5-1994

Fields in Physics are like Curried Functions or Physics for Functional Programmers

Gary T. Leavens
Iowa State University

Fields in Physics are like Curried Functions
or
Physics for Functional Programmers

Gary T. Leavens

TR #94-06b
April 1994, revised May 1994

Keywords: field, physics, functional programming, currying, teaching, examples.
1994 CR Categories: D.1.1 [*Programming Techniques*] Applicative (Functional) Programming; J.2 [*Physical Sciences and Engineering*] Physics; K.3.2 [*Computers and Education*] Computer and Information Science Education — Computer science education.

Department of Computer Science
229 Atanasoff Hall
Iowa Sate University
Ames, Iowa 50011-1040, USA

Fields in Physics are like Curried Functions or Physics for Functional Programmers

Gary T. Leavens
Department of Computer Science, 229 Atanasoff Hall
Iowa State University, Ames, Iowa 50011-1040 USA
leavens@cs.iastate.edu

May 6, 1994

Abstract

Good motivating examples for teaching the utility of curried functions can be taken from Physics. The curried function perspective can also be used to help functional programmers understand fields in Physics. The correspondence between the curried function view of vector fields and the usual view taken in Physics is also explained.

1 Introduction

The earth and everything in it is held together by forces that can be modeled using curried functions. Yet many students think that curried functions have little to do with their everyday experience, and so have trouble grasping the concept. Typical examples, given in texts on programming languages or functional programming, are curried mapping and reduction functions on lists, which connect only with students' experience in programming. A novel[1] approach is to draw more compelling examples from Physics.

Once one understands curried functions, it is easy to understand the concept of a field in Physics. Fields are similar to curried functions, although vector fields in Physics are modeled in a slightly different but mathematically equivalent way.

The rest of this note is organized as follows. Section 2 gives some background on the Newtonian gravitational field used for a running example, and the programming language Scheme used in this note for examples. Section 3 presents the way in which examples from Physics can be used to teach curried functions. Section 4 explains how functional programmers can use their understanding of curried functions to understand the concept of fields

[1]Gerald J. Sussman apparently discussed something like this in his invited lecture at the 1982 *ACM Symposium on LISP and Functional Programming.* Sussman says that he never wrote down his lecture. Although his lecture was titled "Teaching the Control of Complexity," the idea does not seem to have made it into the practice of teaching functional programming, and it does not appear in [ASS85].

```
(define G 6.670e-11)      ; N·m^2/kg^2

(define grav-force
   (lambda (m1 r m2)      ; kg m kg
      (if (zero? r)
         0.0
         (- (/ (* G (* m1 m2))
               (square r))))))

(define square
   (lambda (num)
      (* num num)))
```

Figure 1: Definitions of the universal gravitational constant `G`, and the function `grav-force` in Scheme. The auxiliary Scheme function `square` returns the square of its actual parameter.

in Physics, and discusses how the view of fields as curried functions can be reconciled with the standard interpretation of fields in Physics. Section 5 offers some conclusions.

2 Background

A simple physical example that can be used to teach curried functions is Newton's law of gravity [New87] [FLS63, Chapter 7]. Coding Newton's law in the programming language Scheme [SS78] [RCA+86] [IEE91] [ASS85] [SF89] will help explain Scheme's notation and curried functions.

2.1 Newton's Law of Universal Gravitation

The magnitude of the gravitational force, F, between two masses, m_1 and m_2, separated by distance (radius) r is given by the following equation.

$$F = \frac{G m_1 m_2}{r^2} \tag{1}$$

In the above equation, G is the universal gravitational constant. Using International System (SI) units, G is 6.670×10^{-11} N·m^2/kg^2, where "N" stands for "newtons" (1 newton = 1 kg·m/s^2), "m" stands for "meters", "kg" stands for "kilograms", and "s" stands for "seconds." With SI units, m_1 and m_2 would be given in kilograms, r in meters, and thus F in newtons.

2.2 Scheme

Equation (1) implicitly defines the force, F, as a function of m_1, r, and m_2. Figure 1 codes this as the function `grav-force` in Scheme. The code refines the equation, by defining the force to be zero, if the distance r is zero.

In Scheme **define** introduces a name being defined; the name is followed by an expression whose value becomes the name's value. There are three **define** expressions in Figure 1. The first such **define** binds the name `G` to the universal gravitational constant in SI units. A semicolon (`;`) starts a comment, which continues to the end of the line; the first comment gives the units of `G`.

