PURE DATA SPACES....DRAFT...DRAFT...DRAFT

SAUL YOUSSEF
Department of Physics
Boston University

June 4, 2025

Abstract

All about spaces.

1 Overview

In Reference 1, we introduced an axiomatic framework, based on the finite sequence as the foundational concept. With finite sequences assumed, we had

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

This kind of data, such as (:) (:(:)) ((:):(:) (:)), is "pure data" in the sense that it is "made of nothing." Such data has two natural operations: *concatenation* of data A with data B, written (A B), and *pairing* data A and data B as a *coda*, written (A:B).

2 *Definitions are embodied by a chosen partial function δ from codas to data called a context.*

As modest as it is, 1 and 2 appear to be able to capture Mathematics in general. Consequences include:

- Fixed points of δ (*atoms*), represent fixed data such as bits, bytes and text;
- Data outside the domain of δ are *variables*, which might become defined over time as *definitions* are added to δ;
- δ contains it's own *language*, allowing user specification of data and adding definitions to δ;
- Equality of data A=B is determined by $c \sim \delta(c)$ and by compatibility with finite sequences. Proof and computation are then defined by the equality. A sequence $A_1 = A_2 = \cdots = A_n$ is both a proof of A_n given A_1 and a computation $A_1 \mapsto A_n$. Proof and computation are essentially the same thing;
- Data is *true* if it is empty, *false* if it is atomic, and *undecided* otherwise. Undecided data may also be *undecidable*, as in the Godel phenomena or other seeming paradoxes. Data which is *never false* as definitions are added is a *theorem*.

In this work, we investigate the concept of a *space*, introduced in Reference 1.
...SEARCHING....ORGANICS....

2 Foundation

In [1], we argue that Mathematics and Computing as a whole can be captured within a very small axiomatic framework, based on the **finite sequence** as the foundational undefined 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 qualifies as a coda, written '(():())' or '(:)'. Any finite sequence of codas is data, so, for example (:) (:(:)) ((:):(:(:))) is data consisting of a sequence of three codas. We call this *pure data* because 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 that $c = (c^L : c^R)$.

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 relation 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, 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 any 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.

3 Genesis

The mathematical objects that we are used to will all be represented as pure data, and all meaning about mathematical objects, including what constitutes a valid proof or a valid computation, is determined by an assumed context embodying a chosen collection of definitions. For instance, a valid proof in δ is just a sequence $\mathrm{A}_1 \overset{\delta}{=} \mathrm{A}_2 \overset{\delta}{=} \ldots \overset{\delta}{=} \mathrm{A}_n$, which can be viewed either as proving $\mathrm{A}_1 \overset{\delta}{=} \mathrm{A}_n$ or as computing $\mathrm{A}_1 \mapsto \mathrm{A}_n$. This is discussed in [1] and will gradually become clearer as we go. The immediate issue is to understand what constitutes a "valid" definition and how to choose an initial context.

In the beginning, there are no definitions, and the corresponding context is uniquely the empty partial function from coda to data, δ_0. Within δ_0, data are equal only if they are identical as pure

data, the only valid proofs are $X \stackrel{\delta}{=} X$ for any X and the only valid computations do nothing $X \mapsto X$ for any X. To define a non-empty context $\delta_0 \leq \delta$, we need a way to specify the domain of δ which is unchanged by any later later definition. Since the empty sequence is the only invariant, the way to specify if a coda (A:B) is 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 decide on what (:) maps to. There are three possibilities;

1. $\delta : (:) \mapsto ()$,
2. $\delta : (:) \mapsto (:)$, or
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 means that for any data A the number of (:) atoms in A grows without limit. Choice 3 is degenerate in the sense that no computation could produce a final answer. Thus, we are constrained to choice 2, meaning that (:) must be an atom in δ and, therefore, is an atom for any $\delta' \geq \delta$. It is convenient to generalize choice 2 and let δ :(:X)$\mapsto$(:X) for all data X, so that every coda (:X) is an atom.

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 required context partial ordering.

4 Definitions

Starting with the context δ chosen above, any coda $(:X)$ is an atom, so there is an easy supply of invariants to define new atoms. 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 $\delta \cup \delta_{(:)} \cup \delta_{((:):)} \cup \delta_{((:):(:))}$.

Given text as atomic data, we can add an identity 'pass' so that $(\text{pass} : X{=}X)$ for all data X.

