PURE DATA SPACES....DRAFT...getting there...

SAUL YOUSSEF
Department of Physics
Boston University

June 21, 2025

Abstract

All about spaces.

1 Overview

....WRITE INTRODUCTION...

- Conceptual overview: small(est?) axiomatic system analogous to ZFC, where proof and computation are the same thing.
- Point of the paper is to investigate the concept of a "Space" introduced in ref. 1. This is analogous to "Set" or "Category."
- Theme is to "grow the simplest spaces organically" and compare the simplest results with classical mathematics.

2 Foundation

In [1], we argue that Mathematics and Computing as a whole can be captured within a small axiomatic framework, based on **finite sequence** as the foundational concept. Assuming that finite sequences are understood, *data* and *coda* are defined by

Definition 1. Data is a finite sequence of **codas**, where each **coda** is a pair of **data**.

The two foundational operations defined on data are 1) concatenation of data A and data B, written '$A\ B$', and 2) pairing of data A and data B as a coda, written with a colon '$A : B$'. By Definition 1, the empty sequence, written '()', qualifies as data and, therefore, () paired with itself is a coda, written '(():())' or '(:)'. Any finite sequence of codas is data, so, for example, (:) (:(:)) ((:):(:(:))) is data consisting of a sequence of three codas. We can think of this as *pure data* since it is "data made of nothing." By convention, the colon binds from the right first and binds less strongly than concatenation, so that $A : B : C$ is defined to be $(A : (B : C))$ and $A : B\ C$ is defined to be $(A : (B\ C))$. Data is typically written with upper case, and codas are typically written in lower case. To indicate the left and right data of a coda, we sometimes use L/R superscripts so, for any coda c, $c = (c^L : c^R)$.

All meaning within the system is determined by a chosen partial function from coda to data called a *context*. Contexts are partially ordered by inclusion, so if δ and δ' are contexts, $\delta \leq \delta'$ if δ

and δ' are equal on the domain of δ. Given a context δ, equality of data is the equivalence generated by $c \sim \delta(c)$ for any coda c, and by compatibility with concatenation and colon in the sense that if $A \sim B$, then $(A\ X) \sim (B\ X)$, $(X\ A) \sim (X\ B)$, $(A : X) \sim (B : X)$ and $(X : A) \sim (X : B)$ for any data X. This relation $A \sim B$ is denoted $A \overset{\delta}{=} B$, or simply $A = B$ when the context is unambiguous. It follows that if $\delta \leq \delta'$, then $A \overset{\delta}{=} B$ implies $A \overset{\delta'}{=} B$. We say that if A and B are equal then they are "always equal," thinking of moving from δ to δ' as the passage of "time." The fixed points of a context δ are *atoms*. Note that if c is an atom in context δ, and if $\delta \leq \delta'$, then c is still an atom a context in δ'. Thus, atoms are "permanent" and if c is an atom, c is "always" an atom.

Definition 2. Within a given context δ, coda c is an **atom** if $\delta : c \mapsto c$. If data A contains an atom in its sequence, A is **atomic** data. Data A is **invariant** if every coda a in the sequence of A is an atom and if a^L and a^R are both **invariant**.

It follows from the definition that empty data is invariant and atoms a and b are equal if and only if $a^L = b^L$ and $a^R = b^R$. If a and b are atoms, A and B are data, then $(a\ A) = (b\ B)$ and $(A\ a) = (B\ b)$, if and only if $a = b$ and $A = B$. If $A = B$ and A and B are invariant, then A and B are identical as pure data [proof].

3 Genesis

In the proposed framework, all mathematical objects are pure data and all meaning is defined by a chosen context δ and its corresponding equivalence. A **proof** in δ, for example, is just a sequence $A_1 \overset{\delta}{=} A_2 \overset{\delta}{=} \ldots \overset{\delta}{=} A_n$ which can be viewed either as proving A_n given A_1 or as a computation $A_1 \mapsto A_n$. The immediate issue is how to choose δ.

In the beginning, there are no definitions, and the corresponding context is the empty partial function δ_0. Since the domain of δ_0 is empty, it has no fixed points and, therefore no atoms. Thus, the empty sequence is the only invariant, $X \overset{\delta}{=} X$ is the only valid proof, and the only valid computations do nothing $X \mapsto X$ for any data X. To define a non-empty context $\delta_0 \leq \delta$, we need, at least, an invariant specification of the domain of δ as a partial function. Since the empty sequence is the only invariant, the only way to specify if a coda (A:B) is to be in the domain of δ is to require either A or B or both to be the empty sequence. In each of these cases, the coda (:) is within the domain of δ, and so we must, in any case, decide on what (:) maps to. There are three possibilities;

choice 1: $\delta : (:) \mapsto ()$,

choice 2: $\delta : (:) \mapsto (:)$, or

choice 3: $\delta : (:) \mapsto$ *anything other than* $()$ *or* $(:)$.

Since pure data is "made of (:)", choice 1 trivially causes all data to be equal to the empty sequence. On the other hand, choice 3 would mean that for any non-empty data A, the number of (:) atoms in A grows without limit. This would mean, for instance, that nocomputation could produce a final result. Thus, we are constrained to choice 2, where (:) is an invariant atom in δ, and, therefore an atom in any context $\delta \leq \delta'$ as well. It is convenient to generalize choice 2 and to let $\delta{:}(:X) \mapsto (:X)$ for all data X, since this provides a supply of invariant atoms (:), (:(:)), (:(:) (:)),... which can be used to expand the domain of δ.

We adopt a convention for expanding the domain of δ while preserving context partial ordering. A context of the form

$$\delta_a : (a\ A : B) \mapsto \delta_a(A, B) \tag{1}$$

for some invariant atom a is called a **definition**. We conventionally restrict ourselves to contexts $\delta \cup \delta_a \cup \delta_b \cup \ldots$ where a is an invariant atom in δ, b is an invariant atom in $\delta \cup \delta_a$, c is an invariant atom in $\delta \cup \delta_a \cup \delta_b$ and so forth. By requiring a, b,... to be disjoint, we maintain the context partial ordering.