The second **define** binds the name `grav-force` to a Scheme function. A **lambda** expression makes a Scheme function, with formal parameters named within the following set of parentheses, and a body which is an expression. The function `grav-force` thus has three formal parameters: `m1`, `m2`, and `r`. When a function is called, it returns as its value the result of the expression that is its body, using the actual parameter values as the values of its formal parameters. For example, the function call

```
(grav-force 5.96e24 6.37e6 68.0)
```

has as its approximate value 666.2 newtons; this is the magnitude of the gravitational force when m_1 is 5.96×10^{24} kilograms (the earth's mass), r is 6.37×10^6 meters (the earth's radius), and m_2 is 68.0 kilograms. As above, the application of a Scheme function, `f`, to an actual parameter, `x`, is written `(f x)`. Even arithmetic operators, such as multiplication are written this way; for example, the expression `(* r r)` is the square of `r`. An **if** expression of the form (**if** b e_2 e_3) returns the value of e_2 if the value of the test b is true, and otherwise returns the value of e_3. The built-in Scheme predicate, `zero?`, used in Figure 1, returns true just when its argument is zero.

3 The Gravitational Force Example

This section shows various ways to explain curried functions to students. It illustrates the kind of example that can be drawn from physics. (Other examples include the dielectric force law.)

A curried version of the Scheme function `grav-force` is given in Figure 2. Currying a function with more than one argument means expressing that function using nested one-argument functions [Fre91, pages 153–156] [Sch24] [Cur30] [CFC58] [Bar84, Page 6] [SF89, Section 7.3]. As a start to explaining this concept, the Scheme expression

```
(((grav-force-c 5.96e24) 6.37e6) 68.0)
```

also has as its approximate value 666.2 newtons. In general the Scheme function `grav-force-c` is such that for all m_1, r, and m_2, the following equation between Scheme expressions holds.

$$\texttt{(((grav-force-c } m_1\texttt{) } r\texttt{) } m_2\texttt{)} = \texttt{(grav-force } m_1 \ r \ m_2\texttt{)} \qquad (2)$$

In the Scheme expression `(((grav-force-c` m_1`)` r`)` m_2`)`, the Scheme function `grav-force-c` is applied to m_1, and this returns another Scheme function. The function returned is the one defined by the expression (**lambda** `(r) ...`) in Figure 2. Note that when this function is defined, there is already a value for `m1`, which this function will ultimately use. That Scheme

```
(define grav-force-c
   (lambda (m1)     ; kg
      (lambda (r)      ; m
         (lambda (m2)     ; kg
            (if (zero? r)
                0.0
                (- (/ (* G (* m1 m2))
                      (square r))))))))
```

Figure 2: The curried function `grav-force-c`. The universal gravitational constant, `G`, and the Scheme function `square` are defined above.

function is applied to r, and that application returns another function; this function is the one defined by the expression `(lambda (m2) ...)` in Figure 2. This function is then applied to m_2, which returns a number.

Another way to understand curried functions is by looking at their types[2]. To start with the type of a function that is not curried, the type of `grav-force` is as follows.

$$[(\mathrm{kg} \times \mathrm{m} \times \mathrm{kg}) \to \mathrm{N}] \tag{3}$$

In the above notation, "kg" stands for a set of numbers that are thought of as kilograms, "m" stands for a set of numbers that are thought of as meters, "N" stands for a set of numbers that are thought of as newtons, and the notation $[S \to T]$ means the set of all functions with domain S and range T. That is, `grav-force` has a type that is the set of functions whose domain is triples of kilograms, meters, and kilograms, and whose range is newtons. The type of the curried function `grav-force-c` is as follows.

$$[\mathrm{kg} \to [\mathrm{m} \to [\mathrm{kg} \to \mathrm{N}]]] \tag{4}$$

That is, `grav-force-c` has a type that is the set of functions whose domain is kilograms, and whose range is the set of functions $[\mathrm{m} \to [\mathrm{kg} \to \mathrm{N}]]$. Thus the range type of `grav-force-c` is the set of functions whose domain is meters, and whose range is the set of functions $[\mathrm{kg} \to \mathrm{N}]$. Using the notation, "$x : T$", to mean that x has type T, the table in Figure 3 illustrates the type of `grav-force-c` using examples.

