P Computation

Robert S VanDerzee
Department of Computer Science
the University of Virginia

February 1, 2021

Contents

Preface

Acknowledgments

This is motivated by a question posed in a Compilers course at the University of Virginia, taught by Professor Dwyer, Ph. D.

> How do we unify the '+' operator to accommodate both integer and floating point operands?

Which deals with the issue of sub-typing and type inference within language compilers. After reading sections of [Mitchell 1984], I was inspired to determine a method to make type inferring a decidable problem with pure computational type unification.

> "The path of least resistance changes with knowledge."

Robert S VanDerzee
October 2020

1 Introduction

Robbie VanDerzee

Under current computational models, we have found and proven there are many undecidable problems. The context of these undecidable problems strictly takes the form of problem representation. It is known that certain forms of problem representation effect characteristics of the solutions of the problems — even in general. Under those constraints and computational assumptions, we can make determinations about these various solution characteristics: elements like decidability, complexity, consistency, completeness, and other more specified forms of solution characteristics[1].

1.1 Conventional representation

One of the most fundamental assumptions of program and data representation within these problems is numerical representation. Colloquial terms of numerical representation are base number systems, which reflect coefficient multiplications of powers of a base. These representations can be mathematically described as seen in Eq. (1.1).

$$v = \sum_{n=0}^{k} a_n b^n = a_0 b^0 + a_1 b^1 + ... + a_n b^n + ... + a_k b^k \tag{1.1}$$

Where we define $0 \leq a_n \leq 1$ to enforce the uniqueness of the representations. In modern electrical-based computation systems b is 2 for data validity and processing clarity[2]. However, all classes of b and associated a are equivalent representation forms and translation is generally defined and trivial to compute.

1.2 P computation representation

Suppose we based our computational models in a fundamentally alternative form. We want this form to unify all numeric types in an easily extendable unit. Consider an alternative representation of Numeric types using the prime factorization notation Eq. (1.2).

$$v = integer\langle k \rangle = \{\ p_0,\ p_1,\ p_2,\ ...,\ p_k\ \} \tag{1.2}$$

[1] Like sorting stability.

[2] Both of these are practical reasons, other base computers are possible.

We will begin with a definition of integer classes, and demonstrate a method to extend this class as far as necessary, in a manner conducive to type-less desired nary operation[3].

This starting equation is axiomatic, such that we must assume the principle of counting in order to define the integer class. Accepting this assumption allows the extension we desire. We will also define 0 in this system to be the empty list. Definitions for these numerics are non-unique, but we will choose the shortest representation when comparing. Otherwise, we define these types by their lengths, indicated by k.

For clarity, let's instantiate this construction for the traditional counting integers. We will consider the $\min(k)$ representations in Tab. (1.1).

Table 1.1 $\mathbb{Z}^+$ conventional representation relative to P_{comp} proposal.

$\mathbb{Z}^+(v)$	$P_{comp}(v)$
0	$\emptyset$
1	$\{0\}$
2	$\{1\}$
3	$\{0,1\}$
4	$\{2\}$
45	$\{0,2,1\}$

Integer type bi-directionality is not enforced in this representation, that is, negative and positive integers only differ by symbolic interpretation. So the concept is not distinguished other than how the numbers are used, and is purely equivalent.

This representation serves as the foundation of all other elements included in this computational model and it's respective **nary** operators, the **P programming language**, and the **P compiler**.

[3] An **nary** operator is the class of operator functions with output determined by n ordered arguments.

2 Extending P computation

We would like to extend this representation to the different traditional types, namely the Integer ($\mathbb{Z}$), Rational ($\mathbb{Q}$), Real ($\mathbb{R}$), and Complex Numbers ($\mathbb{C}$), and then demonstrate that we can easily unionize these types such that nary operators can operate without type checking.

It is important to note that we have made the arbitrary decision to represent the count indices with base 10. There are numerous methods to convert this into a safer (less bit error) representation, but there is also the recursive extension. The recursive extension only allows for the symbols $\in \{0, 1\}$. Operands will increase the dimension of the index list to enforce this property; a model which we will convert to in implementation.

2.1 Integer type directionality

It is of immediate interest how we extend this representation to succinctly include the mirror integers (conventionally negative integers). There are functional properties of computation where, though these representations are symmetric, the functional results are highly asymmetric. This arises from the nature of rotation, especially in the case of complex fields. One beautiful example of this is the Riemann Zeta Function along the critical line (2.1).