4 Definitions

Before proceeding to mathematical exploration, we need to add enough expressive power to the original context δ. To do this, we introduce minimal definitions using the mechanism explained in the previous section.

1. Define "bits", "bytes" and "byte sequences" so that words are single atoms, so that a can be a word in future definitions δ_a.
2. Define all of the naturally occurring combinatoric operations given two finite sequences A and B. For example δ_{ap} maps (ap A : b_1 b_2 ... b_n) to (A:b_1) (A:b_2)... (A:b_n), so that ap "applies A to each atom of input."
3. Define a minimal internal language $\delta_{\{\}}$ so that a coda ({`language expression`} A : B) is mapped to data as described by the expression and possibly using data A and data B.
4. Include definitions giving direct access to context-level values. This include a Boolean definition δ_{bool} to determine if data is empty, $\delta_=$ to test if two data are equal in δ, and δ_{def} to add definitions via Equation 1.

These are meant to be a minimal collection of definitions, including only the minimal inevitable finite sequence concepts and, more generally, the combinatorics of Equation X. There are intentionally no definitions which presume underlying arithmetical operations. A natural number, '123', for instance is a single atom byte sequence and there is no definition which implies interpreting this as a number in the usual way. The idea is to allow natural numbers and whatever else to emerge "organically." There are approximately fifty definitions needed to create a basic system system. We define some here for orientation. More details can be found in the Glossary.

4.1 Bits, Bytes and Byte sequences

For convenience, we first define atomic *bits*, *bytes*, and *byte sequences*, so that we can make more readable definitions. Define

- $\delta_{(:)}$: single bit atoms, so that ((:):), ((:):(:)) are the zero bit and the one bit respectively.
- $\delta_{((:):)}$: atoms containing eight bits.
- $\delta_{((:):(:))}$: atoms containing arbitrary length byte sequences.

so, for instance, text strings are atomic data within $\delta \cup \delta_{(:)} \cup \delta_{((:):)} \cup \delta_{((:):(:))}$.

4.2 Combinatorics

Given that text strings are now invariant atoms, we can add definitions with readable names. Define the identity 'pass' so that (pass : X=X) for all data X.

- δ_{pass} : (pass A:B) $\mapsto$ B

and define 'null' so that (null : X=()) for all X.

- δ_{null} : (null A:B) $\mapsto$ ()

It is convenient to define "getting the A and B components of a coda" like so

- δ_{left} : (left A:B) $\mapsto$ A
- δ_{right} : (right A:B) $\mapsto$ B
- δ_{put} : (put A:B) $\mapsto$ (A:B)
- δ_{get} : (get : (:B)) $\mapsto$ B FIX ME

We proceed to define the minimal combinatorics involving of A and B as finite sequences. These don't have commonly understood names, so we are forced to invent new names. For example the name 'ap' is meant to suggest the concept "apply A to each atom of B" is expressed by the definition.

- δ_{ap} : (ap A : b B) $\mapsto$ (A:b) (ap A : B)
- δ_{ap} : (ap A : ()) $\mapsto$ ()

The domain of δ_{ap} is defined with the assumption that b is an atom, so the domain of the first branch of δ_{ap} is exactly codas (ap A, B) where B starts with an atom. By definition, if we list multiple "branches" as in this case, the first branch in the domain applies.

4.3 Language

Axiomatic systems like ZFC[ref] or dependent type theories[ref] are formal languages. There is an alphabet, special symbols, and syntax rules for defining valid expressions. Our approach avoids this by defining a language as just one more definition like any other. The basic idea is to define minimalist language, mainly giving access to the two foundational operations via text blank space (for concatentation) and text colon (for pairing to data as a coda). So, the idea is

- $\delta_{\{\}}$: ({x y} A : B) $\mapsto$ ({x} A:B) ({y} A:B)
- $\delta_{\{\}}$: ({x:y} A : B) $\mapsto$ ({x} A:B) : ({y} A:B)

where x and y are byte sequences as defined above. As written, this is ambiguous since x and y may contain space and colon characters, but these ambiguities can be resolved just by choosing and order, thus forcing the language into one definition $\delta_{\{\}}$. This definition includes

- $\delta_{\{\}}$: ({A} A : B) $\mapsto$ A
- $\delta_{\{\}}$: ({B} A : B) $\mapsto$ B

so that A and B have special meaning in the language. As a result, for example, ({B}:1 2 3)=1 2 3, ({B B}:1 2 3)=1 2 3 1 2 3, and ({A B} a b : 1 2)=a b 1 2.

Given a language expression in the form of byte sequence data L, the corresponding data is, simply, (L:). Every sequence of bytes is a valid language expression, so there is actually no need to define or check for syntax errors. Similarly, an *evaluation* of (L:) is merely some sequence (L:)=D_1=D_2=...=D_n producing an "answer" D_n. Typically, this is done with a simple strategy based on applying δ whenever possible or up to time or space limits. This also means that there is also no such thing as a "run time error."

4.4 System

The most basic question to ask about data is whether it is empty or not. This is captured by the definition δ_{bool}, defined so that (bool:B) is () if B is empty ("true") and (bool:B)=(:) if B is atomic ("false").

- δ_{bool} : (bool A : B) $\mapsto$ () if B is empty, (:) if B is atomic.
- δ_{not} : (not A : B) $\mapsto$ (:) if B is empty, () if B is atomic.

The equality defined by the context δ is available within δ via the following definition:

- $\delta_{=}$: (= A : ()) $\mapsto$ A
- $\delta_{=}$: (= (): A) $\mapsto$ A

and if a and b are atoms,

- $\delta_{=}$: (= a A : b B) $\mapsto$ (= a:b) (=A:B)
- $\delta_{=}$: (= A a : B b) $\mapsto$ (=A:B) (= a:b)

so, if $(A : B)$ is empty, A and B are "always" equal and if $(A : B)$ is atomic, A and B are "never" equal. The full language has a little bit of syntactic sugar, so one can write (A=B) instead of (= A:B) [ref].

New definitions can also be added to context via

- δ_{def} : (def `name` A : B $\mapsto$ ())

which is defined to be in domain as a partial function if `name` is not already in the current context, and, in that case, it adds the following definition to context.