The above discussion explains what curried functions are, but it does not explain to students why curried functions are useful. Curried functions are useful as tool-makers, since a curried function is a function that produces other functions, which can be used to do useful computations (including making other functions). One way to illustrate this is by a series of applications, and discussions about the utility of each step.

[2]What is meant by "type" is similar to what physicists mean by "unit". For purposes of this note, a type is a set. A value "has a type" if it belongs to that set. In Scheme functions are also values, and thus have types.

Scheme expression	type
`grav-force-c`	$[\mathrm{kg} \rightarrow [\mathrm{m} \rightarrow [\mathrm{kg} \rightarrow \mathrm{N}]]]$
`5.96e24`	kg
`(grav-force-c 5.96e24)`	$[\mathrm{m} \rightarrow [\mathrm{kg} \rightarrow \mathrm{N}]]$
`6.37e6`	m
`((grav-force-c 5.96e24) 6.37e6)`	$[\mathrm{kg} \rightarrow \mathrm{N}]$
`68.0`	kg
`(((grav-force-c 5.96e24) 6.37e6) 68.0)`	N

Figure 3: Scheme expressions and their types, illustrating the type of the curried function `grav-force-c`.

For example, to work with the gravitational force exerted by the earth, one passes to `grav-force-c` the mass of the earth. This is done in the definition of the function `earths-force-fun` below, which assigns to each radius from the earth's center a function that assigns to each mass placed at that distance a force.

```
(define mass-of-earth 5.96e24)       ; kg
(define earths-force-fun             ; type:  [m → [kg → N]]
   (grav-force-c mass-of-earth))
```

Passing any other mass, such as the mass of the galaxy or a student to `grav-force-c` gives the analogous function for that mass.

As an example of how to use `earths-force-fun`, to work with the gravitational force of the earth at the earth's surface, one can pass the earth's radius to `earths-force-fun`. This is done below in the definition of the SML function `earths-force-at-surface`. This function assigns to each mass at the earth's surface the magnitude of the force exerted by the earth's gravity on that mass.

```
(define radius-of-earth 6.37e6)       ; m
(define earths-force-at-surface       ; type:  [kg → N]
   (earths-field radius-of-earth))
```

Passing any other radius from the earth's center to `earths-force-fun`, such as the distance from the earth's center to the orbit of the space shuttle or to the sun or moon, gives an analogous function for that radius.

As an example of how to use `earths-force-at-surface`, the following expression computes the gravitational force exerted by the earth (at its surface) on a mass of 68 kilograms (about 150 pounds).

```
(earths-force-at-surface 68.0)
```

The value of this expression is approximately 666.2 newtons. Using a unit mass, one can find the acceleration per unit mass at the earth's surface, which is about 9.8 meters per second squared.

4 Using Curried Functions to Understand Fields

The previous section showed how examples from Physics can be used as an aid to understanding the concept of curried functions. This section shows how the concept of curried functions can be used as an aid to understanding the concept of a field in Physics. The gravitational field of a mass is again used as an example. Before doing that, a bit more background on Physics is needed.

4.1 Force as a Vector Quantity

Forces are applied in a particular direction[3]; for example, the force on a ball is downwards, so that when one lets go of the ball it falls. Thus, a more accurate statement of Newton's law uses the concept of vectors [FLS63, Chapters 11–12].

The distance from one mass to another is a directed quantity, and so is written as a vector. This vector can be thought of as the coordinates of the second mass in three-dimensional space, and thus has x, y, and z components. A force vector would have forces for its x, y, and z components, where the number in each component tells what part of the force vector can be thought of as directed along each axis.

In terms of vectors, Newton's law of gravity can be stated as follows [FLS63, Chapter 12], where $\vec{F}$ is the force vector, $\vec{r} = \langle r_x, r_y, r_z \rangle$ is the radius vector (the vector from m_1's location to m_2's), and $r = \sqrt{r_x^2 + r_y^2 + r_z^2}$ is the distance from m_1's location to m_2's.

$$\begin{aligned} \vec{F} &= \frac{G m_1 m_2}{r^2} \cdot \left(\frac{-\vec{r}}{r} \right) & (5) \\ &= \frac{-G m_1 m_2}{r^3} \cdot \vec{r} & (6) \end{aligned}$$

Formula (6) is what a physicist would write. The r^3 in the denominator of this formula is explained by Formula (5), which shows the magnitude of the force multiplied by a unit vector $(-\vec{r}/r)$ in the direction opposite to $\vec{r}$.

