arXiv:2305.05507v1 [cs.DC] 9 May 2023

数学的纯数据基础

Saul Youssef 波
士顿大学物理系

2025年2月17日

摘要

我们提出一种数学公理化基础，以有限序列 作为基础概念，而非像集合论那样以逻辑和集合为基础，也非像依赖类型理论那样以类型 为基础。有限序列引出纯数据 这一概念，并以此表示所有数学对象。作为一个公理系统，该基础仅包含一条公理，用于定义何为有效定义。利用这条公理，可以定义一种内部的真/假/未定值逻辑和一种内部语言，从而无需再引入与逻辑和语言相关的公理。有效证明和有效计算均通过纯数据的相等性来定义。纯数据的代数结构引出了丰富的空间 和态射 理论，它们在现代数学中发挥着类似于范畴论的作用。作为应用，我们探讨数学机器学习、数学的一致性，以及由 G¨odel、贝里、柯里和 Yablo 引发的悖论。

1 基础

任何推理系统都必然具有一些基础概念，这些概念在第一个定义之前就已被直观理解。例如，在类型理论[1, 2, 3, 4], 中，没有“类型”的定义；而在策梅洛-弗兰克尔集合论[5] （ZFC）中，没有“真”“假”或“集合”的定义。在我们的情形中，我们有一个基础性的未定义概念：有限序列。我们假定有限序列以及关于有限序列的显然陈述都是可以理解的。另一方面，逻辑、逻辑值以及命题的类比物，将作为“数据”的不同形式出现。数据定义为有限序列，具体如下：

- 数据 是由尾码组成的有限序列，而$coda$ 是由数据组成的一对。

如果 A 和 B 是数据，就有一种自然的数据代数，其中 $A\ B$ 是由数据 A和数据 B 连接而成的，A则是：将 A 和 B 配对成一个 coda 后形成的数据。例如，根据定义，尾码的空序列（写作‘()’）是数据。因此，两个空序列的配对（写作‘(:)’）也是数据，所以，以下由三个尾码构成的序列 (:(:)(:)(:)) (:(:)) ((:):((:):)) 也是数据。我们称之为“纯数据”，因为它是“由有限个空无构成的序列制成的”数据。通常意义上的数据（比特、字节、语言表达式）和数学概念，都自然地表现为纯数据的不同形式，并可用数据代数中表达的简单定义来描述。从符号表示上说，冒号运算先从右侧结合，且其结合力弱于连接运算，因此 $A : B : C$ 表示 $(A : (B : C))$，而 $A : B\ C$ 表示 $(A: (B\ C))$。我们用“coda”一词既指按定义将两个数据配对的操作，也指我们正在定义的公理系统[6]。

在 coda 中，数学问题的答案由等价关系‘≒’确定，该关系通过一个从尾码到数据的偏函数定义，这个偏函数称为“上下文”。给定一个上下文 δ，等式定义如下：

$$A\ B = \delta(A)\ B = A\ \delta(B) \tag{1}$$

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

对于数据 A 和 B，其中偏函数 δ 已扩展为一个从数据到带单位元的数据的函数。从概念上说，δ 被迫具有与数据上的单位函数相同的性质。在给定的上下文中，数据 A 在 $A = ()$ 时是空的，在 $\delta(A) = A$时是不变的。

在 coda 中，新定义作为从尾码到数据的偏函数添加到上下文中。我们采用一种约定，以保证每个这样的偏函数都有各自不相交的定义域。令 coda $A: B$ 的定义域为由序列 A中的第一个 coda 组成的数据；如果 A 为空，则定义域为空序列。将尾码映射到数据、且具有特定不变定义域的偏函数称为定义。如果定义之间不发生冲突，则可以将其添加到“有效”上下文中，以下内容保证了这一点。

