(\mathcal{V})$ there is some subgroup $G \subseteq \mathrm{Units}(\mathcal{V})$ and $h \in \hom(G, \mathbb{Q}^+)$ such that $\psi = \psi_{G,h} \in \mathcal{E}_{\mathrm{ops}}(\mathcal{V})$ as defined in Figure 5.

Let G be the set

$$G = \{ \mu \in \mathrm{Units}(\mathcal{V}) \mid \psi(\mu) \neq \{(0,0)\} \},$$

and assuming step (1), define h by:

$$h(\mu) = k \quad \text{if } \psi(\mu) = \{ (r, kr) \mid r \in \mathbb{Q} \}.$$

Pick any $\mu_1, \mu_2 \in G$. Then

$$\begin{array}{rcl} \psi(\mu_1) & = & \{ (r, k_1 r) \mid r \in \mathbb{Q} \} \\ \psi(\mu_2) & = & \{ (r, k_2 r) \mid r \in \mathbb{Q} \} \end{array}$$

for some $k_1, k_2 \in \mathbb{Q}^+$. From the interpretation of * we know that

$$\psi(\mu_1 \cdot \mu_2) = \{ (r, k_1 k_2 r) \mid r \in \mathbb{Q} \}$$

and from this deduce that $\mu_1 \cdot \mu_2 \in G$ and that $h(\mu_1 \cdot \mu_2) = k_1 k_2 = h(\mu_1) h(\mu_2)$. From the interpretation of / and 1 we also know that

$$\psi(\mu_1^{-1}) = \{ (r, r/k_1) \mid r \in \mathbb{Q} \}$$

and from this deduce that $\mu_1^{-1} \in G$ and that $h(\mu_1^{-1}) = 1/k_1 = 1/h(\mu_1)$. We already know that $h(\mathbf{1}) = 1$. Thus we have just shown that G is a subgroup of $\mathrm{Units}(\mathcal{V})$, and that h is a homomorphism from G into $\mathbb{Q}^+$. This completes the proof.

□

Theorem 3 (Pi Theorem for Λ_u). *Let τ be a closed type of the form*

$$\forall u_1 \ldots \forall u_m . \mathsf{num}\, \mu_1 \to \cdots \to \mathsf{num}\, \mu_n \to \mathsf{num}\, \mu_0.$$

Let A be the $m \times n$ matrix of unit exponents in $\mu_1, \ldots, \mu_n$, and B the m-vector of unit exponents in μ_0. If the equation $AX = B$ is solvable for integer variables in X, then

$$\tau \cong^+ \mathsf{num}\,\mathbf{1} \to \overset{n-r}{\cdots} \to \mathsf{num}\,\mathbf{1} \to \mathsf{num}\,\mathbf{1}$$

where r is the rank of A.

Proof. To save space, let $\vec{u} = u_1 \ldots u_m$ and $\tau_i = \mathsf{num}\, \mu_i$. We first identify a class of isomorphisms which do not rely on dimensional invariance and whose bijections simply manipulate unit expressions. Think of them as changing the set of base units used, for example replacing units of length and time by units of velocity and acceleration. They can be classified into three subclasses:

(R1). The exchange of two base units:

$$\forall \vec{u}.\tau \cong \forall \vec{u}.\{u_i \mapsto u_j, u_j \mapsto u_i\}\tau.$$

(R2). The replacement of a base unit by its inverse:

$$\forall \vec{u}.\tau \cong \forall \vec{u}.\{u_i \mapsto u_i^{-1}\}\tau.$$

(R3). The invertible combination of two units (for $i \neq j$ and $z \in \mathbb{Z}$):

$$\forall \vec{u}.\tau \cong \forall \vec{u}.\{u_i \mapsto u_i \cdot u_j^z\}\tau$$

A second class of isomorphisms manipulates the *arguments* in a way analogous to the manipulation of units seen above. Again, dimensional invariance is not required for their validation; the appropriate bijections are straightforward and are omitted.

(C1). The exchange of two arguments:

$$\begin{array}{rl} & \forall \vec{u}.\tau_1 \to \cdots \to \tau_i \to \cdots \to \tau_j \to \cdots \to \tau_n \to \tau_0 \\ \cong & \forall \vec{u}.\tau_1 \to \cdots \to \tau_j \to \cdots \to \tau_i \to \cdots \to \tau_n \to \tau_0. \end{array}$$

(C2). The inversion of an argument:

$$\begin{array}{rl} & \forall \vec{u}.\tau_1 \to \cdots \mathsf{num}\ \mu_i \cdots \to \tau_n \to \tau_0 \\ \cong & \forall \vec{u}.\tau_1 \to \cdots \mathsf{num}\ \mu_i^{-1} \cdots \to \tau_n \to \tau_0. \end{array}$$

(C3). The invertible combination of two arguments (for $i \neq j$ and $z \in \mathbb{Z}$):

$$\begin{array}{rl} & \forall \vec{u}.\tau_1 \to \cdots \mathsf{num}\ \mu_i \to \cdots \to \tau_n \to \tau_0 \\ \cong^+ & \forall \vec{u}.\tau_1 \to \cdots \mathsf{num}\ (\mu_i \cdot \mu_j^z) \cdots \to \tau_n \to \tau_0. \end{array}$$

This relies on the j'th argument being non-zero, which explains $\cong^+$.