To code vector functions in Scheme, the mathematical vectors used in physics will represented in Scheme using lists.[4] For example, the vector $\langle 1, 2, 3 \rangle$ will be represented by the Scheme expression `(list 1 2 3)`.

The Scheme function `gravity-law` defined in Figure 4 codes Equation (6). The function `gravity-law` is not curried; but takes m_1, the radius vector $\vec{r}$, and m_2 as its three parameters. In the type comment for `gravity-law`, the unit m* means a vector of meters, represented as a list, and N* means a vector of newtons, represented as a list. The **let** expression is local definition construct; it binds `r` to the value of `(distance r-vector)` and returns the result of the expression following the double right parentheses. This expression does the scalar multiplication called for

[3]The idea of using directed quantities in mechanics goes back at least as far as Aristotle [Ari61, Book V] [Dug55, Page 21].

[4]Scheme's vectors, which are like arrays in other programming languages, could also be used.

```
(define gravity-law    ; type: [kg × m* × kg → N*]
   (lambda (m1 r-vector m2)
      (let ((r (distance r-vector)))
         (scalar-multiply
            (- (/ (grav-force m1 r m2)
                  r))
            r-vector))))

(define distance       ; type: [m* → m]
   (lambda (r-vector)
      (sqrt (sum (map square r-vector)))))

(define sum            ; type: [(m^2)* → m^2]
   (lambda (vec)
      (if (null?  vec)
         0.0
         (+ (first vec) (sum (rest vec))))))

(define scalar-multiply
   (lambda (scalar vec)
      (map (lambda (vec-component)
              (* scalar vec-component))
           vec)))
```

Figure 4: The Scheme function **gravity-law**, which models Newton's law of Universal gravitation, and some auxiliary functions.

in Equation (6). (The scalar is the first argument to `scalar-multiply`, which represents $(-Gm_1m_2)/r^3$, and the vector is `r-vector`.)

The following approximate equation between Scheme expressions is an example of using `gravity-law`.

```
(gravity-law 5.96e24 (list 1.0 0.0 6.37e6) 68.0)
 ≈ (list -1.05e-4 0.0 -666.2)                          (7)
```

The other Scheme functions defined in Figure 4 are auxiliary functions. The Scheme function `distance`, is such that if its argument is a list representing the vector $\langle r_x, r_y, r_z \rangle$, then it returns $\sqrt{r_x^2 + r_y^2 + r_z^2}$. It uses the built-in Scheme function `map` to square the components of its argument, the auxiliary function `sum` to add the squares, and the built-in function `sqrt` to find the positive square root of the sum of the squares. The Scheme function `scalar-multiply` is such that if its arguments are s and a list representing a vector $\langle r_x, r_y, r_z \rangle$, then it returns a list representing the vector $\langle sr_x, sr_y, sr_z \rangle$.

```
(define gravity-law-c    ; type:  [kg → [m* → [kg → N*]]]
   (lambda (m1)              ; kg
      (lambda (r-vector)       ; m*
         (lambda (m2)            ; kg
            (let ((r (distance r-vector)))
               (scalar-multiply
                  (- (/ (grav-force m1 r m2)
                        r))
                  r-vector))))))
```

Figure 5: The curried function `gravity-law-c`, which is used to explain the concept of fields. The auxiliary functions `distance`, `scalar-multiply`, and `grav-force` are defined above.

4.2 A Curried Function View of Fields

Currying the function `gravity-law` defined in Figure 4 gives a curried function that can be used to explain the concept of a gravitational field. The curried version of `gravity-law` is shown in Figure 5.

The gravitational field of a given mass, m_1, in this view, is modeled by the curried function, `(gravity-law-c` m_1`)`. This function is the function defined by `(`**lambda** `(r-vector)` ...`)` in Figure 5. It assigns to each radius vector (point in the space around m_1) another function (the one defined by `(`**lambda** `(m2)` ...`)` in Figure 5). That function assigns to each mass placed at the point given by `r-vector` a force vector.