定义公理。空上下文是有效的。如果 δ 是有效上下文，d 是一个定义，并且没有 *coda* 同时属于 δ 和 d的定义域，那么 δ 和 d 的并集是有效上下文。

作为一个公理系统，coda 到此已完备。这与例如 ZFC 形成鲜明对比：即便假设采用谓词逻辑，它仍有十条公理[5]。这种简化的原因将在接下来的章节中变得清晰。第 3 节将展示，coda 包含一种隐含于其既定结构中的内部逻辑。我们采用这种逻辑在 coda 内进行推理，因此不需要与命题逻辑或谓词逻辑相关的公理。第 4 节将展示，coda 还包含一种内部语言，它像其他定义一样通过定义引入。这种语言有一个不同寻常的性质：所有字节序列都是有效语法，因此也不需要额外的语言公理和语言语法公理。我们也不需要用于规定有效“演绎规则”的公理，因为在 coda 中，演绎、证明和计算都由上述数据相等性决定。从不同角度看，它们都只是一个数据序列 $A_0 = A_1 = A_2 = \cdots = A_n$：从$A_0$演绎出 A_n、以 A_0 为起点计算 A_n，或证明 $A_0 = A_n$。与具有柯里–霍华德对应关系的类型系统不同，在 coda 中，证明和计算是一回事。

在接下来的章节中，“数据”和“纯数据”是可以互换的术语；除非另有说明，“序列”始终指有限序列。由于 coda 只有一条公理，我们可以将定义公理简称为“公理”。

2 原子

给定固定的不变数据 D，其长度小于或等于 1，部分单位函数$i_D : (DA: B) \mapsto (DA: B)$ 是一个定义。根据公理，如果当前上下文 δ的定义域与 i_D的定义域不相交，就可以将 i_D 添加到当前上下文中。在这种情况下，尾部（$DA: B$）称为原子，因为当前或未来的任何定义都无法从数据序列中移除（$DA : B$）。序列中至少含有一个原子的数据称为原子数据，而等于 () 的数据称为空数据。方程 (1) 和 (2) 表明，空数据不可能等于原子数据；公理则表明，无论未来如何定义，这一事实始终成立。我们用“始终”和“永不”来指称不受未来定义影响的性质，因此原子数据“始终”是原子的，空数据“始终”是空的，而原子数据与空数据“永不”相等。

域	描述	作用
pass	原样传递输入	（传递 A:B）$\mapsto B$
null	对任意输入均为空数据	(空值 A:B)$\mapsto$()
rev	反转数据的顺序	(反转： B)$\mapsto$ ()，如果 B 为空 (反转 : b)$\mapsto$ b，如果 b 是一个原子 (反转： B C)$\mapsto$ (反转:B) (反转:A)
if	条件 B	（如果 A:B）$\mapsto B$，如果 B 为空 （如果 A:B）$\mapsto$ ()，如果 B 是原子的。
ap	将参数 A 应用于每个输入	(ap A： B)$\mapsto$ ()，如果 B 为空 (ap A : b)$\mapsto$ A:b，如果 b 是一个原子 (ap A： B C)$\mapsto$ (ap A:B) (ap A:C)
nat	自然数	(nat: n)$\mapsto$ n (nat: $n+1$)
=	相等性	(= A : B) $\mapsto$ ()，如果 $A = B$，则为原子，如果 $A \neq B$
def	创建一个新定义	(def A:B)$\mapsto$ 添加定义（A A':B'）$\mapsto B$。

