

PURE DATA FOUNDATIONS OF MATHEMATICS

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

April 28, 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.

- *Data* is a finite sequence of *codas*, and 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 δ , 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 δ has been extended to a function from data to data with identities. Conceptually, relations are assumed until δ 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 its 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 all of the codas with a particular invariant domain to data is called a *definition*. Definitions can be added to a valid context if they do not clash with previous definitions.

AXIOM OF DEFINITION. *The empty context is valid. If δ is a valid context and d is a definition, and if no coda is in the domain of both δ and d , then the union of δ 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 convenient combinatoric operations as illustrated in figure 1.$

Since coda has only one axiom, we 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, 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 $A_0 = A_1 = A_2 = \dots = A_n$, 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

The previous sections can be simply summarized. We have recursively defined “pure data” to be finite sequences of pairs of pure data called codas. Given such data, any partial function from coda to data qualifies as a “definition.” Under the conditions of the Axiom of Definition, definitions can be added to a “context” which defines an equality on data and meaning within the system. These simple definitions and, for instance, the combinatorial examples in figure 1 are meant to be sufficiently understood based on the foundational concept of the **finite sequence** without the need for logical axioms or logical values. We presume, for instance, that reversing the order of a finite sequence is understood without making a choice between, say, classical and intuitionistic logic.

Classical logic can be thought of as establishing at least two things: 1) defining how to reason correctly and 2) assigning meaning to “propositions.” In this section, we will propose a natural pure data oriented alternative.

In the framework of coda, all mathematical objects (values, functions, variables, morphisms, theorems, ...) are meant to be merely pure data. This means that “propositions” will appear in the form:

Is data A equal to data B ?

Since equality itself is available as a definition, then this question is also the specific data $A = B$ or $(= A : B)$. But if a question like “Is data A equal to data B ?” is data, it is natural to expect that the answer to the question should be encoded in the specific data $(= A : B)$. This suggests that “logic” in coda should appear as a natural coarse classification of data in general. The classification should match our intuition and ordinary reasoning and should be stable under adding new definitions to a given context. A suitable classification is indeed available.

Given a particular context, data A is said to be *atomic* if one or more of the codas in it’s sequence is invariant. Note that since being empty and being atomic are disjoint categories which are preserved when adding new definitions to a context, this classification is an interesting candidate:

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

although true data is “always true” and false data is “always false,” [always], undecided data may become true or false with added definitions. In this sense, undecided data are like “variables” which may receive a true or false value depending on future choices. This is not quite the correct picture, however, because some undecided data actually remain undecided independent of any future added definitions. Such *undecidable* data occurs, for example, with the Godel phenomenon as we shall see in section Y. The fact that coda logic is “almost” two-valued logic is from the duality between *true* data $()$ and *false* data $(:)$ visible in any display of pure data. Although proposing to change something as fundamental as logic can be disorienting, one soon realizes that the situation is still quite familiar and that the seemingly odd “undecided” category works perfectly. Examples illustrate the flavor of these definitions assuming a context where “foo” has not been defined:

true data: $()$, $(\text{pass}:)$, $\text{null} : a\ b\ c$, null:foo:bar , $(\text{and}:)$, $(\text{or } a:)$, $(\text{xor } :a)$,

false data: $(:)$, $a\ b\ c$, $\text{first } 3 : a\ b\ (\text{foo:bar})$, $(\text{and } a:b)$, $(\text{or } a:b)$,

undecided data: foo:bar , pass:foo:bar , $\text{last} : a\ b\ (\text{foo:bar})$.

In the above, “and”, “or”, and “xor” are familiar binary operations from classical logic in the cases where their arguments are true or false in the coda sense. For instance, if data A and B are both either true or false then $(\text{xor } A : B)$ is $()$ for *true* and $(:)$ for *false* according to the standard truth table for the **xor** operation. On the other hand, if either A or B are undecided, the data $(\text{xor } A : B)$ has no definitions that apply. In effect, $(\text{xor } A : B)$ “waits” until both A and B are defined enough to have logical values. This is an example where $(\text{xor } A : B)$ acts as a logical expression with “variables” A and B .

Undefined data may occur even when data does not explicitly contain an undefined coda such as (foo:bar) above. For example the natural numbers in coda is the data $(\text{nat}:0)$ because a definition $(\text{nat} : n) \rightarrow n\ (\text{nat} : n + 1)$ means that $(\text{nat}:0) = 0\ 1\ 2\ 3\ 4\ 5\ (\text{nat}:6)$, etc. The sum of the first 1000 natural numbers is perfectly well defined and is computed by “sum $n : \text{first } 1000 : \text{nat} : 0$ ”. What happens, however, if we decided to sum all natural numbers via “sum $n : \text{nat} : 0$ ”? The answer is that this data remains undecided no matter how much the expression is evaluated. Here, the undecided logical category is, quite nicely and naturally, the logical value of data that is not defined.

Classical logic can be thought of as establishing at least two things: 1) defining how to reason correctly and 2) assigning logical meaning to “propositions.” Within coda, 1) reasoning is correct if and only if it is application of a definition in context and 2) logical meaning is defined by data

being either empty, atomic or neither. Rather than propositions being foundational undefined entities, propositions in coda are just data and their logical values are also just data. There is an understandable and perhaps even unpleasant disorientation caused by attempting to re-defining something as fundamental as classical logic. We have found that this disorientation is only temporary once one realizes that the familiar binary logical operations remain, and the seemingly odd class of “undecided” data fits in perfectly as “logic valued variables” and as “the answers to questions that have no answers.” Confidence may be gained in section X where we examine Godel phenomena and directly compute some paradoxes.

4 Language

Both predicate logic with ZFC or with dependent type theory are formal languages. They have alphabets with special symbols and axiomatic syntax rules which distinguish meaningful sentences from nonsense sentences such as “ $xyxy\forall\exists\exists xx\forall$ ”. The situation with coda is much simpler because we can define the language itself as an ordinary definition, added to a context via the Axiom of Definition just like all other definitions. The basic idea behind the language definition is to give textual control over the concatenation and colon operations. Thus, if x and y are language expressions, partial functions

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

allow specification of the two operations. The language literals “A” and “B” allow specification of left and right components of a generic coda $(A : B)$ via

- $(\{A\} A : B) \rightarrow A$
- $(\{B\} A : B) \rightarrow B$

These are partial functions fused into one language partial function by enforcing an order which also defines the precedence of operations. In the actual language, there are just a few more partial functions defining grouping with parenthesis, string literals with angle brackets, removing extraneous spaces and adding a bit of syntactic sugar so that $A = B$ in the language is interpreted as $(= A : B)$ and $A * B : X$ in the language is interpreted as $A : B : X$ for any data X and $X?$ is interpreted as $(? : X)$. Some features of the language are unusual.

- Because the language is a definition like any other, language expressions in curly braces freely mix with all other data during evaluation.
- The language has the property that *any* sequence of bytes is a valid language expression.
- The mapping from finite byte sequence s to the data $(\{s\} :)$ is onto.

The language is intentionally minimalist and tiny enough for the compiler and parser to be easily read and understood by humans [github].

- 5 Proof and Computation
- 6 Spaces
- 7 Mathematical Machine Learning
- 8 Is Mathematics Consistent?
- 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).