- $\delta_{\texttt{name}}$:(`name` A':B') $\mapsto$ (B A':B')

so, for instance (def first2 : first 2 : B), means that (first2 : a b c d) is interpreted as ({first 2:B} : a b c d) which is equal to the data (a b).

We take this as the "organic" base of naturally occurring definitions plus the language will be a starting point in searching for the "spaces" defined in section 5. Further definitions used in the text can be found in the Glossary. Examples, tutorials, a complete definition of the language, and software can be found in reference [x].

5 Global Structure

Pure data has a global structure in the sense that the foundational operations $(A\ B)$ and $(A : B)$ are defined for any data A, B. This suggests defining a corresponding associative product $(A \cdot B)$ and associative sum $(A + B)$ by

$$(A \cdot B) : X = A : B : X \tag{2}$$

$$(A + B) : X = (A : X)\ (B : X) \tag{3}$$

for all data X. Since $(\text{pass} \cdot A)=(A \cdot \text{pass})=A$ and $(A + \text{null})=(\text{null} + A)=A$ for any A, pure data is a *semiring* with units pass and null. Following standard terminology, we say that A is *idempotent* if $A \cdot A=A$, is an *involution* if $A \cdot A=1$, and A has an *inverse* if $A \cdot A'=A' \cdot A=1$ for some data A'. In the case of a *product* $A=A_n \cdot \cdots \cdot A_1$, we say that A *starts with* A_1 and *ends with* A_n.

Data A is *commutative* if $A : X\ Y{=}A : Y\ X$ for all X, Y, and is *distributive* if $A : X\ Y{=}(A : X)\ (A : Y)$ for all X and for all Y.

Any data A can be viewed as defining a binary operation on data defined by $(X \overset{A}{*} Y) = (A : X\ Y)$. Since $\overset{A}{*}$ is associative if and only if (A:X Y)=(A:(A:X) Y)=(A:X (A:Y)) for all X,Y we say that data A is *associative* if it has this property.

If we say that data $A \leq B$ if $A \cdot B = A$, this defines a global preorder structure on data where $\leq$ is transitive in general and reflexive on idempotents. For example, null$\leq X \leq$pass for any data X. The equivalence $A \leq B$ and $B \leq A$ is indicated by saying that A and B are *compatible* data.

6 Spaces

A very basic requirement for useful mathematics is to have a way to define and refer to collections. In classical mathematics, this need is met by the concept of a Set. In the framework of pure data, every mathematical object must be identified with some data, and this must include any sort of collection of mathematical objects. Thus, we are lead to ask, given some data S, how could S represent a collection of other data? There are a couple of plausible ways to do this.

1. The collection corresponding to S would be the data $(S{:}X)$ for any data X.

2. The collection corresponding to S would be all of the fixed points of S.

These two ideas coincide if we require S to be idempotent, so that $(S : X)$ is always a fixed point. It is natural to also expect compatibility with sequences in the sense that if $(S : X)$ is in the collection and $(S : Y)$ is in the collection, then $(S : X)\ (S : Y)$ is also in the collection. This is guaranteed if we require

$$(S : X\ Y) = S : (S : X)\ (S : Y) \tag{4}$$

for all data X, Y. These requirements are neatly satisfied if S is associative, since associative data is idempotent and also always satisfies Equation 4. Thus, we are lead to a simple definition.

Definition 3. Data S is a **space** if S is associative.

If S is a space, we say that any data $(S{:}X)$ is *in* S. The data $(S{:})$ is called the *neutral data* of S and a space where $(S{:}){=}()$ is called a *neutral* space. It is easy to verify that if spaces S and T commute, then $S \cdot T$ is also a space and from the preorder perspective of Section 5, any data compatible with a space is itself a space.

Many of the basic definitions are spaces, including 'pass', 'null', 'bool' from section 4 and others, as shown in figure X. More examples come from noting that any data which is both idempotent and distributive is also a space. Thus if J is idempotent data, then (ap J) is idempotent and distributive, and is, therefore, a space.

6.1 Morphisms

A product F that starts with space S and ends with space T is a *morphism* from S to T and can be written $S \xrightarrow{F} T$. A morphism $S \xrightarrow{F} T$ always defines a function mapping $(S : X)$ in S to $(T : F : X)$ in T. Since spaces are idempotent, $F \cdot S{=}T \cdot F{=}F$. Since the product of data is associative, morphisms can be composed, so that if $S \xrightarrow{F} T$ and $T \xrightarrow{G} U$, then $S \xrightarrow{G \cdot F} U$. If F and G are morphisms from S to T, we can define a *sum of morphisms* by $F \oplus G = T \cdot (F + G) \cdot S$, so that

$S \xrightarrow{F \oplus G} T$. Note that morphism between any two spaces always exist, because $T \cdot S$ is a morphism from S to T. A space S is always "its own identity morphism" since $S \xrightarrow{S} S$.

Morphisms f, g from a space S to itself are closed under composition via $S \xrightarrow{f \cdot g} S$ and are closed under addition via $S \xrightarrow{f \oplus g} S$. Distribution from the right follows as in Section 5, with $(f \oplus g) \cdot h = (f \cdot h) \oplus (g \cdot h)$ for all morphisms f, g and h from S to S. Multiplicative and additive units are inherited via $1{=}S{\cdot}\text{pass}{\cdot}S{=}S$ and $0{=}S{\cdot}\text{null}{\cdot}S$, so that $1 \cdot f{=}f \cdot 1{=}f$ and $0 \oplus f{=}f \oplus 0{=}f$ for all f. For any morphism f from S to S, $0 \cdot f = 0$. The spaces where 0=1 are exactly the constant spaces, where $(S{:}X){=}(S{:})$ for all X. In the special case $S{=}$pass, the semiring of Section 5 is recovered.