表 1：*coda* 中的典型定义。每个定义都是一个从尾码到数据的偏函数，其定义域是指定的尾码。列出多个作用时，总作用由左侧模式适用的第一个作用定义。

正如我们将在第 3 节看到的，对空数据和原子数据的分类是 coda 内部逻辑的基础。原子也是定义永久数据（如位、字节和字节串）的机制。从一个空上下文开始，可以参考上面的纯数据示例提出第一个定义；在这些示例中，纯数据似乎是“由 () 和 (:) 构成的”。显然，如果我们将 (:) 定义为 $\mapsto$ ()，所有纯数据都会坍缩为 ()。这意味着应当将 (:) 映射到 (:)。由于 (:) 的不变定义域是 ()，我们可以为任何定义域为 () 的 coda 定义 (:B)$\mapsto$(:B)。这样，(:) 既是原子又是不变的，因此可以用作新定义的定义域。依此继续，我们将不变的“0 位”定义为 ((:):)，将不变的“1 位”定义为 ((:):(:))。定义域为 0 位的原子通常称为位序列，定义域为 1 位的原子通常称为字节序列。这样，普通字节序列就可以用作新定义的不变名称。例如，定义“pass”将所有定义域 (pass A: B) 映射到 B。根据具体情形，有时将“pass”视为一条“命令”会更方便，把 B 看作“输入”，把 A 看作“参数”。或者，也可以把“pass”视为一个二元运算符，以数据 A 和数据 B 为输入，产生数据 (pass A:B)。表 1 和表 2 展示了典型的定义示例。除了大约 50 个简单的预定义定义外，coda 还包含 (def A:B) 的定义，使人们能够根据定义公理，在 A 不变且尚未在上下文中使用的前提下，向上下文添加新定义。一类重要且典型的定义是应用，例如表 2 中的“ap”。它们是对序列进行的简单组合运算。由于有限序列在基础层面上已有明确理解，我们认为表 2 中的说明已足够。软件[6]还提供了进一步的说明和内置示例。

3 逻辑

在 coda 的框架中，具有数学意义的对象以纯数据表示。这意味着数学问题通常会呈现为以下形式

- 数据 A 是否等于数据 B?

示意性 coda	$\mapsto$	示意性结果
ap $A{:}b_1\ b_2\ b_3$	$\mapsto$	$(A{:}b_1)\ (A{:}b_2)\ (A{:}b_3)$
app $a_1\ a_2\ a_3{:}B$	$\mapsto$	$(a_1{:}B)\ (a_2{:}B)\ (a_3{:}B)$
ap2 $a\ a_1\ a_2\ a_3{:}B$	$\mapsto$	$(a\ a_1{:}b_1)\ (a\ a_2{:}b_2)\ (a\ a_3{:}b_3)$
aps $A{:}b_1\ b_2\ b_3\ b_4\ b_5\ b_6$	$\mapsto$	$(A\ b_1{:}(A\ b_2{:}(A\ b_3{:}b_4(A\ b_5{:}b_6)))$
apif $A{:}b_1\ b_2 \ldots$	$\mapsto$	b_1（如果（$A{:}b_1$）为空，则为 ...） b_2（如果（$A{:}b_2$）为空，则为 ...）...

表 2： 内置定义通常是对作为有限序列的数据进行组合运算。*“ap”*系列是将*“*参数*”A* 应用于*“*输入*”B* 这一概念的不同变体，具体应用方式各不相同。

$True$	(), (**通过**：)，**空值**：a b c，（**且**：），（**或** a：），（**异或**：a）
假	a b c，**第一个** 3：a b（foo：bar），（**且** a：b），（**或** a：b）
未决	foo：bar，**通过**：foo：bar，**最后**：a b（foo：bar）

表 3： 在显示定义域的上下文中，以粗体文本表示真/假/未决数据的示例。

由于数据相等性本身可以作为定义使用，因此这类问题的答案应当可以从具体数据（$= A{:}\ \ B$）中推导出来。这表明，coda 中的“逻辑”应当是对一般数据进行适当粗粒度分类的一种方式。当然，人们会期待，某些特定数据的逻辑分类不应因通过公理添加新定义而发生变化。空数据和原子数据这两个互不相交的范畴已经符合这一标准，这启发我们作出如下定义：