For example, the earth's gravitational field can be modeled by the function `earths-field` declared in the following code. The function `earths-field` assigns to each point around the earth a function, which function assigns to each mass placed at that point a force vector.

```
(define earths-field    ; type:  [m* → [kg → N*]]
   (gravity-law-c mass-of-earth))
```

The earth's gravitational field at a point on the earth's surface can be modeled by the function `earths-field-at-surface` declared in the following code. The function `earths-field-at-surface` assigns to each mass placed at the given point the force vector exerted (downward) by the earth's gravity on that mass.

```
(define earths-field-at-surface    ; type:  [kg → N*]
   (let ((point-on-earth (list 1.0 0.0 6.37e6)))    ; m*
      (earths-field point-on-earth)))
```

Finally, the following expression computes the gravitational force vector exerted by the earth at a point on its surface on a mass of 68 kilograms (about 150 pounds).

```
(earths-field-at-surface 68.0)      ; N*
```

The value of this expression is approximately the following force vector (represented as a list) `(list -1.05e-4 0.0 -666.2)`.

4.3 Vector Fields in Physics

Physics does not use the curried function view of fields. Instead a physicist thinks of a field as a function that assigns to each point in space a quantity [FLS63, Section 12-4]. In this view, the gravitational field of the earth is a function that assigns to each point around the earth a vector, whose units are newtons per kilogram. This vector gives the force vector per unit mass at the given point. The force vector that would result from putting a mass m_2 at such a point is obtained by multiplying this force per unit mass vector by m_2.

Historically, this view of fields was made standard by the tremendous success of Maxwell's mathematics of the electromagnetic field [Max64]. Although this view of fields may have been invented before Maxwell, Maxwell's use of it certainly came before curried functions were conceived by Frege [Fre91], and well before curried functions were used by Schönfinkel [Sch24], or made widely known by Curry [Cur30] [CFC58]. At that time there were no units for curried functions in Physics, which remains the case today. Instead Maxwell described his mathematics by using an analogy to fluid flow, in which the field potential is likened to the fluid's pressure, which is measured in force per unit area [Tri66, page 105–106] [Max64, part II (49)]. Maxwell's view of fields is perfectly satisfactory in physics, because fields have a physical reality [Far52] [FLS63, Chapter 28].

As in the Feynman lectures [FLS63, Section 12-4], this view of fields can be thought of as factoring Formula (6) as follows.

$$\vec{F} = m_2 \cdot \frac{-Gm_1\vec{r}}{r^3} \tag{8}$$

This vector quantity, $(-Gm_1\vec{r})/r^3$, is the gravitational field of the mass m_1. This field is written as $\vec{C}$ in the Feynman lectures, and is defined as follows.

$$\vec{C} = \frac{-Gm_1\vec{r}}{r^3} \tag{9}$$

Thus one can write $\vec{F} = m_2 \cdot \vec{C}$.

The Scheme code that corresponds to the view taken in Equation (9) is given in Figure 6. Note that this is still a curried function, as the mass m_1 is needed before a field is created. However, when applied to a mass it no longer returns a curried function. The Scheme expression `(physics-gravity-law` m_1`)` gives the gravitational field of m_1 as measured in units of a newtons per kilogram vector. A physicist would write this as $g(\vec{r})$, letting m_1 be understood from context, and letting $\vec{r}$ implicitly range over all vectors, as in the following equation, where it is clear that a function of $\vec{r}$ is being defined.

$$g(\vec{r}) = \vec{C} = \frac{-Gm_1\vec{r}}{r^3} \tag{10}$$

```
(define physics-gravity-law   ; type:  [kg → [m* → (N*/kg)]]
   (lambda (m1)                  ; kg
      (lambda (r-vector)          ; m*
         (let ((r (distance r-vector)))
            (scalar-multiply
               (- (/ (* G m1)
                     (cube r)))
               r-vector)))))

(define cube
   (lambda (r)
      (* r (* r r))))
```

Figure 6: The physics view of Newton's law of universal gravitation. The auxiliary functions `distance` and `grav-force` are defined above.

It is interesting that a physicist would not assign units to g in the above equation. However, by calling g a "gravitational field" a physicist remembers that g is a function of type $[\mathrm{m}^* \rightarrow (\mathrm{N}^*/\mathrm{kg})]$.