Since every data $(S{:}X)$ in a space S has a corresponding constant morphism $S \xrightarrow{\text{const } (S:X)} S$, nothing is lost if we think of the endomorphisms of a space as embodying its internal structure including the data belonging to the space. As defined, a morphism from space S to space T may generally be any function from the data of S to the data of T. Morphisms which also "preserve the structure" of S and T are the 'homomorphisms' from S to T. A morphism F from S to T is a *homomorphism* if

$$F \cdot (f \oplus g) = (F \cdot f) \oplus (F \cdot g) \tag{5}$$

for all $f, g \in S$. In the special case where S and T are distributive spaces, $f \oplus g = f + g$, and F is a homomorphism if and only if F is distributive. In the special case where $S = T$, homomorphisms $h \in S$ are exactly the morphisms which distribute both from the left and from the right.

Given a space S, we can identify a number of distinguished classes of morphisms within the semiring of S as follows.

- **Subspaces.** If a morphism $f \in S$ happens to also be a space, we say that f is a *subspace* of S. The name "subspace" is justified because every data contained in f is also contained in S, and every morphism $f{\cdot}\text{X}{\cdot}f$ of f is also a morphism of S. Algebraically, subspaces partially distribute from the left with $f \cdot (g \oplus h) = f \cdot ((f \cdot g) \oplus (f \cdot h))$. Every space has subspaces 1 (the whole space), 0 (just the neutral data of the space) and every data $(S{:}X)$ in S has a corresponding constant subspace morphism.

- **The Group of units**. The collection of morphisms $g \in$S with multiplicative inverses ($g \cdot g' = g' \cdot g = 1$) is called *the group of units* of the space S. The group of units may including *involutions* where $g \cdot g = 1$. Data in S which is invariant under the group of units is called *central* data. Morphisms which commute with the group of units are called *central* morphisms.

- **Homomorphisms.** A homomorphism $f \in S$ satisfies $f \cdot (g \oplus h) = (f \cdot g) \oplus (f \cdot h)$ for all $g, h \in S$. If $f, g \in S$ are homomorphisms, then $f \cdot g$ is a homomorphism, and, if S is algebraic, $f \oplus g$ is also a homomorphism. Since 0 and 1 are always homomorphisms, homomorphisms are a sub-semiring if S is an algebraic space. Central homomorphisms are of special interest as these are, in a sense, the maximally platonic homomorphisms.

- **Algebraic Morphisms.** The morphisms of an algebraic space are automatically algebraic as well. In general, if $f, g \in$S and f is algebraic, then $g \cdot f$ is algebraic. If both f and g are algebraic, $f \oplus g$ is algebraic.

- **Neutrality** A morphism $f \in S$ *preserves neutrality* if $(f{:}S{:}){=}(S{:})$ and *reflects neutrality* if $(f{:}S{:}X){=}(S{:})$ implies $(S{:}X){=}(S{:})$. The morphisms which preserve neutrality are exactly the morphisms which satisfy $f \cdot 0{=}f$ in addition to satisfying $0 \cdot f = 0$. Since 0 and 1 preserve neutrality, and since $(f \cdot g)$ and $(f \oplus g)$ preserve neutrality if f and g do, neutrality preserving morphisms are a subsemiring of S. If $f \in S$ and $g \in S$ reflect neutrality, then so does $f \cdot g$.

A morphism $f \in S$ is a *positive morphism* if $(f{:}X)$ is never equal to the neutral data of S. If f and g are positive in S, then so is $f \cdot g$ and, if S itself is neutral, $f \oplus g$ is also a positive morphism.

Since the morphisms of S always include a constant subspace for each data $S : X$ in S, the collection of morphisms of S is strictly richer than the data contents of S. We shall see that it is fruitful to think of a space S as its collection of endomorphisms.

6.2 Subspaces are substructures

Suppose that space S has subspace s and space T has subspace t, and suppose that $F{=}t \cdot X \cdot s$ is a morphism from s to t. Since $(t \cdot X \cdot s) = (T \cdot t \cdot X \cdot s \cdot S)$, F is also a morphism from S to T, with the extra property that F respects the structure of s and t in the sense that $t \cdot F = F \cdot s$ (figure X). Since the morphisms of S automatically includes a constant subspace for each data $(S : X)$ in

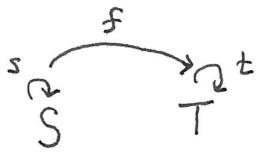

Figure 1: *Subspaces are substructures. If s is a subspace of S and t is a subspace of T, then a morphism from s to t is a morphism from S to T which preserves the corresponding structure.*

S, the morphisms of S are strictly richer than the contents of S.

6.3 Isomorphisms

If the diagram in Figure X commutes for spaces S, T and for some homomorphisms α, α', we say that S and T are *isomorphic* spaces. It easily follows that the semirings of S and T are isomorphic as monoids, and thus, have the same idempotents, involutions, group of units, homomorphisms, subspaces, constants and identity. As a special case, if S and T are distributive spaces, then α and α' may be any distributive morphism.

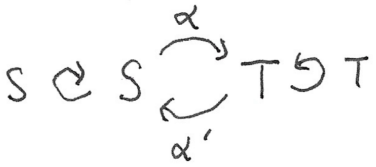

Figure 2: *In a definition adopted from Category Theory, spaces S and T are isomorphic if the diagram commutes for some homomorphisms α and α'.*

6.4 Organic Spaces

Since the general concept of a collection leads to Definition 3, it is clear that spaces must play a fundamental role in the mathematics of pure data, perhaps in a way analogous to the role that Sets or Categories play in classical mathematics. However, it is also clear that spaces are quite different from either Sets or Categories. Most basically, a set is defined by its contents and a space is not.

For example, the space (ap {a}) contains the same data as (is a), but these spaces are not equal. Since a space always contains neutral data (S:) a space can never be empty. An implicit operation mapping (S:X) and (S:Y) to (S:X Y), means that a space always somes with some algebraic structure. Although spaces have morphisms and associative composition, the theory of spaces must also be quite different than Category Theory because, for example, morphisms exist between any two spaces. These considerations suggests that pure data spaces may provide a different, and possibly interesting perspective on known mathematics. This possibility informs the strategy of the rest of this paper. We search for the simplest, most natural, most "organic" spaces, understand them in the context of the structures above, and compare with known mathematics.