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

and define 'null' so that $(\text{null} : X{=}())$ for all X.

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

It is convenient to define "getting the left and right components of a coda" like so

- $\delta_{\mathbf{left}}$: (**left** A:B) $\mapsto$ **left** A
- $\delta_{\mathbf{right}}$: (**right** A:B) $\mapsto$ B

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

We proceed to define the minimal combinatorics involving of A and B as finite sequences. For example, **rev** "reverses the order of B":

- $\delta_{\mathbf{rev}}$: (**rev** A : B b) $\mapsto$ b (**rev** : B)
- $\delta_{\mathbf{rev}}$: (**rev** A : ()) $\mapsto$ ()

where b is an atom. The B parts of these definitions can sometimes be referred to as "the input", so, for instance, (rev:1 2 3) = 3 2 1. We may say that 1 2 3 is the "input to rev" and "3 2 1" is the result.

Many of the most basic and important operations don't have common names, so, for example, the name 'ap' is chosen to suggest "apply A to each atom of B."

- $\delta_{\mathbf{rev}}$: (**ap** A : b B) $\mapsto$ (A:b) (**ap** A : B)
- $\delta_{\mathbf{rev}}$: (**ap** A : ()) $\mapsto$ ()

See the glossary for more definitions as needed in the text.

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 basic operations via text blank space and text colon. So, the basic 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 easily be resolved by choosing and order in the text string and by ordering the above, 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.

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.

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).

Given a language expression in the form of a byte sequence L, the corresponding data is (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 means that there is also no such thing as a "run time error."

If one continues in this vein, there are about 50 definitions which define obvious combinatorial operations definable in the general setting (a A : B) $\mapsto$ $F(a, A, B)$ and referring only to A and B as finite sequences. As we will see, this is "organic" base of naturally occurring definitions plus the language will be a starting point in searching for "spaces." 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 Spaces

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 and associative sum via

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

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

for all data X. Since (pass $\cdot$ A)=(A $\cdot$ pass)=A and (A + null)=(null + A)=A for any A, and call this 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. If A$\cdot$B=A for data A, and B, we say that A *absorbs* B. If A absorbs B and B aborbs A, we have an equivalence A$\sim$B and we say that data A and B are *compatible*.

In order to do mathematics, one needs a way to define and refer to abstract collections of mathematical objects, which we will call a "space." Thus, given suitable data S, we need a way for S to define a collection. There are a couple of plausible ways to do this. One could define the collection corresponding to S to be:

- The data S:X for any data X.

- The fixed points of S.

These ideas coincide if we require S to be idempotent, so that S:X is always a fixed point. It is natural to 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

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

for all data X, Y. These requirements are satisfied provided that S is "associative."

Any data A can be viewed as defining a binary product on data defined by $(X \overset{A}{*} Y)=(A:X\ Y)$. The operator $\overset{A}{*}$ is associative if and only if $(A:X\ Y)=(A:(A:X)\ Y)=(A:X\ (A:Y))$ for all X,Y. In this case, we say that A is *associative*. Associative data is automatically idempotent and also satisfies equation 2, thus providing everything needed to also qualify as a space.

Definition 3. Associative data is a **space**.

Many of the basic definitions from section 4 provide examples of spaces including **pass**, **null**, **bool**, first, once, has...any constant is a space; any distributive idempotent data is a space; for any idempotent I, ap I is a distributive space.

A product F that starts with space S and ends with space T is called a **morphism** from S to T and can be written $S \xrightarrow{F} T$. F 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 is associative, morphisms can be composed as usual so if $S \xrightarrow{F} T$ and $T \xrightarrow{G} U$, then $S \xrightarrow{G \cdot F} U$.