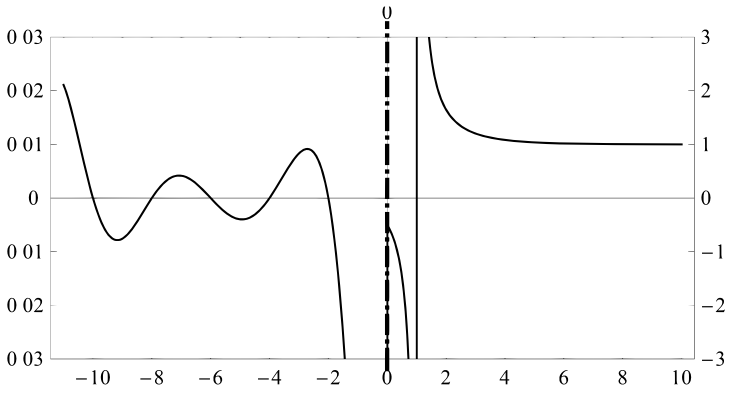

Figure 2.1 Riemann zeta function along the critical line, 1/2.

Traditional representations use the '-' negation symbol prepended to the Integer string. Likewise, computational representations of this are preformed in binary with

two's complement form (and others) in order to unionize operators between the types. We will do nothing to immediately represent this directionality. Instead, we will consider this directionality a subset of Complex numerics, which we can easily extend our representation to. This is because directionality is effectively rotation. If we encompass rotation in the general form, we do not have issue representing inverted numeric behavior.

To accomplish this, we will add an additional characteristic as shown in Eq. (2.1), which sets the first index to be the power of -1, which we will now call the -1^{st} prime number:

$$v = \pm\ integer\langle k\rangle = \{0/1 \in p_{-1},\ p_0,\ p_1,\ p_2,\ ...,\ p_k\ \} \tag{2.1}$$

We will no longer signify the distinctions between $\pm$ *integer* and *integer*. These types are now unified.

This property will work such that even representations of this index are positive, and odd are negative. The interval between (which will be encompassed by the representation of rational types) will be complex rotations.[1]

2.2 Rational types

A rational type is expressed mathematically in reduced form as in Eq. (2.2).

$$v = \frac{p}{q} \ni (p, q) \in \mathbb{Z} \wedge gcd(p, q) \equiv 1 \tag{2.2}$$

such that $gcd(p, q)$ is defined to be 1 when p and q share no factors. Our representation already enforces this notion, all we must do is extend the directionality of the power. We can extend our P_{comp} type as follows in Eq. (2.3).

$$v = rational\langle k\rangle = \{\pm\ p_{-1},\ \pm\ p_0,\ \pm\ p_1,\ \pm\ p_2,\ ...,\ \pm\ p_k\ \} \tag{2.3}$$

Note that this directionality representation can be absorbed from the previous definition, if needbe. Furthermore, there is no strict limitation on the rotational argument p_{-1}, as this does not effect desired behavior, but rather stores additional information about **nary** operations preformed to obtain the $rational\langle k\rangle$, which we denote with v. However, the underlying representations of p_n where $-1 \leq n \leq k \in \mathbb{Z}$ are also $\in \mathbb{Z}$. We will continue this process to escape that restriction. We can see how this definition behaves for some rational types in Tab. (2.1).

[1] Another useful property of this representation is that it will easily allow for operation tracking; allowing computation to exist as a function of time, rather than a static state.

Table 2.1 Z conventional rational representation relative to P_{comp} proposal.

$Z(v)$	$P_{comp}(v)$
$1/2$	$\{0,-1\}$
$1/3$	$\{0,0,-1\}$
$1/4$	$\{0,-2,0\}$
$1/5$	$\{0,0,0,-1\}$
$5/7$	$\{0,0,0,1,-1\}$
$-5/7$	$\{1,0,0,1,-1\}$
$11/5$	$\{0,0,0,-1,0,1\}$

We will no longer signify the distinctions between *int* and *rational*. These types are now unified, and we will call this unification *rational*.