Consider the matrix of integers A' associated with the units of the arguments in the resulting types above. The isomorphisms (R1), (R2) and (R3) correspond to the application of elementary *row* operations to A to produce A' (the exchange of two rows, the multiplication of a row by -1, and the addition of a scalar multiple of one row to another row). Similarly, the isomorphisms (C1), (C2) and (C3) correspond to elementary *column* operations. The application of a set of elementary row operations to a matrix A is equivalent to multiplication on the left by an invertible matrix U to give UA. Similarly, the application of a set of column operations to a matrix A is equivalent to multiplication on the right by an invertible matrix V to give AV. Furthermore, for any $m \times n$ matrix A, there exists an invertible $m \times m$ matrix U and invertible $n \times n$ matrix V such that

$$UAV = \begin{pmatrix} D_r & 0 \\ 0 & 0 \end{pmatrix}$$

where 0 denotes a block of zeroes, $r = \mathrm{rank}(A)$ and D_r is an $r \times r$ matrix with positive diagonal elements $s_1, \ldots, s_r$ and zeroes elsewhere, such that s_i divides s_{i+1} for $1 \leqslant i \leqslant r-1$. The matrix UAV is unique and is known as the *Smith normal form* of A [1]. The values $s_1, \ldots, s_r$ are its *invariant factors*.

Using this particularly simple form, we can reduce our original problem to a much simpler one. By a composition of type isomorphisms represented by elementary row and column operations on the matrix A, we obtain a new type whose arguments have units given by UAV and whose result has units given by $UB = (c_1, \ldots, c_m)$:

$$\begin{array}{rl} \tau \cong^+ & \forall \vec{u}.\mathsf{num}\ u_1^{s_1} \to \cdots \to \mathsf{num}\ u_r^{s_r} \\ & \to \mathsf{num}\ \mathbf{1} \overset{n-r}{\to \cdots \to} \mathsf{num}\ \mathbf{1} \to \mathsf{num}\ (u_1^{c_1} \cdots u_m^{c_m}). \end{array}$$

Recall an assumption made in the statement of the theorem: that the equation $AX = B$ is solvable for integers in X (if this is not the case then all terms of type τ are trivial as was shown in Section 5.2). Hence $UA(VV^{-1})X = UB$, and this is the case if and only if $UAVY = UB$ is solvable for integers in Y. Therefore we must have $c_i = s_i z_i$ for some $z_i \in \mathbb{Z}$ when $1 \leqslant i \leqslant r$ and $z_i = 0$ when $r+1 \leqslant i \leqslant m$. By a trivial isomorphism we can remove the superfluous bound variables $u_{r+1}, \ldots, u_m$ and rewrite the above as

$$\begin{array}{rl} \tau \cong^+ & \forall u_1 \ldots u_r.\mathsf{num}\ u_1^{s_1} \to \cdots \to \mathsf{num}\ u_r^{s_r} \\ & \to \mathsf{num}\ \mathbf{1} \overset{n-r}{\to \cdots \to} \mathsf{num}\ \mathbf{1} \to \mathsf{num}\ (u_1^{s_1 z_1} \cdots u_r^{s_1 z_r}). \end{array}$$

Now define bijections in Λ_u between this type (call it τ') and the dimensionless type

$$\tau'' = \mathsf{num}\ \mathbf{1} \overset{n-r}{\to \cdots \to} \mathsf{num}\ \mathbf{1} \to \mathsf{num}\ \mathbf{1},$$

as follows:

$$\begin{array}{rl} I = & \lambda f : \tau'.\lambda x_1 : \mathsf{num}\ \mathbf{1} \ldots \lambda x_{n-r} : \mathsf{num}\ \mathbf{1}. \\ & f_{\mathbf{1} \ldots \mathbf{1}}(1) \cdots (1)(x_1) \cdots (x_{n-r}) \end{array}$$

$$\begin{array}{rl} J = & \lambda g : \tau''.\Lambda u_1 \ldots u_r. \\ & \lambda x_1 : \mathsf{num}\ u^{s_1} \ldots \lambda x_r : \mathsf{num}\ u^{s_r}. \\ & \lambda x_{r+1} : \mathsf{num}\ \mathbf{1} \ldots \lambda x_n : \mathsf{num}\ \mathbf{1}. \\ & x_1^{z_1} * \cdots * x_r^{z_r} * g(x_{r+1}) \cdots (x_n) \end{array}$$

In a similar fashion to the example sketched in Section 5.3 we can prove that these bijections compose to give the identity with respect to terms of types τ' and τ''. As before, one direction (showing that $J(I(f))$ is equivalent to f for positive values) requires the use of dimensional invariance (at last!). $\square$