5.1 Morphisms

It is helpful to think of morphisms from S to S as belonging to S, and so we will refer to a morphism $S \xrightarrow{f} S$ as a morphism "of S" and we may indicate this as $f \in S$. Morphisms of a space S have a semiring structure which is a slight generalization of the semiring defined in the previous section. Given morphisms $f, g \in S$, we have a semiring with product $f \cdot g$ as previously defined, and with associative sum $f \oplus g$, defined to be $S \cdot (f + g) \cdot S$. Distribution from the right follows as before, so $(f \oplus g) \cdot h = (f \cdot h) \oplus (g \cdot h)$ for all $f, g, h \in$S. Multiplicative and additive units are inherited from the space **pass** via $1 = S \cdot \textbf{pass} \cdot S = S$ and $0 = S \cdot \textbf{null} \cdot S$, so that $1 \cdot f = f \cdot 1 = f$ and $0 \oplus f = f \oplus 0 = f$ for all $f \in$S. Note that if S is algebraic, the sum of morphisms is commutative. Since every space S is also a product of spaces starting at S and ending at S, S qualifies as a morphism so every space S is a) a morphism from S to itself, so $S \xrightarrow{S} S$ or $S \in S$, and is b) the multiplicative unit 1 of the S semiring and c) a morphism $S \in$ **pass** of the space of all data since S=**pass**·S·**pass**.

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

- **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 $f \cdot X \cdot f$ of f is also a morphism of S. Algebraically, subspaces partially distribute from the left as in $f \cdot (g \oplus h) = f \cdot ((f \cdot g) \oplus (f \cdot h))$. In every space, 1, 0 and every constant morphism are subspaces.

- **Neutral Morphisms.** It is easy to see that $0 \cdot f = 0$ for any $f \in S$. If $f \cdot 0 = 0$ also holds, we say that f is a *neutral* morphism. Since 0 and 1 are neutral and since $f \cdot g$ and $f \oplus g$ are neutral if f and g are neutral, neutral morphisms are a sub-semiring of S.

- **Positive Morphisms.** Morphism $f \in S$ is *positive* if $(f:X)=(0:X)$ implies $(S:X)=(S:)$. For example, data $D \in$ **pass** is positive if $D:X=()$ implies $X=()$. If f and g are positive, so is $f \cdot g$ and $f \oplus g$, positive morphisms are closed under addition and multiplication. FIX ME

- **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.

- **Homomorphisms.** If a morphism f is distributive, we say that f is a **homomorphism** of S since $f : X\ Y = (f : X)\ (f : Y)$ is a morphism of the main structure of S. Homomorphisms fully distribute from the left as well as the right as in $f \cdot (g \oplus h) = (f \cdot g) \oplus (f \cdot h)$. Homomorphisms are also closed under addition and multiplication, and since 0 and 1 are always homomorphisms, the homomorphisms are a sub-semiring of S.

- **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 $g \cdot g = 1$. As usual, involutions g and h commute if and only if $g \cdot h$ is also an involution. 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.

It is easy to confirm that if spaces S and T are isomorphic as in figure X, it follows a) that α and α' are bijections between the contents of S and the contents of T, b) the semirings of S and T are isomorphic as monoids, c) if α and α' are homomorphisms, S and T are also isomorphic as semirings.

5.2 Subspaces are substructures

Suppose space S has subspace s and space T has subspace t, and 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).

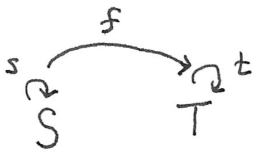

Figure 1: *Subspaces are substructures.*

Since the morphisms of S automatically includes a constant subspace for each data S:X in S, the morphisms of S are strictly richer than the contents of S.

5.3 Isomorphisms

Borrowing a definition from category theory, spaces S and T are *isomorphic* if the diagram in figure X commutes for some morphisms α and α', where S and T serve as their own "identity morphisms." It easily follows that the semirings of S and T are isomorphic as monoids, and thus, have the same idempotents, involutions, group of units and identity. If α and α' are also homomorphisms, the semirings of S and T are isomorphic as semirings.

Now we do examples of spaces and their morphisms, growing spaces as organically as possible and aiming for the "most mathematical" spaces in the sense of the most algebraic and central.

5.4 Organic Spaces

Since the general concept of a collection leads directly 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

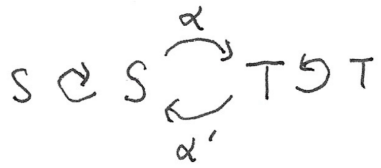

Figure 2: *In a definition adopted from Category Theory, spaces S and T are isomorphic if the diagram commutes, using S and T themselves instead of identity morphisms.*

is not. For example, the space (ap a) contains the same data as (is a), but these are not the same. Since a space always comes with an implicit operation mapping (S:X) and (S:Y) to (S:X Y), spaces comes with a much richer internal structure than Sets do in general as defined above. Although spaces have morphisms and associative composition, the theory of spaces must be quite different than Category Theory because morphisms exist between any two spaces. This suggests that pure data spaces could provide a different and possibly useful perspective on known mathematics. This is the strategy of the rest of this paper. We want to search for the simplest, most natural, most "organic" spaces, understand them in the context of the structures above, and compare with known mathematics. Another difference with the pure data framework is that searching for all spaces is well defined. We proceed by starting with the foundational definitions and language as already defined in section X and examine which spaces "grow organically" from the total collection of pure data. Figure X illustrates how we start. A substantial number of the initial definitions are already spaces. We start with these and then search for "less organic" spaces defined to be longer or deeper as data.