- 数据为空时为 真，数据为原子数据时为 假，否则为 未决。

表 3 展示了这种分类的示例，其中包括经典逻辑中常见的二元运算。例如，如果数据 A 和 B 都是逻辑值真或假，那么根据经典逻辑中异或运算的标准真值表，(xor $A{:}\ \ B$) 在 真时为 ()，在 假时为 (:)。另一方面，如果 A 或 B 中任一个为未决，那么数据 (xor $A : B$) 就没有适用的定义。实际上，(xor $A : \ \ B$) 会“等到 A 和 B 都变得足够明确”后，才具有逻辑值。我们提供了一个“bool”定义，因此当 A 为真时，(bool:A) 为 ()；当 A 为假时为 (:)；否则为未决。虽然真数据始终为真，假数据始终为假，但未决数据可能会随着新增定义而变为真或假。这表明，我们正在定义一种二值逻辑，其中未决数据在某种意义上就是变量。这并不是完整的情况，因为有些未决数据无论添加什么定义都必须保持未决。这种不可判定数据在多种情况下都很有用，正如第 7 节和第 8 节所述。

重新定义经典逻辑这样基础的概念，可能会让人感到困惑，这种困惑是可以理解的。一旦意识到，熟悉的二元逻辑运算依然存在，而看似古怪的“未决”数据完全可以作为“逻辑值变量”以及“没有答案时对问题的回答”，这种困惑就只是暂时的。第 7 节和第 8 节将探讨哥德尔现象，并用广为人知的悖论检验所提出的逻辑，届时读者或许会更有信心。

4 语言

ZFC 谓词逻辑和依赖类型理论都是形式语言。它们都有各自的字母表，其中包含特殊符号，也都有公理化语法规则，用来区分有意义的句子

与无意义的句子，例如 **xx**∀**yy**∃∃。就 coda 而言，情况比较简单，因为这种语言看起来不过是像其他定义一样的又一个定义。基本思路是对数据代数中的运算 $A\ B$ 和 $A:\ B$ 进行文本控制。因此，如果 x 和 y 是语言表达式，偏函数

$$(\{x\ \ y\}\ A:B) \mapsto (\{x\}\ A:B)\ (\{y\}\ A:B) \tag{3}$$

$$(\{x:y\}\ A:B) \mapsto (\{x\}\ A:B):(\{y\}\ A:B) \tag{4}$$

允许指定这两种运算。语言字面量“A”和“B”

$$(\{A\}A:B) \mapsto A \tag{5}$$

$$(\{B\}A:B) \mapsto B \tag{6}$$

允许通过语言规范指定 coda 的左侧和右侧组件。该语言还有其他内容，但并不多。其他偏函数定义了带括号的分组操作，允许使用尖括号括起字符串字面量、移除多余的空格，并添加少量语法糖，使 $A = B$ 被解释为 $(= A:\ B)$，$A * B:\ X$ 被解释为 $A:\ B:\ X$，而 X? 被解释为 $(?\ X)$，从而使 X? 的行为类似于一个“变量”。所有这些偏函数都融合为一个单独的偏函数，作用于所有以花括号内文本开头的尾码。融合的顺序决定了各项操作的优先级。图 1 展示了该语言在 coda 笔记本[6]中的使用示例。

```
first 2 : a b c d
```

```
a b
```

```
step : first 2 : a b c d
```

```
[0] (({first 2 }:):({ a b c d}:))
[1] (({first 2}:):({a b c d}:))
[2] (({first}:) ({2}:):({a}:) ({b c d}:))
[3] a (first 1:({b}:) ({c d}:))
[4] a b (first 0:({c}:) ({d}:))
[5] a b
```

```
def first3 : {first 3 : B}
first3 : a b c d
```

```
a b c
```

图 1： 摘自一次 *coda* 笔记本会话。顶部单元格展示了*“first”*命令如何从包含四个尾码的*“input”*序列中选取前两个元素。中间单元格展示了使用*“step”*执行相同操作的方式，逐步显示对数据“{ *first 2 : a b c d}:”*进行求值的各个步骤，该数据是语言表达式*“first 2 : a b c d”*的 *coda* 定义。由于语言与其他定义一样，也是在上下文中给出的定义，因此，随着求值的进行，语言操作会与其他定义交织在一起。第三个单元格展示了一个在语言内部添加的典型定义。