2.3 Irrational and Complex types

Irrational types are easily extended from the definition followed in Sec. (2.2). Simply, we allow the recursive extension. That is, each index is now defined with a rational, instead of the previously declared count variable p_n. Using this, we have simultaneously unified complex types, as rational exponents of -1 indicate rotations. Mathematically speaking, we do not care of the specification of k, we allow k to be infinite. In type implementations, however, we will specify k because memory is not unlimited, and useful information can be gleaned from k, even though it has no effect on v for v which have maximum prime k. A formal specification of the new type is defined in Eq. (2.4), for which we let $rational\langle k\rangle = r$.

$$v = complex\langle k\rangle = \{\ r_{-1},\ r_0,\ r_1,\ r_2,\ ...,\ r_k\ \} \tag{2.4}$$

Again, we can see this by instantiating some translations between $\mathbb{C}$ and our type, which we will now denote with $\mathbb{P}$, in Fig. (2.2)

Table 2.2 $\mathbb{C}$ conventional rational representation relative to $\mathbb{P}$ proposal.

$\mathbb{C}$	$\mathbb{P}$
$\sqrt{2}$	$\{\{0\},\{0,-1\}\}$
$\sqrt{6}$	$\{\{0\},\{0,-1\},\{0,0,-1\}\}$
$\sqrt[3]{7.5}$	$\{\{0\},\{1,0,-1\},\{0,0,-1\}\,\{0,0,-1\}\}$
i	$\{\{1\}\}$
$\sqrt{-6}$	$\{\{1\},\{0,-1\},\{0,0,-1\}\}$
$\sqrt[\sqrt{2}]{2}$	$\{\{0\},\{0,\{0,-1\}\}\}$

[2]

These types are now unified, and we shall call this new construction of $\mathbb{P}$, $\mathbb{P}_3$, indicating a 3 depth $\mathbb{P}$ construction. We must address the fact that we have yet defined the operators sufficient to complete the translation. Interestingly, we will be able to unify $\mathbb{C} = a + bi \ni (a, b) \in \mathbb{R}$ into a singular $\mathbb{P}_3$ number. But first, we will investigate higher order types, and the representation ranges of $\mathbb{P}$; of which do not conventionally have translations, but may provide use in $\mathbb{P}$ computers.

2.4 Higher order types

What does $\mathbb{P}_4$ relate to, then? Our class specifications allow for the above recursion to repeat infinitely, such that all irrational and complex numbers are expressed. We can visualize what $\mathbb{P}_4$ and higher do as vector space. If we accept the notion that $\mathbb{P}_3$ represents a number in 2 dimensional space, we can reason that a $\mathbb{P}_4$ number may only exist in a 3 dimensional space.[3] Geometrically, in $\mathbb{P}_4$ we allow for two rotations to happen, which are perpendicular axis. This is a 3 dimensional sphere volume, with radius specified by the magnitude of the number $v \in \mathbb{P}_4$.

Recursive extensions of $\mathbb{P}_3$, $\mathbb{P}_4$, and others do not strictly increase the volumetric space which the number has access to. It only occurs when the elements themselves have rotations applied to them. This can occur for any c_i in $v \in \mathbb{P}_d$, where i is the i_{th} index in v and d is the label we apply to the dimensionality. We can count these rotations to determine which dimension plane v exists in. No matter the extension, v is still type unified.

Furthermore, we can use this definition to **limit** our symbol quantity without effecting the dimensionality of v. Further discussion of this matter will be explored in the computational logic stage of our extension.

We will now call this globally unified type the P desic. The term stems from geodesic, but the geometric comes from the field of primes, hence P desic. We will later prove that this mathematical construction, the P desic, contains all possible previous representations of numbers, and vector numbers, with no issue. If we so choose, we can name this type *number* in other contexts, due to this equivalence. We will refrain from doing so until this truth is verified.

2.5 Representation ranges

2.6 Nary operators

2.7 Computation logic

[2] We illustrate these as sets of order lists, but this representation is not final as the recursive nature of our definition would be better served with a tree.

[3] The nuance here is that some numbers in 3 dimensional space actually exist on the 2 dimensional plane.

3 P functions

4 P programming language

We can construct a program language with these type and logic in mind.

5 P compiler

6 P algorithmic analyses

A Appendix

A.1 Physical Implementation Architectures

Bibliography

J. C. Mitchell. 1984. Coercion and type inference. *Proc. 11th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages*, POPL-84(11): 175–185.