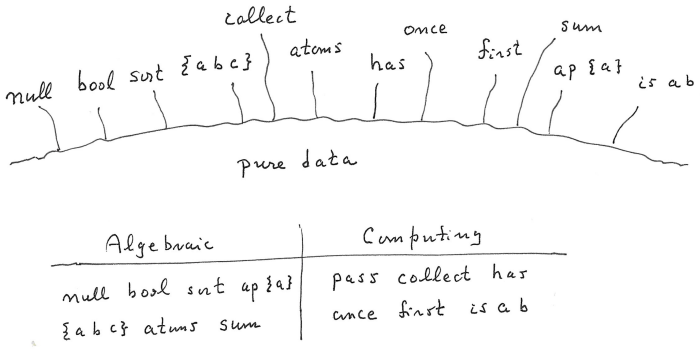

Figure 3: *Growing spaces "organically" from pure data.*

Note that in figure X, roughly half of the most organic spaces are commutative in the sense that $S : XY = S : YX$ for all X, Y. A space with this property has, in a sense, transcended the one foundational concept of pure data: the concept of a finite sequence. Because of that, we view such spaces as more mathematical and more Platonic, while the non-algebraic spaces have a more prosaic, more Computer-Science-y flavor. We will call commutative spaces with this property *algebraic* spaces.

6 The Space of pure data

Since idempotents have fixed point images, the only space whose image is all pure data is the identity space **pass**. Let's examine the structure of **pass** as a space. From the preorder perspective, **pass**

- **Subspaces**. Every space S is a subspace **pass**·S·**pass** of **pass**.
- **Neutral morphisms**. The zero of the semiring of **pass** is, by definition, **pass**·**null**·**pass**, which is equal to **null**. The neutral morphisms of **pass** is the subring of data D satisfying D:X=().
- **Positive**. The positive morphisms of **pass** is the subring of data D such that D:X=() implies X=().
- **Homomorphisms**. Homomorphisms of **pass** is the subsemiring of distributive data.
- **Group of units**. The group of units is the group of sequence permutations such as **swap 1 2** or **rev**.

The central data are sequences of identical atoms such as (:) (:) (:) or (a a a). The central morphisms are ap {a} etc.

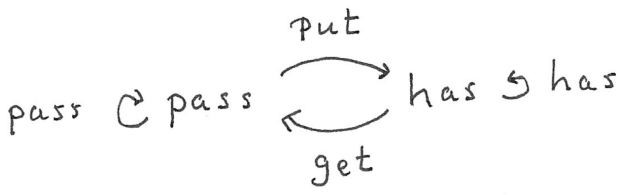

Figure 4: *In a definition adopted from Category Theory, spaces S and T are isomorphic if the diagram commutes, using S and T themselves instead of identity morphisms.*

Isomorphisms with has, has m for any atom m, etc.
Compare Spaces vs Sets vs Categories

- Unlike sets, spaces cannot be empty.
- Unlike sets, the content of a space does not determine the space.
- Unlike in category theory, every space is a morphism and there are morphisms between any pair of spaces.
- The set of all sets and the category of all categories is very flaccid. However the space of all data has lots of structure which is deeply shared.

7 Organic Numbers

The natural numbers occur most organically simply as sequences of identical atoms, so we might think of (:) (:) (:) or (a a a) as "3", depending on a conventional choice of atom. Thus, to study natural numbers, we want to find a space N where the data of the space contains all of the natural number lengthed sequences of some chosen atom. There are many spaces which do this, for example,

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

2) The space **ap const a** contains {(), a, a a, a a a,. . . }.

3) The space **is a**, also containing {(), a, a a, a a a,. . . }.

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

2) The space `ap const a` contains {(), a, a a, a a a,. . . }.