这种语言是对数据代数的一种刻意设计的极简封装。只有字符‘**():{}**=*/ ’具有语言意义。它没有保留关键字，也没有用于.....的特殊语法。

变量、函数、类或异常。任何有限字节序列 s 都可以通过 $s \mapsto (\{s\}:)$“编译”为纯数据。这种语言不同寻常之处在于，任何有限字节序列都是有效的语言表达式。这意味着我们不必指定字母表，甚至不必定义有效和无效的语法。不存在所谓的无效语法。编译器和解析器的源代码足够精简，人类可以轻松阅读和理解 [6]。

5 证明与计算

如第 2 节所述，coda 既是一个形式系统，也是一个计算系统。作为形式系统，其“句子”就是纯数据，而有效的“演绎规则”是：根据方程（1）和（2），上下文 δ 可以应用于任何数据。给定一些起始数据 A_0，例如，可以将任意选定的 coda c （位于 A_0 内）替换为 $\delta(c)$。可以按需重复此操作，从而得到某个序列 A_0，A_1，……，A_n。该序列取决于所作的选择，但（1）和（2）保证 $A_0 = A_1 = \cdots = A_n$ 对任何选择都成立。在这一框架中，证明与计算的区别仅在于应用δ的策略，即在何处、何时应用。上述应用 δ 的过程称为求值。本文的大多数结果都是通过一种非常简单的求值策略计算得到的：尽可能在所有位置应用 δ，然后重复此过程，直到序列 A_0，A_1，…… 出现重复，或达到选定的最大长度。图 1 和图 2 展示了这种策略的示例。在图 1 中，序列收敛到一个原子答案。当然，情况并非总是如此，如图 2 中自然数的求值所示。请注意，在图 2 中计算“最后一个自然数”（last:nat:0）时发生了什么。计算自然数的逆序中的第一个自然数（first:rev:nat:0），或所有自然数的和（sum n:nat:0），也会得到类似的结果。这些情况都会产生未决数据，而这些数据也是不可判定的。不可判定数据发挥着不可或缺的作用：它是“无解时问题的答案”。

这里获得的结果大多源自上述简单的求值策略。我们预计，采用更精细的方法将会带来诸多收益，例如优化 δ 的应用位置和时机，并将同样有效的 $\delta(A) \mapsto A$ 移动也纳入其中，同时考虑 $A \mapsto \delta(A)$。截至本文撰写之时，这仍是一个基本未经探索的领域。Coda 具有一些独特而引人注目的计算特性。由于 δ 是一个偏函数，根据定义，它不可能发生“循环”。这意味着，求值序列 A_0，A_1，……可以无限持续下去，但序列中的每一步都保证能够终止。

6 空间

由于数据代数是所有纯数据共有的一种结构，我们可能会认为，这种代数扮演着类似于范畴论在现代数学[12]中所扮演的角色。这里，我们将通过几个步骤来发展这一想法，首先着眼于指定一些我们感兴趣的数据集合（称为“空间”）及其相关“态射”的最自然方式。首先，给出几个初步定义。

- 数据 A 是幂等的，条件是 幂等的 if $A : A : X = A : X$ 对所有 X都成立。
- 数据 A 是分配的，条件是 $A : XY = (A : X)(A : Y)$ 对所有 X, Y都成立。
- 数据 A 是阿贝尔的，条件是 $A : XY = A : YX$ 对所有 X, Y都成立。