The pure data framework gives us some advantages in searching for the "simplest" spaces, since we could search all pure data and find all spaces up to a specified maximum data width and depth. This direct approach is computationally difficult, but we can proceed in this spirit, by starting with the basic definitions which also happen to be spaces, some of which are shown in figure X. We can take the point of view that we let spaces "grow organically" from the space of all pure data and examine the simplest results first. Aside from the "organic" theme, a second theme is

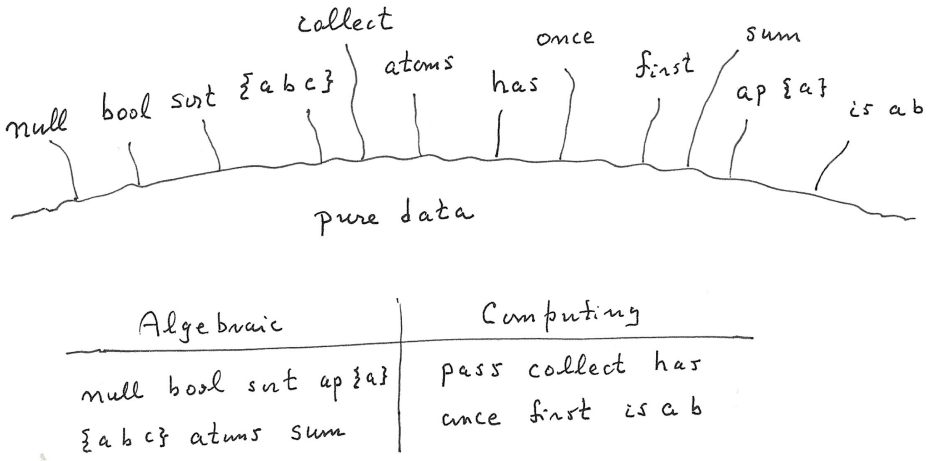

Figure 3: *The space of all pure data contains all mathematical objects. Our strategy for exploring this space is to first introduce the minimal combinatorics implicit from the foundational finite sequence concept, and then examine the simplest spaces first as they appear "organically." Since algebraic spaces are commutative, they, in a sense, transcend the foundational sequence concept and are more "Platonic."*

to take special notice of 'algebraic' data to see how this matches with the intuition that algebraic spaces and morphisms will be the most mathematically significant, worthy of special naming, and the most Platonic in the sense of being independent of this particular system.

7 The Space of all data

Since spaces are fixed points on their contents, any space that contains all data, must be a fixed point on all data, and so the distributive space 'pass' is the unique space containing all pure data. We shall think of spaces as "growing from the garden" of pure data, in other words, from the contents of pass.

Before growing the simplest spaces, it helps to orient ourselves by examining the semiring struc-

ture of 'pass' itself. The multiplicative and additive identities of pass are 1=pass·pass·pass=pass, and 0=pass·null·pass=null respectively. Note that every data X is actually a morphism of pass, since X=pass·X·pass. Consider the classes of morphism defined in Section 6.

- **Subspaces.** Since every data is a morphism of pass, every space is a subspace of pass.
- **The Group of Units.** The units of pass are the permutations such as (swap 1 2) and (rev). The central data of pass are the data which are invariant under permutations. In other words, sequences of identical atoms, such as (:) (:) (:), or (a a a a a). These can be interpreted as "organic natural numbers."
- **Homomorphisms.** All distributive data are morphisms of pass, and are, therefore homomorphisms of pass. This includes distributive spaces such as (ap J) for idempotent J, and distributive non-spaces such as (ap {B B}), which doubles every atom of input. The central morphisms are those which commute with the permutations, and we recognize the central homomorphisms as natural number multiplication of the central data. In this sense, the natural numbers are the most Platonic morphisms of pass. This is explored further in Section X.
- **Algebraic Morphisms.** Since algebraic morphisms commute with permutations, all of the central homomorphism are also algebraic. The issue is whether there are non-distributive morphisms which are also algebraic. A prominent example is 'bool' which is an algebraic space, but is not distributive. Bool is further explored in section Y.
- **Neutrality.** A morphism D∈pass is neutral preserving if (D:)=(), is neutral reflecting if (D:X=()) implies X = () and is positive if (D:X) is never empty. For example,
  - pass, rev, first, {B B}, (is a) are neutral preserving.
  - pass, rev, first, are neutral reflecting.
  - const 1 2 3, {#}, {B 1 2 3} are positive.

As an example of an isomorphism, consider the isomorphism between 'pass' and 'has' illustrated in Figure X. This isomorphism $X \leftrightarrow (: X)$ embodies "storing and retrieving data X". A similar isomorphism $X \leftrightarrow (a : X)$ exists for every atom a.

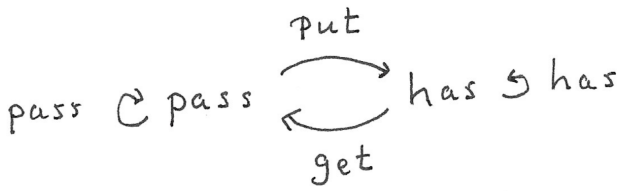

Figure 4: *The spaces* **pass** *and* **has** *are isomorphic via* $X \leftrightarrow (: X)$. *A similar isomorphism exists for each atom type. This shows that the space of all pure data is isomorphic to a subspace of itself for each atom type.*

In classical Mathematics, the concept of a Set intentionally gives no information about it's contents, and so the Set of all Sets hardly gives any insight into the contents of Mathematics in general. This situation is interestingly different. Pass contains everything, but it also has nontrivial structure and gives specific indications about what will be most important (algebraic morphisms and central homomorphisms, natural numbers and booleans in this case). Pass is the "garden" where we will search for the simplest, most "organic" spaces.

8 Organic Numbers

As we have seen the "organic" natural numbers appear as the central data of pass where we identify (:) (:) (:) or (a a a) with the natural number "3" depending on a conventional choice of atom. Thus, we want to find a central space morphism N where the data of the space contains one sequence for each natural number. There are many spaces which do this, for example,

1) The space (atoms) contains all sequences of (:) atoms, (), (:), (:) (:),....

2) The space (ap const (:)) also contains all (:) sequences.