3) The space `is a`, also containing {(), a, a a, a a a,. . . }.

These are three different spaces. Choices 1) and 2) are compatible in the preorder sense, however 3) is not compatible with the others and, for instance, has a different kernel. The main point, however, is that 1) 2) and 3) are isomorphic and they therefore have the same semiring structure. To be definite, let N 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 these are distributive morphisms, multiplication distributes over addition in N. Since the group of units is just the identity 1, all of the homomorphisms are central.

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 these are distributive morphisms, multiplication distributes over addition in N. Since the group of units is just the identity 1, all of the homomorphisms are central.

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 these are distributive morphisms, multiplication distributes over addition in N. Since the group of units is just the identity 1, all of the homomorphisms are central.

Subspaces of N are indicated by subspace morphisms. As always, 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. Besides 0 and 1, it is easy to find two classes of subspaces.

- **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}.

- **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 are not at all the same space.

- **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}.

- **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 `rem a a a` contains the same data as `min a a`, but are not at all the same space.

These two cases can be generalized as follows. For $p, q \geq 1$, let rem(p,q) be the morphism defined by

$$\text{while } n \geq p, q : \text{remove p from n} \tag{5}$$

All of these are also subspaces and this generalizes the modular sum and saturation cases since rem(p,p) is $n \mapsto n \bmod p$ 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

$$\text{rem}(p, q) \cdot \text{rem}(p', q') = \text{rem}(\text{lcm}(p, p'), \max(q, q')) \tag{6}$$

so that the rem subspaces are a semilattice. This completes the subspace analysis of N since 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 $x, y \in \mathbf{N}$... □

The most organic generalization of (**is a**) is (**is a b**), which we refer to as N_2. As above, let's consider subspaces.

- 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.lex, the data of N.lex can be written $a^m b^n$ for $m, n \geq 0$. As in the previous case, a homomorphism M of N.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 $\begin{pmatrix} m_{11} & m_{21} \\ m_{12} & m_{22} \end{pmatrix}$ in $\text{Mat}_{2\times 2}(\mathbf{N})$ where the action of M is exactly matrix multiplication. Thus, the data of N.lex are pairs of natural numbers the homomorphisms are $\text{Mat}_{2\times 2}(\mathbf{N})$. The automorphisms of N.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 the integers. 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.

Exploring further in this vein is certainly possible. It's worth keeping in mind that if spaces commute, their commutator is a new space. For instance in **is a b c d**, one can reduce to a subspace by (a b)=(b a)=(c d)=(d c)=1, this reduction commutes with a lexical sort and so the two can be composed to create $\mathbf{Z} \times \mathbf{Z}$ with homomorphisms $\mathrm{Mat}_{2\times 2}(\mathbf{N})$. More generally, any confluent convergent rewrite system on some chosen alphabet is automatically a space and could be subject to similar analysis.

Proof. .... □

7.1 Number Sequences

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

$$\mathbf{ap}\ (\mathbf{put}\ \mathbf{n}) \cdot N \cdot (\mathbf{get}\ \mathbf{n}) \tag{7}$$

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

T = (n:a a a) (n:) (n:a a) (n:a) (n:)

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 are therefore "more mathematical." Consider a morphism from

Some of the morphisms of $\mathbb{N}$ are "inherited" from $f \in$N. Define **inner**:F to be

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

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 (5) and (6) works for any space, not just for S and so if we define a "functor" **Seq** to be

$$(\text{put A}) \cdot \text{B} \cdot (\text{get A}) \tag{9}$$

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

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

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

$$\mathbb{N} \cdot \{\text{B (A : B)}\} \cdot \mathbb{N} \tag{11}$$

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

Summarize...