As an example, the earth's gravitational field can be modeled by the function `physics-earths-field` declared in the following code. Unlike the function `earths-field` defined above, `physics-earths-field` assigns to each point around the earth a force per unit mass vector, not a function.

```
(define physics-earths-field   ; type:  [m* → (N*/kg)]
   (physics-gravity-law mass-of-earth))
```

Passing to `physics-earths-field` a radius vector representing some point on the earth's surface gives the field strength at the earth's surface. This is measured in units of force per unit mass.

```
(define field-strength-at-surface   ; type:  N*/kg
   (let ((point-on-earth (list 1.0 0.0 6.37e6)))     ; m*
      (physics-earths-field point-on-earth)))
```

Unlike the function `earths-field-at-surface` defined above, the force per unit mass vector `field-strength-at-surface` is not a function. However, the force per unit mass vector can be used to find the force vector on a given mass by scalar multiplication. For example, to find the force vector on a 68 kilogram mass at the given point, one would write the following in Scheme.

```
(scalar-multiply 68.0 field-strength-at-surface)
```

4.4 A Correspondence between Different Views of Vector Fields

When a physicist says "force per unit mass vector" a functional programmer may think "function from masses to force vectors." (The opposite transla-

```
(define gravity-law-c-2    ; type:  [kg → [m* → [kg → N*]]]
   (lambda (m1)               ; kg
      (lambda (r-vector)       ; m*
         (vector-per-unit->function
            ((physics-gravity-law m1) r-vector)))))

(define vector-per-unit->function
   (lambda (vec)
      (lambda (scalar)
         (scalar-multiply scalar vec))))
```

Figure 7: The curried function view of Newton's law of universal gravitation, coded using the physics view.

```
(define physics-gravity-law-2    ; type:  [kg → [m* → (N*/kg)]]
   (lambda (m1)                   ; kg
      (lambda (r-vector)           ; m*
         (function->vector-per-unit
            ((gravity-law-c m1) r-vector)))))

(define function->vector-per-unit
   (lambda (f)
      (f 1.0)))
```

Figure 8: The physics view of Newton's law of universal gravitation, coded using the curried function view.

tion also makes sense.) In general, a functional programmer may think of the phrase "per unit" as signaling the presence of a function. This works for other fields, such as the dielectric field, where the physicist says "force per unit charge vector" and the function programmer may think "function from charges to force vectors". This translation is not a vague way of thinking, but can be formalized in Scheme and mathematics as is done below.

The translation is expressed in Scheme by coding the curried function view of Newton's law of universal gravitation, `gravity-law-c`, using the physics view, `physics-gravity-law`, and vice versa. The first translation is shown in Figure 7, and the reverse is shown in Figure 8. Since these translations are built around the auxiliary functions `vector-per-unit->function` and `function->vector-per-unit`, they can be applied to any field.

Mathematically, let T and S be sets (of numbers with appropriate units). Assume that S has a distinguished element 1. The type T/S (of as numbers with units T/S) can be thought of as having elements of the form $t/1$, where t has type T (i.e., $t \in T$). Assume that there is an operator, $\cdot$, such that for all t of type T the following holds.

$$1 \cdot (t/1) = t \tag{11}$$

From this it follows that for all x of type T/S,

$$((1 \cdot x)/1) = x. \tag{12}$$

Let the type $E[S \rightarrow T]$ of *equivariant maps* from S to T be such that for all f of type $E[S \rightarrow T]$ and for all s of type S, the following holds [BW90, Definition 3.2.2].

$$f(s) = s \cdot ((f(1))/1) \tag{13}$$

Theorem 1 *There is an isomorphism (of sets) between the type T/S and the type $E[S \rightarrow T]$ of equivariant maps from S to T.*

Proof: The isomorphism is given by the function $I : T/S \rightarrow E[S \rightarrow T]$ that is defined such that for all x of type T/S and for all s of type S,

$$(I(x))(s) \overset{\text{def}}{=} s \cdot x. \tag{14}$$

That is, I is the function $\lambda x \,.\, \lambda s \,.\, s \cdot x$, which assigns to each x of type T/S the equivariant map, f_x, such that $f_x(s) = s \cdot x$. The inverse function I^{-1} is defined as follows.

$$I^{-1}(f) \overset{\text{def}}{=} ((f(1))/1) \tag{15}$$

That is, I^{-1} is the function $\lambda f \,.((f(1))/1)$, which assigns to each equivariant map f of type $E[S \rightarrow T]$ its value at 1 (in the appropriate units).

That I and I^{-1} are inverses is be shown as follows.

$$\begin{aligned}
I^{-1}(I(x)) &= I^{-1}(\lambda s \,.\, s \cdot x) && (16)\\
&= (((\lambda s \,.\, s \cdot x)(1))/1) && (17)\\
&= ((1 \cdot x)/1) && (18)\\
&= x && (19)\\
I(I^{-1}(f)) &= I((f(1))/1) && (20)\\
&= \lambda s \,.\, s \cdot ((f(1))/1) && (21)\\
&= \lambda s \,.\, f(s) && (22)\\
&= f && (23)
\end{aligned}$$

∎

The function I used in the above proof is `vector-per-unit->function` in Figure 7, and its inverse, I^{-1} is coded by `function->vector-per-unit` in Figure 8.