3) The space (ap const a) contains (), a, a a, a a a,....

4) The space (is a), also containing (), a, a a, a a a,....

Choices 1 and 2 are identical, choice 3 is compatible with 1 and 2 in the preorder sense, only differing in the choice of standard atom. Space 4 is not the same space as 3, even though it contains the same data. The essential point, however, is that all four are isomorphic and therefore, have the same semiring structure so they are equivalent for the purpose of the analysis below. To be definite, we choose N to be the space (is a).

The first thing to notice about N is that it's main structure is natural number addition, since (N:X Y) is the natural number sum of (N:X) and (N:Y). Since homomorphisms are distributive, a homomorphism f of N is defined by f:a, and, thus, f is multiplication by some natural number, for instance, (ap const a a a) is multiplication by 3. Since homomorphisms are distributive morphisms, they automatically distributes over addition in N. Since the identity is the only unit, all homomorphisms are central.

Subspaces of N are indicated by subspace morphisms. As always, the subspace 1 is the whole of N and the subspace 0 consists of just the neutral data of N, which, since in is distributive is the empty sequence. The following are easily verified to be classes subspace morphisms as well.

1. **Saturation subspaces** An morphism which computes, for example, min(n,2) such as (min a a), is a subspace containing the three data {(), a, a a}.

2. **Modular arithmetic subspaces** An morphism which removes three (a) atoms while possible, such as (while rem a a a) is a subspace which computes the sum of it's inputs modulo 3. Note that (while rem a a a) contains the same data as (min a a), but they have a completely different algebraic structure.

These two unsurprising subspaces have a slightly more surprising generalization. For $p, q \geq 1$, let rem(p,q) be the morphism defined by

$$\text{while } n \geq p, q : \text{remove p from n} \tag{6}$$

The rem(p,q) are also subspaces and modular sum and saturation cases since rem(p,p) is $n \mapsto n$ mod p and rem(1,q) is $n \mapsto \min(n,q-1)$. General arguments $p, q \geq 1$. The rem subspaces are closed under composition via

$$\text{rem}(p,q) \cdot \text{rem}(p',q') = \text{rem}(\text{LCM}(p,p'), \max(q,q')) \tag{7}$$

where LCM is the least common multiple. This makes the rem a closed commutative semilattice of subspaces of N.

This completes the subspace analysis of N since we can show that every subspace of N is either the whole of N, a constant, or is rem(p,q) for some $p, q \geq 1$.

Proof. Let $f : \mathbf{N} \to \mathbf{N}$ where $f(x+y) = f(f(x)+y) = f(x+f(y))$ for all $x, y \in \mathbf{N}$... □

At this point, it is quite possible to consider pairs of natural numbers representing, say, the pair (3,2) as (n a a a : a a) or (n: (:a a a) (:a a)). Rather than doing this, however, we first want to investigate the most organic generalizations of (is a), for instance, (is a b), which we refer to as N_2. As with the case of N, let's consider subspaces of N_2.

- The subspaces 1 and 0 give the whole of N_2 and just the empty sequence as subspaces, respectively. Projections N_2·(is a)·N_2 and N_2·(is b)·N_2 produce two N-isomorphic subspaces.
- Lexical sorting of N_2 sequences is a non-distributive subspace of N_2. Call this endomorphism N_2.lex, the data of N_2.lex can be written $a^m b^n$ for $m, n \geq 0$. As in the previous case, a homomorphism M of N_2.lex is defined by

  $M : a \mapsto a^{m_{11}} b^{m_{21}}$, and
  $M : b \mapsto a^{m_{12}} b^{m_{22}}$

  for some choice of $\begin{pmatrix} m_{11} & m_{21} \\ m_{12} & m_{22} \end{pmatrix}$ so the action of M is matrix multiplication. Thus, the data of N_2.lex are pairs of natural numbers the homomorphisms are $\mathrm{Mat}_{2\times 2}(\mathbf{N})$. The automorphisms of N_2.lex are $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and the central products are the matrices $\begin{pmatrix} m & n \\ n & m \end{pmatrix}$ for $m, n \in \mathbf{N}$.
- Reducing data in N_2 by removing (a b) and (b a) subsequences defines another subspace (N_2.reduce) consisting of sequences containing only a's or only b's but not both. It is easy to see that data in N_2.reduce can be identified with ring of integers $\mathbf{Z}$. The action of the subspace itself does integer addition. The two automorphisms of N_2.reduce are the identity and the endomorphism which swaps $a \leftrightarrow b$. A homomorphism f is determined by $f : a$, since $f : b$ is determined by commuting with swap. Thus, the central homomorphisms of N_2.reduce are exactly integer multiplication, distributing over addition since homomorphisms are distributive.
- Lexical sort followed by reducing (b b) to (b) results in a subspace equivalent to $\mathbf{N} \times \{0, 1\}$....MORE

Continuing to (is a b c),...MORE.

Continuing to (is a b c d), one gets $\mathbf{N}^4$ with lexical sort, $\mathbf{Z}^2$ if one adds (a b)=(b a)=(c d)=(d c)=1. The latter case has an involution (a,b)↔(c,d) making the central homomorphisms equivalent to Complex number multiplication....MORE

8.1 Number Sequences

Given the space N from the previous section, let $\mathbb{N}$ be

$$\text{ap (put n)} \cdot N \cdot \text{(get n)} \tag{8}$$

where 'n' is some chosen atom. Since (put n) · N · (get n) is idempotent, $\mathbb{N}$ is a distributive space containing sequences of n-atoms containing N-data, such as

$$\text{T} = \text{(n : a a a) (n :) (n : a a) (n : a) (n :)} \tag{9}$$

Unlike the case of N, the action of $\mathbb{N}$ is merely to concatenate sequences. It's easy, however, to identify natural number sum again as one of several subspaces

- sum:T = (n:a a a a a a)
- sort:T = (n:) (n:) (n:a) (n:a a) (n:a a a)
- min:T = (n:)
- max:T = (n:a a a)
- first:T = (n:a a a)