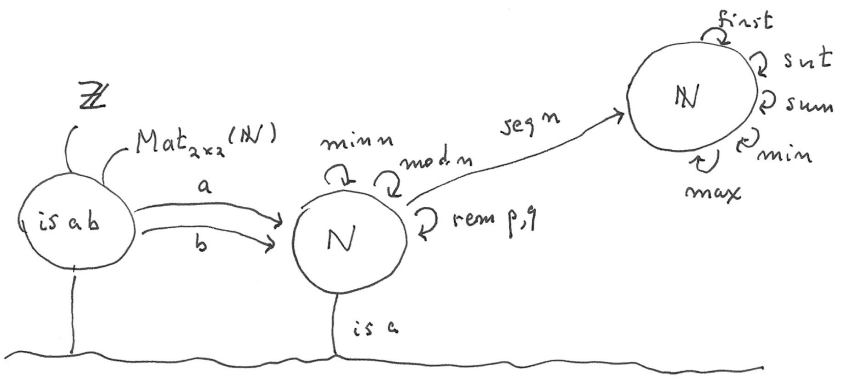

Figure 5: *Organic numbers.*

$f \cdot g$	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	-	TRUE	FALSE	NOT
FALSE	-	-	FALSE	FALSE
NOT	-	-	-	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 $f \oplus f = f$.

8 Boolean

The space **bool** is algebraic and positive, but not distributive. Since **bool** contains two data: () for *true* and (:) for *false*, there are only four morphisms: ID=**bool**, the constants TRUE={} and FALSE={(:)}, and one involution NOT=**bool**·**not**·. 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.

8.1 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

$$(\text{b :})\ (\text{b :})\ (\text{b : (:)})\ (\text{b :})\ (\text{b : (:)})\ (\text{b : (:)}) \tag{12}$$

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}. Since $\mathbb{L}_1$ has 27 morphisms and $\mathbb{L}_2$ already has $7^7 = 823,543$ morphisms, $\mathbb{L}_1$ is the last chance to be complete and explicit. For $\mathbb{L}_1$, we can specify a morphism $f \in \mathbb{L}_1$ by

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			

listing the values of f on 0,T,F in standard order, so 0TF is the identity morphism, etc. Figure X shows the 27 morphisms showing that even in a small space like $\mathbb{L}_1$, all the categories of morphisms appear in a nontrivial way.

Consider, at least, the inner morphisms of $\mathbb{L}_2$. These are the morphisms of the form

$$\mathbb{L}_2 \cdot (\textbf{put } b) \cdot F \cdot (\textbf{get } b) \cdot \mathbb{L}_2 \tag{13}$$

where F can be any data. Since (**get** b):X only depends on the number of (:) values in the b-atoms of X and since the morphism always produces exactly one b-atom, there are eight inner morphismsas shown in [**?**].

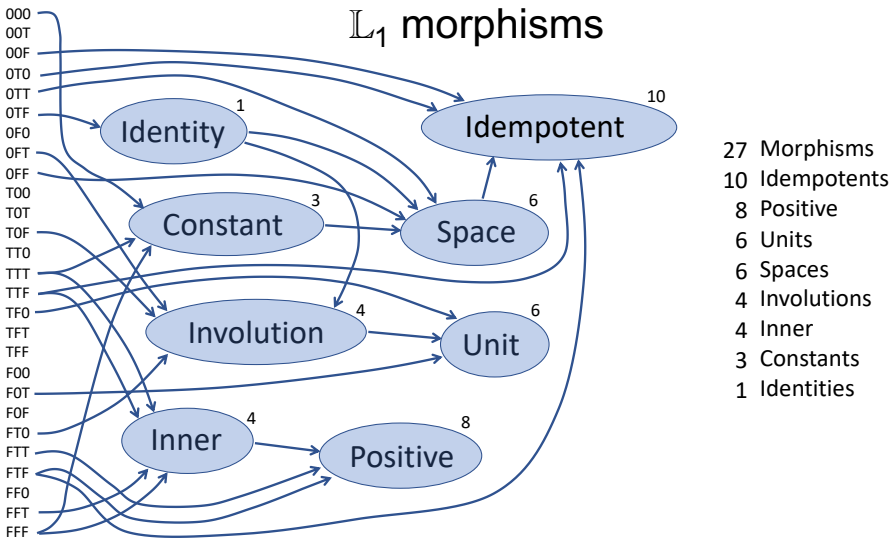

Figure 6: *Morphisms of* $\mathbb{L}_1$.

Figure 2 illustrates these definitions and demonstrates that composition of morphisms is associative. If $S \xrightarrow{F} T$ and $T \xrightarrow{G} U$, then $S \xrightarrow{G \cdot F} U$, where S, T and U are spaces (with structure). Figure 2 also illustrates how definitions can be adopted from Category Theory. For example, we say that spaces S and T are **isomorphic** if the diagram of Figure 2(c) commutes for some morphisms α and α', denoted by $S \overset{\alpha}{\cong} T$.