A physicist might object to these translations, saying that vectors support addition, scalar multiplication, etc. However, one can also use these translations to define such operations on equivariant maps. For example, one can define addition of equivariant maps f and g of type $E[S \rightarrow T]$ as follows, where the plus on the right hand side is vector addition.

$$(f + g) \overset{\text{def}}{=} I(I^{-1}(f) + I^{-1}(g)) \tag{24}$$

The translation into Scheme and the definition of scalar multiplication of functions are left to the reader.

These translations can also be lifted to functions, and hence both I and I^{-1} can be considered functors [Lan71] [Pie88] [BW90] between the category T/S (a sub-category of the category of sets), and the category $E[S \rightarrow T]$ of equivariant maps. Hence I is an isomorphism of these categories [BW90, Section 3.2], meaning that they are mathematically equivalent perspectives.

4.5 The Utility of Another View of Vector Fields

Why is another way to think about vector fields useful?

> I would answer, that it is a good thing to have two ways of looking at a subject, and to admit that there *are* two ways of looking at it. — James Clerk Maxwell [Max90, page 208]

One answer is that it is always interesting for teachers and students to connect disparate areas of knowledge. Students who understand curried functions are given a way to understand vector fields (and vice versa).

If physicists have not considered curried functions before (which seems plausible), then it may be that the curried function view will be of some conceptual advantage in Physics. However, any such advantage would be highly speculative. A more likely benefit for Physics would be the expansion of the vocabulary of "units" to include functions.

Another answer is that while the two views may be mathematically equivalent, it may be computationally advantageous to use one or the other. The advantage of the curried function point of view is that partial applications of curried functions may produce functions that are significantly faster than the uncurried function. If one is doing several computations about the gravitational field of some particular object (e.g., the earth, sun, or galaxy), then fixing these parameters by partial applications, such as `earths-field-at-surface` may result in faster code. For example, consider the rewrite of `gravity-law-c` in Figure 9. When the Scheme function call `(gravity-law-c-3` m_1`)` is evaluated, a negation and a multiplication are done. This work is saved if the function `(lambda (r-vector) ...)` that results from the call is bound to a name and used over and over (as in `earths-field`). Note that this work could not have been saved unless `gravity-law-c-3` were curried. Similarly, for a fixed `r-vector`, the considerable work in computing `r` and `force-per-unit-vector` is saved. (However, this work would also be saved using the physics view of the field, even without currying.)

5 Conclusion

Examples from Physics, such as the gravitational force example discussed above, are excellent ways to motivate students of functional programming. The example is fairly intuitive, connects with the real-world experience of students, and connects with other courses they may be taking or will take

```
(define gravity-law-c-3   ; type:  [kg → [m* → [kg → N*]]]
  (lambda (m1)                        ; kg
    (let ((negative-G-m1 (- (* G m1)))) ; N·m²/kg
      (lambda (r-vector)              ; m*
        (let ((r (distance r-vector))) ; m
          (let ((force-per-unit-vector ; N/kg
                  (scalar-multiply
                    (/ negative-G-m1
                       (cube r))
                    r-vector)))
            (lambda (m2) ; kg
              (scalar-multiply
                m2
                force-per-unit-vector)))))))
```

Figure 9: A potentially faster version of `gravity-law-c`, which does as much computation as possible upon receiving each actual parameter.

in Physics. The aura of Physics as being "natural" adds "naturality" to the concept of curried functions, which might otherwise seem highly "artificial". The context of some real-world application adds some complications, but the increased interest more than makes up for these complications. The increased interest comes from showing the utility of curried functions in a real-world example.

As a side benefit of understanding curried functions, functional programmers gain an additional perspective on Physics, in that they may use their understanding of curried functions to help them understand the concept of fields in Physics.