All but the last are algebraic subspaces and, predictably, have established names. Some morphisms of $\mathbb{N}$ can be "inherited" from $f \in$N. Define inner:F to be

$$\mathbb{N} \cdot (\text{put n}) \cdot \text{F} \cdot (\text{get n}) \cdot \mathbb{N} \tag{10}$$

so that (inner:F):X means letting F act on the concatenated N-contents of X, returning the result in a single n-atom. The sum morphism above, for instance, is equal to inner:N. The construction of Equation 7 works for any space, and so we can define a "functor" *Seq* to be

$$(\text{put A}) \cdot \text{B} \cdot (\text{get A}) \tag{11}$$

Then $\mathbb{N}$ is equal to (Seq n:N) and given any atom s and any space S, (Seq s:S) is the space of S-values stored in s-atoms. Similarly, if we define *inner* to be

$$\mathbb{N} \cdot \{(\text{put n}) \cdot \text{B} \cdot (\text{get n})\} \cdot \mathbb{N} \tag{12}$$

then inner:F is the inner version of F$\in$S. Other general constructions of this type are possible. For example, let *series* be

$$\mathbb{N} \cdot \{\text{B } (\text{A} : \text{B})\} \cdot \mathbb{N} \tag{13}$$

and, thus (series:F) applies F to data appending the result. For example, the morphism series$\cdot$(inner:back a a) generates the Fibbonacci sequence (n:a) (n:a) (n:a a) (n:a a a)....

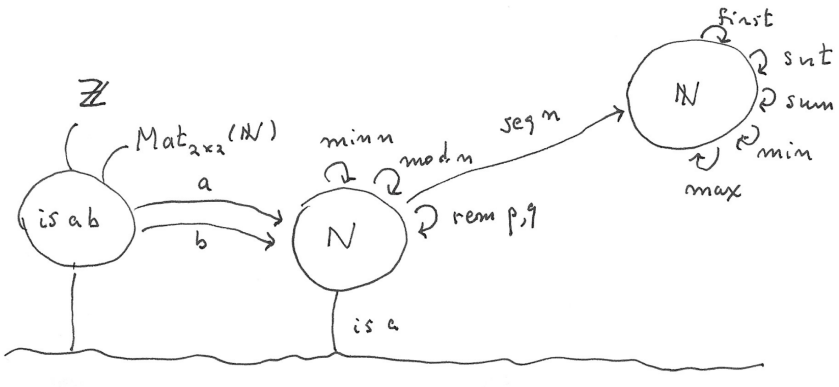

Figure 5: *Organic numbers.*

SUMMARIZE...

$f \cdot g$	ID	TRUE	FALSE	NOT
ID	ID	TRUE	FALSE	NOT
TRUE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE
NOT	NOT	FALSE	TRUE	ID

$f \oplus g$	ID	TRUE	FALSE	NOT
ID	ID	ID	FALSE	FALSE
TRUE	ID	TRUE	FALSE	NOT
FALSE	FALSE	FALSE	FALSE	FALSE
NOT	FALSE	NOT	FALSE	NOT

Table 1: *Product and sum of the four morphisms ID, TRUE, FALSE, NOT of the space bool.* Note that ID is the unit of multiplication and TRUE is the unit of addition, and we have $f \oplus g = g \oplus f$, since bool is algebraic, and $f \oplus f = f$.

9 Boolean

The space 'bool' is algebraic and central, but is not a homomorphism. Since bool contains two data: () for *true* and (:) for *false*, bool has only four morphisms: the identity ID=bool, two constants: TRUE={} and FALSE={(:)}, and one involution NOT=bool·not·bool. The morphism semiring is explicit in Table X, where it's useful to note that $\oplus$ is commutative since **bool** is algebraic, and $f \oplus f = f$ for $f \in$**bool**.

The subspaces of bool are ID, TRUE and FALSE, so bool has only trivial subspaces. The morphisms ID and TRUE are neutral, ID is the only positive morphism and all morphisms are algebraic since bool itself is algebraic. The group of units is ID and NOT, and ID is the only central morphism.

9.1 Boolean Sequences

As in the case of N, we can let $\mathbb{L}$=(Seq b:bool), be the distributive space of bool-valued data stored in b-atoms, so a typical data in $\mathbb{L}$ is

$$(\text{b :})\ (\text{b :})\ (\text{b : (:)})\ (\text{b :})\ (\text{b : (:)})\ (\text{b : (:)}) \tag{14}$$

Let's agree to write such sequences replacing (b:) with T, (b:(:)) with F and the empty sequence with 0, so that the above is written TTFTFF. Let's also consider the shortest subspaces of $\mathbb{L}$ first. The space morphism **first**$\in \mathbb{L}$ are the sequences of length at most 1. Let's call this space $\mathbb{L}_1$ and, similarly, let $\mathbb{L}_2$ be (first 2)$\in \mathbb{L}$. Thus, $\mathbb{L}_1$ contains data {0,T,F} and $\mathbb{L}_2$ contains data {0,T,F,TT,TF,FT,FF}. For $\mathbb{L}_1$, we can specify a morphism $f \in \mathbb{L}_1$ by listing the values of f on 0,T,F in standard order, so, for example, '0TF' denotes the identity morphism. Since $\mathbb{L}_1$ has 27 morphisms, we can be completely explicit about this space. Table X shows the semiring of morphisms and Figure Y shows special properties. Even with relatively small space like $\mathbb{L}_1$, all the classes of morphisms appear in a nontrivial way.

Moving to $\mathbb{L}_2$, there are already $7^7 = 823,543$ morphisms; far too many to be as explicit as in the case of $\mathbb{L}_1$. One simple attempt is to examine just the inner morphisms, since there are only eight of these consisting of the morphisms $\mathbb{L}_2 \cdot (\text{put } b) \cdot \text{F} \cdot (\text{get } b) \cdot \mathbb{L}_2$ for some data F. These eight depend only on the total number of (:) values in the b-atoms of input, and each morphism returns exactly one b-atom. The inner morphisms of $\mathbb{L}_2$ are all algebraic homomorphisms, so we may expect them to be mathematically interesting.