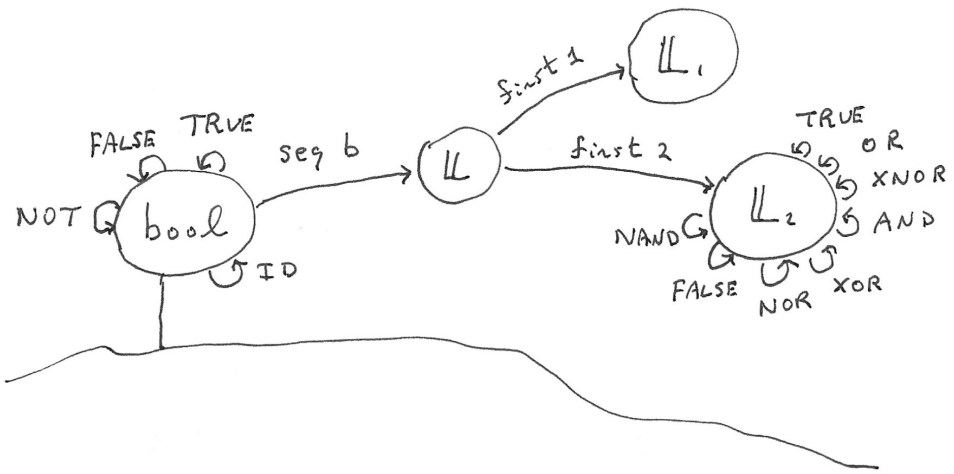

Figure 7: *A summary of the boolean spaces discussed in the text.* **bool** *is the most organic - the space of two boolean values () for true and (:) for false. The space* $\mathbb{L}$ *is sequences of such values with subspaces of at most one (*$\mathbb{L}_1$*) and at most two (*$\mathbb{L}_2$*) values.*

9 Outlook

10 Glossary

For example, starting with context δ, we can add more atoms of the form $\delta_a : c \mapsto c$:

- $\delta_{(:)}$: ((:) A : B) $\mapsto$ ((:) A : B), *single bit atoms where ((:):), ((:):(:)) are zero,one respectively.*
- $\delta_{((:):)}$: (((:):) A : B) $\mapsto$ (((:):) A : B), *eight bit bytes.*
- $\delta_{((:):(:))}$: (((:):(:)) A : B) $\mapsto$ (((:):(:)) A : B), *byte sequences.*

so that text strings or other data are atomic data within $\delta \cup \delta_{(:)} \cup \delta_{((:):)} \cup \delta_{((:):(:))}$. We can then proceed to add named definitions that 'get coda components':

- $\delta_{\mathbf{pass}}$: (**pass** A:B) $\mapsto$ B
- $\delta_{\mathbf{null}}$: (**null** A:B) $\mapsto$ ()
- $\delta_{\mathbf{right}}$: (**right** A:B) $\mapsto$ B
- $\delta_{\mathbf{arg}}$: (**arg** A:B) $\mapsto$ A

and can add definitions for simple combinatorics

- $\delta_{\mathbf{ap}}$: (**ap** A : b B) $\mapsto$ (A:b) (ap A:B), *apply A to each atom in B.*
- $\delta_{\mathbf{rev}}$: (**rev** A : B b) $\mapsto$ b (rev : B), *reverse the order of atoms in B.*

definitions computing the boolean value of data

- $\delta_{\mathbf{bool}}$: (**bool** A : B b) $\mapsto$ () if B is empty, (:) if B is atomic, *logical value of B.*

definitions making new definitions

- $\delta_{\mathbf{def}}$: (**def** a : B) $\mapsto$ Add δ_a to context if a is an unused invariant atom.

As explained in reference [1], a language is introduced merely as one more definition where the domain of the language context $\delta_{\{\}}$ is codas starting with a curly brace as in ({*language expression*} A : B).

- $\delta_{\mathbf{bool}}$: (**bool** A : B b) $\mapsto$ () if B is empty, (:) if B is atomic, *logical value of B.*

definitions making new definitions

- $\delta_{\mathbf{def}}$: (**def** a : B) $\mapsto$ Add δ_a to context if a is an unused invariant atom.

As explained in reference [1], a language is introduced merely as one more definition where the domain of the language context $\delta_{\{\}}$ is codas starting with a curly brace as in ({*language expression*} A : B).

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.