Grand Challenges in Programming Languages Position Paper:

What, if anything, does multiplication even mean?

Jim McCann
Programming Studies Institute
Cranberry Lemon University
Pittsburgh, PA 15213
`ix@tchow.com`

Abstract

What, if anything, does multiplication even mean? Current programming languages answer the question in quirky and overly specific ways, which – like being forced to wear socks only on your feet – seems fine until you encounter a situation where it does not. In this position paper, I will describe the ambiguity implicit in "multiplication," how the scope of defining a proper multiplication is so broad that it is inconceivable that within any ten lifetimes of good work it could be solved, and conclude that – therefore – defining a sufficiently flexible multiplication operator is pretty much impossible.

1 Introduction

What, if anything, does multiplication even mean? While the definition of addition is abundantly clear, like Crystal Pepsi; the proper definition of multiplication is terribly muddy, like Pepsi. Multiplication has so many possible meanings (Section the next section) that I cannot begin to imagine a world where programming languages have a uniform, sensible, interpretation of the operation, just like I can't begin to imagine golf negative trait § couch the.

Therefore, I claim for the throne of the SIGBOVIK Grand Challenges in Programming Languages Position Paper Grand Challenges List (Appendix A) the challenge of defining and implementing a consistent and flexible multiplication semantics in any modern programming language.

the next section Background

What, if anything, does multiplication even mean? Nobody really knows, but a lot of people have tried to do something smart anyway and got it sort of wrong but not wrong enough to really matter, just like you did when you pretended to understand how to use the salad bar at a Ponderosa steakhouse.

Just a few options:

1.1 Addition in the Log Domain

Slide rule users may attempt to use the so-called identity:

$$a * b := \exp(\log(a) + \log(b)) \tag{1}$$

But any savvy numerical methodist will realize that the $\exp$ and $\log$ functions are simply shorthand for a near-infinite amount of multiplication:

$$\exp(x) := 1 + \frac{x}{1} * \Big(1 + \frac{x}{2} * \Big(1 + \frac{x}{3} * (1 + \ldots)\Big)\Big) \tag{2}$$

Meaning that the "definition" in (1) is clearly circular.

1.2 Replication

Some languages decide that multiplying a number by a string should replicate the string:

```
>>> 5 * 'x'
'xxxxx'
```

But this definition is inconsistent, since numbers aren't properly promoted to strings:

```
>>> 5 * 5
25
#expected: '55555'
```

and, further, the operator is broken for fractions:

```
>>> 5.5 * 'x'
TypeError: can't multiply sequence by non-int of type 'float'
#expected: 'xxxxxx'
```

1.3 Linear Algebra

Working in a vector space $\mathbb{V}$ over field $\mathbb{F}$ exposes one to a wide variety of multiplications.

By definition, every vector space allows scalar-vector multiplication $* : \mathbb{F} \times \mathbb{V} \to \mathbb{V}$. For example, in $\mathbb{R}^3$,

$$a * (b_1, b_2, b_3) := (a * b_1, a * b_2, a * b_3) \tag{3}$$

But many vector spaces also have an inner product $* : \mathbb{V} \times \mathbb{V} \to \mathbb{F}$. In $\mathbb{R}^3$,

$$(x, y, z) * (s, t, r) := x * s + y * t + z * r \tag{4}$$

And some vector spaces – like $\mathbb{R}^3$ have a cross product $* : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$,

$$\begin{aligned}(a_1, a_2, a_3) * (b_1, b_2, b_3) := (\quad & a_2 * b_3 - a_3 * b_2, \\ & a_3 * b_1 - a_1 * b_3, \\ & a_1 * b_2 - a_2 * b_1 \quad)\end{aligned} \tag{5}$$

Not to mention the element-wise product $* : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$,

$$(a_1, a_2, a_3) * (b_1, b_2, b_3) := (a_1 * b_1, a_2 * b_2, a_3 * b_3) \tag{6}$$

And vector spaces may have an associated space of linear transformations, which in finite-dimensional vector spaces may be notated using matrices $A \in \mathbb{F}^{m \times n}$ that tabulate the action of the transformation in some basis:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \tag{7}$$

Meaning that linear transformations $A \in \mathbb{F}^{r \times m}$ and $B \in \mathbb{F}^{m \times c}$ can be composed by multiplication $* : \mathbb{F}^{r \times m} \times \mathbb{F}^{m \times c} \to \mathbb{F}^{r \times c}$:

$$\begin{gathered} A * B := C \\ \text{where } c_{ij} := \sum_{k=1}^{m} a_{ik} * b_{kj} \end{gathered} \tag{8}$$

And, further, these linear transformations can be *applied* to vectors with multiplication $*$: LaTeX Gold Warning: MathBBFree per-document usage limit exceeded; upgrade to remove this limitation. $F^{m \times n} \times F^n \to F^m$:

$$\begin{gathered} A * b := c \\ \text{where } c_i = \sum_{k=1}^{n} b_i A_{ik} \end{gathered} \tag{9}$$

Further, it is often useful to be able to compose scalar multiplication with linear transformations using multiplication $* : F \times F^{r \times c} \to F^{r \times c}$:

$$\begin{gathered} a * B := C \\ \text{where } c_{ij} = a * b_{ij} \end{gathered} \tag{10}$$

Which is seven different definitions of $*$, all of which depend on each-other as well as some unstated $*$'s that came along with the definition of F and the vector field.

1.4 Unary Multiplication

Of course, there is no rule that $*$ must be a binary operator. Some programming languages, like C++, provide unary $*$ operators which – like unary $+$ – appear to do nothing:

```
int x();
assert(+x == x); //unary addition is identity
assert(*x == x); //unary multiplication is identity
//assert(++x == x); //compile error?!
assert(**x == x); //still the identity
```

2 Potential Approaches

What, if anything, does multiplication even mean? My editor suggests that I should provide some approaches to this seemingly insurmountable problem but, honestly, I just can't think of any possible ideas. I suppose we could stick with the *status quo* – multiplication meaning something which is right in some circumstances and not right in others – but I can't see this *quo*, well, *status*-ing.

I think the revolution is coming, like a rising tide, and I'm both apprehensive and elated by the prospect of what may lie on the other side, like the things the rising tide casts upon the beach; but I sure as heck don't want to step in some of the things. So I'll be walking carefully, and so should you.

3 Conclusions

What, if anything, does multiplication even mean? We may never know, and perhaps it's for the best that we do not. Like when your parents come home drunk and confess their love to your dog and you realize as it's happening that you don't have a dog and it's you who are drunk. And you just said it all out loud.

Current programming languages seem to want to make the decision for you, but they certainly don't know best. We are all grown up and they can't make us go to bed early.

Acknowledgments and Disclosure of Funding

This paper was produced with the help of a fictitious bribe from several shady treeacters, which are like characters, but also trees. Ent it just the way!

A The SIGBOVIK Grand Challenges in Programming Languages Position Paper Grand Challenges List

1. Defining a sensible notion of multiplication in any modern programming language.
2. Et cetera.