# Pure Data Foundations of Mathematics

Saul Youssef
Department of Physics
Boston University
youssef@bu.edu

April 22, 2023

**Abstract**

This is an abstract.

## 1 Introduction

## 2 Foundation

Any system of reasoning must necessarily have at least one foundational concept which is understood before a first definition. In our case, the foundational concept is the *finite sequence*. We assume that finite sequences and obvious operations on finite sequences are understood. On the other hand, logic, logical values and predicates (foundational in ZFC) and types (foundational in type theories) will be defined in terms of finite sequences. Taking finite sequences as understood, define 'data' and 'coda' by the following.

A *data* is a finite sequence of *codas*.

A *coda* is a pair of *data*.

Notationally speaking, if $A$ and $B$ are data, $A\ B$ denotes the concatenation of $A$ and $B$ as finite sequences and $A : B$ denotes the data consisting of the single coda formed by pairing $A$ and $B$. By the definition, for example, the empty sequence of codas () is data, and, therefore, so is the pairing of two empty sequences '(:)' and so is this sequence of three codas $(:)(:)((:) : (:))$. This sort of 'pure data made of nothing' is the central concept of coda. We shall see that both data in the ordinary sense (bits, bytes, language expressions) and mathematical concepts such as variables, functions, definitions, logical values, categories, morphisms, types and theorems naturally appear as varieties of pure data with simple definitions expressed in the small algebra of data with the concatenation and the colon operations. Notationally speaking, the colon operation binds from the right first and binds less strongly than concatenation, so that $A : B : C$ means $(A : (B : C))$ and $A : B\ C$ means $(A : (B\ C))$.

In coda, the answers to mathematical questions come from an equivalence relation =, which is defined by a partial function from codas to data called a *context*. Given a context $\delta$, equality is defined by

$A\ B = \delta(A)\ B = A\ \delta(B)$

$A : B = \delta(A) : B = A : \delta(B)$

for data $A$ and $B$ where the partial function $\delta$ has been extended to a function from data to data with identities. Conceptually, relations are assumed until $\delta$ is an identity on data. With a given context, data $A$ is *empty* if $A = ()$ and *invariant* if $\delta(A) = A$.

In coda, new definitions are added to a context as partial functions from codas to data. We use a convention to guarantee that each such partial function has it's own disjoint domain. Let the *domain* of a coda $A : B$ be the data consisting of the first coda in the sequence $A$ and the empty sequence if $A$ is empty. A partial function mapping the codas with a particular invariant domain $D$ to data is called a *definition.* Definitions can be added to a valid context if they do not clash.

**AXIOM OF DEFINITION.** *The empty context is valid. If $\delta$ is a valid context and $d$ is a definition, and if no coda is in the domain of both $\delta$ and $d$, then the union of $\delta$ and $d$ is a valid context.*

Assume an empty context, for instance. Then $(:)$ has invariant domain $()$ and, thus, the partial function $(:) \rightarrow (:)$ is a definition. Adding this definition to the empty context makes $(:)$ invariant in the new context. Continuing in this way, one can define invariant data representing a 0-bit, a 1-bit, finite bit sequences, finite byte sequences representing character strings as well as simple combinatoric operations as illustrated in figure 1.

Since coda has only one axiom, we will refer to the Axiom of Definition as just 'the axiom.' In the following sections, we will see why there is only one axiom. Coda already contains an internal logic which will make logic-related axioms unnecessary. Coda also contains an internal language defined, making language and syntax-related axioms unnecessary. Deduction rules are also not needed because deduction, proof and computation in coda is all determined by data equality. Each of these is a merely a data sequence $A_0 = A_1 = A_2 = \cdots = A_n$, deducing $A_n$ from $A_0$, computing $A_n$ starting with $A_0$ or proving that $A_0 = A_n$, depending on one's point of view. Unlike type systems with a Curry-Howard correspondence, proof and computation are the same thing in coda. Unless we indicate otherwise, 'data' will always refer to 'pure data' as defined in this section and a 'sequence' will always mean a finite sequence.

## 3 Logic

Within coda, mathematical objects are pure data. This means, roughly speaking, that mathematical questions will be appearing in the form:

Is data $A$ equal to data $B$?

Since context equality will also be a definition available in context, the answers to the such questions will also be data. But if a question like $A = B$ is data, it is natural to expect that everything about the answer should be encoded in the specific data $(= A : B)$ including the logical meaning of $A = B$. This suggests that 'logic' in should be defined as a coarse classification of data in general in which is stable under adding definitions to a context. A suitable classification of data is indeed available.

Given a particular context, data $A$ is *atomic* if it contains one or more invariant codas (called *atoms*). It is easy to see that the property of being empty or atomic is not lost when new definitions are added to a context. We say that atomic data is 'always atomic' and empty data is 'always empty.' This motivates the following definitions of logic within coda.

- data is **true** if it is empty.
- data is **false** if it is atomic.

- data is **undecided** otherwise.

Examples illustrate the flavor of this definition in a context where ‘foo’ has not been defined.

> **true data:** (), (pass:), (null:a b c), (null: (foo:bar)).
>
> **false data:** (:), a b c, pass : a b c, a b c (foo:bar).
>
> **undecided data:** (foo:bar), (pass:foo:bar), (foo:1 2 3).

Coda logic can be thought of as “not quite two valued” because undecided data like (foo:bar) may receive a value if more definitions are added to the current context. We clearly have

– True data is ‘always true.’

– False data is ‘always false.’

Undecided data, however, may change from undecided to true or false depending upon added definitions. In that sense, undecided data are “variables.” This suggests that coda logic is classical except that the undecided data are, effectively, logic valued variables. This is, however, not always true because some undecided data remains undecided no matter what definitions are added. Such **undecidable** data are the source of a Godel-related phenomena as demonstrated in Section X. Although the logic of coda is not the same as classical logic, we will argue that this is the “correct” logic for reasoning in general. It is also quite close to classical logic in the sense that there is a definition that corresponds to each of the familiar 16 binary operations of classical propositional logic which behave as expected classically. For instance, when data $A$ and $B$ are either true or false, the data (XOR $A$ : $B$) has the value () for *true* and (:) for *false* depending on the classical truth table of values for XOR. If either $A$ or $B$ are undecided, on the other hand, no definition applies to (XOR $A$ : $B$) and it “waits for $A$ and $B$ to be resolved in future definitions.”

## 4 Language

## 5 Proof and Computation

## 6 Spaces

## 7 Is Mathematics consistent?

## 8 Mathematical Machine Learning

## 9 Summary

## References

[1] R.T.Cox, Am.J.Phys. 14, 1 (1946).

[2] S.Youssef, Mod.Phys.Lett A9, 2571 (1994).

[3] S.Youssef, Phys.Lett. A204, 181(1995).