Table Y shows the eight morphisms and their values on {0,T,F,TT,TF,FT,FF}. Looking at the values for the last four entries {TT,TF,FT,FF} we recognize that these are slight generalizations of the eight standard symmetric binary boolean operators. The values of the eight morphisms on {0,T,F} suggests that the standard binary operator names ('OR' as opposed to 'any') are not particularly illuminating in this larger context.

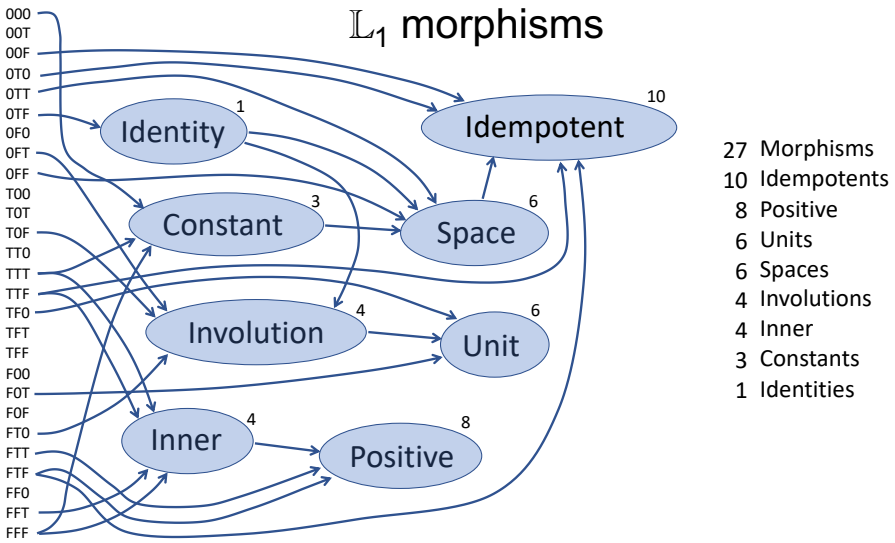

Figure 6: *Morphisms of* $\mathbb{L}_1$.

Table 2: The eight inner morphisms of $\mathbb{L}_2$ slightly generalize the eight standard symmetric binary boolean operators.

$e_i \in \mathbb{L}_2$	0	T	F	TT	TF	FT	FF	standard	generalized	description
e_1	T	T	T	T	T	T	T	TRUE	**always**	always true
e_2	T	T	T	T	T	T	F	OR	**any**	any are true
e_3	T	T	F	T	F	F	T	XNOR	**even**	even (:)s
e_4	T	T	F	T	F	F	F	AND	**all**	all are true
e_5	F	F	T	F	T	T	T	NAND	**notall**	not all are true
e_6	F	F	T	F	T	T	F	XOR	**odd**	odd (:)s
e_7	F	F	F	F	F	F	T	NOR	**none**	none are true
e_8	F	F	F	F	F	F	F	FALSE	**never**	never true
Number of (:)	0	0	1	0	1	1	2			

10 Summary, Global Strategies, Questions

11 Glossary

1. Basics
   (a) δ_{const} : (const A : B) $\mapsto$ A
   (b) δ_{put} : (put A : B) $\mapsto$ (A:B)
   (c) δ_{get} : (get : (:B)) $\mapsto$ B....FIXME
   (d) δ_{atoms} : (atoms : b B) $\mapsto$ (:) (atoms : B)
   (e) δ_{bin} : (bin A:B) $\mapsto$ (bin A:B), atom
2. Control
   (a) δ_{if}: (if ():B) $\mapsto$ B

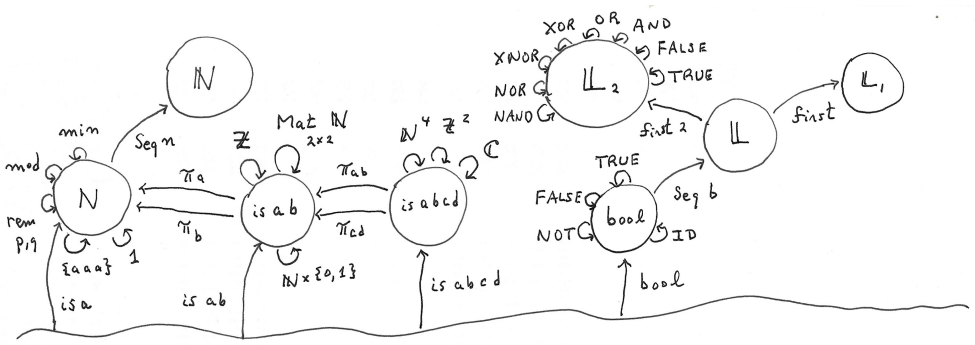

Figure 7: *Some of the spaces and morphisms discussed in the text and "grown organically" from the space of all pure data. The main starting points are natural number spaces (central homomorphism spaces) and bool (central algebraic spaces).*

(b) δ_{if}: (if a A:B) $\mapsto$ ()

(c) δ_{while}: (while A:B) $\mapsto$ B if (A:B)=B

(d) δ_{while}: (while A:B) $\mapsto$ (while A: A : B)

3. Semiring

   (a) prod

   (b) sum

4. Definition

5. Sequence

   (a) δ_{first} : (first : b B) $\mapsto$ b

   (b) δ_{first} : (first : ()) $\mapsto$ ()

   (c) δ_{last} : (last : B b) $\mapsto$ b

   (d) δ_{last} : (last : ()) $\mapsto$ ()

   (e) has

   (f) hasnt

   (g) is

   (h) isnt

   (i) once

   (j) δ_{rev} : (rev : B b) $\mapsto$ b (rev:B)

   (k) δ_{rev} : (rev : ()) $\mapsto$ ()

   (l) rem

   (m) sort

References

[1] *Pure Data Foundation of Mathematics and Computing*, Saul Youssef, 2023.

[2] Nicholas Griffin (2003-06-23). *The Cambridge Companion to Bertrand Russell*. Cambridge University Press. p. 63. ISBN 978-0-521-63634-6.

[3] Barry Mazur, *When is one thing equal to some other thing?*, Harvard University, 2007, https://people.math.harvard.edu/∼mazur/preprints/when_is_one.pdf.