Acknowledgements

Thanks to John Clem for pointing out my misunderstandings of physics and where I needed more explanation. Thanks Dan Friedman for comments on drafts, and to Dan, Mitch Wand, and Phil Wadler for helping me track down people who might have thought of this before. Thanks to David A. Turner, Hal Abelson, and Gerry Sussman, for responding to that search. Thanks to Al Baker, Akhilesh Tyagi, David Fernandez-Baca, Phil Wadler, Yoonsik Cheon, K. Kishore Dhara, Marybeth Gurski, Ke Chen, Yanfang Li, and Mang-Mang Ling, Jack Lutz, and Tim Wahls for comments on drafts.

References

[Ari61] Aristotle. *Physics*. University of Nebraska Press, Lincoln, Nebraska, 1961.

[ASS85] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs.* The MIT Press, Cambridge, Mass., 1985.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland Publishing Co., New York, N.Y., 1984. Revised Edition.

[BW90] Michael Barr and Charles Wells. *Category Theory for Computing Science.* International Series in Computer Science. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1990. ISBN 0-13-120486-6.

[CFC58] Haskell B. Curry, Robert Feys, and William Craig. *Combinatory Logic.* Studies in logic and the foundations of mathematics. North-Holland Pub. Co., Amsterdam, 1958.

[Cur30] H. B. Curry. Grundlagen der kombinatorischen Logik. *Amer. J. Math.*, 52:509–536, 789–834, 1930.

[Dug55] Rene Dugas. *A History of Mechanics.* Editions du Griffon, Neuchatel, Switzerland, 1955. Translated by J. R. Maddox.

[Far52] Michael Faraday. On the Physical Lines of Magnetic Force. *Royal Institution Proceedings*, June 1852. Reprinted in volume 45 of the Great Books Series, Encyclopaedia Britannica, Inc., Chicago, Illinois, 1952.

[FLS63] Richard P. Feynman, Robert B. Leighton, and Matthew Sands. *The Feynman Lectures on Physics*, volume I. Addison-Wesley, Reading, Massachusetts, 1963.

[Fre91] Gottlob Frege. *Collected Papers*, chapter Function and Concept, pages 137–156. Basil Blackwell, Jena, 1984 edition, 1891. Translated by Peter Geach, edited by Brian McGuinness.

[IEE91] IEEE Std 1178-1990. *IEEE Standard for the Scheme Programming Language.* IEEE, New York, N.Y., 1991.

[Lan71] Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, N.Y., 1971.

[Max64] James Clerk Maxwell. A Dynamical Theory of the Electromagnetic Field. *Royal Society Transactions*, CLV, 1864. Reprinted in R. A. R. Tricker, *The Contributions of Faraday and Maxwell to Electrical Science*, (Pergamon Press, 1966).

[Max90] James Clerk Maxwell. *The Scientific Papers of James Clerk Maxwell.* Cambridge, Cambridge, UK, 1890. As quoted in W. Berkson, *Fields of Force: The Development of a World View from Faraday to Einstein* (Wiley, 1974).

[New87] Isaac Newton. *Mathematical Principles of Natural Philosophy*, volume 34 of *Great Books*. Encyclopaedia Britannica, Inc., Chicago, Illinois, 1687. Translated by Andrew Motte, Revised by Florian Cajori.

[Pie88] Benjamin C. Pierce. A Taste of Category Theory for Computer Scientists. Technical Report CMU-CS-88-203, Computer Science Dept, Carnegie Mellon University, Pittsburgh, 1988.

[RCA$^+$86] Jonathan Rees, William Clinger, H. Abelson, N. I. Adams IV, D. H. Bartley, G. Brooks, R. K. Dybvig, D. P. Friedman, R. Halstead, C. Hanson, C. T. Haynes, E. Kohlbecker, D. Oxley, K. M. Pitman, G. J. Rozas, G. J. Sussman, and M. Wand. Revised^3 Report on the Algorithmic Language Scheme. *ACM SIGPLAN Notices*, 21(12):37–79, December 1986.

[Sch24] Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Math. Annalen*, 92:305–316, 1924. An English translation appears in *From Frege to Godel*, edited by Jean van Heijenoort (Harvard Univ. Press, 1967), pages 355-366.

[SF89] George Springer and Daniel P. Friedman. *Scheme and the Art of Programming*. McGraw-Hill, New York, N.Y., 1989.

[SS78] Guy Lewis Steele Jr. and Gerald Jay Sussman. Revised Report on SCHEME A Dialect of LISP. AI Memo 452, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, January 1978.

[Tri66] R. A. R. Tricker. *The Contributions of Faraday and Maxwell to Electrical Science*. Pergamon Press, Oxford, 1966.