考虑到数据代数，任何固定的数据 A 都定义了一个集合

$$A : X \tag{7}$$

```
step : nat : 0
```

```
[ 0] (({nat }:):({ 0}:))
[ 1] (({nat}:):({0}:))
[ 2] 0 (nat:1)
[ 3] 0 1 (nat:2)
[ 4] 0 1 2 (nat:3)
[ 5] 0 1 2 3 (nat:4)
[ 6] 0 1 2 3 4 (nat:5)
[ 7] 0 1 2 3 4 5 (nat:6)
[ 8] 0 1 2 3 4 5 6 (nat:7)
[ 9] 0 1 2 3 4 5 6 7 (nat:8)
[10] 0 1 2 3 4 5 6 7 8 (nat:9)
```

```
rev : nat : 0
```

```
(rev:(nat:6)) (rev:5) 4 3 2 1 0
```

```
last : nat : 0
```

```
(last 1:0 1 2 3 4 (nat:5))
```

图 2: 摘自一次 *coda* 笔记本会话*[6]*。每个单元格都采用正文中解释的简单策略进行求值，最多执行十个步骤。顶部单元格使用 *'step :'*，以显示策略中的每一步，展示 *(nat:0)* 如何以惰性方式表示自然数。请注意，在早期步骤中，语言元素与其他数据混合在一起。下面的两个单元格表明，如果计算某个没有答案的内容，例如最后一个自然数 *(last:nat:0)*，*coda* 会通过提供不可判定数据作为答案来作出响应。

其中 X 可以是任意数据。我们称 $(A:\ X)$ 属于 A。自然地，如果 $(A:\ X)$ 和 $(A:\ Y)$ 属于 A，我们希望 $(A:\ X)$ $(A:\ Y)$ 也属于 A。如果我们要求

$$A : (A : X)\ (A : Y) = A : X\ Y \tag{8}$$

对所有数据 X 和 Y均成立，那么我们就得到了空间的定义。给定两个空间 A 和 B，如果分配数据 F 满足

$$F : A : X = B : F : X, \tag{9}$$

那么 F就是从 A到 B的映射，其中 F的分配性保证了 F保持序列。在这种情况下，我们称 F 为从空间 A 到空间 B的态射 。如果 F 是从空间 A 到 A的态射，我们就称 F 为“ A的态射”。宽泛地使用纯数据相等这一概念，如果 $F * A = B * F$，且其中 $A * B:\ X$定义为 $A:\ B:X$，我们就称 F为从 A到 B的态射。每个空间都有一个特殊元素 $(A:)$，我们称它为该空间的中性数据或中性元素 。如果 A 是一个空间，并且空间 B 以如下意义中和每个 $A:X:$

$$A : (A : X)\ (B : X) = A : (B : X)\ (A : X) = (A :) \tag{10}$$

然后我们称 B A的反空间。分配的空间称为一种类型。数据 F，其中 $F:X$ 是一个空间，称为一个函子。

表 4 显示 空间及其对应态射的示例。可以把它想成 o f

空间	作用	Data	$F * Space = Space * F$
pass	A↦A	所有数据	F 可分配
null	A↦ 0	仅限 0	F 使得 F:X=0
第一个	a A↦ a	单原子	F(a)，其中 a 为原子
bool	A↦0 *or* (:)	0 或 (:)	保持逻辑的 F
类型 n	A↦ (n:3) (n:5)	自然数	（类型 n）*F*（类型 n），对于任意 F
n 的和	（n:5） （n:3）↦ （n:8）	自然数	$F\ （n_1 + n_2）\ = F\ （n_1）\ + F\ （n_2）$
n 的乘积	(n:5) (n:3)↦ (n:15)	自然数	$F(n_1 * n_2) = F(n_1) * F(n_2)$
排序 n	(n:5) (n:3)↦ (n:3) (n:5)	自然数	$n_1 \leq n_2 \implies F(n_1) \leq F(n_2)$
集合 n	(n:5) (n:3)↦ 等价类	等价类	保持类的 F
n 的和	$F_1\ F_2 \ldots F_n \mapsto F_1 * \cdots * F_n$	线性映射 $n \mapsto n$	函子性的
Sum	$F_1\ F_2 \ldots F_n \mapsto F_1 * \cdots * F_n$	线性映射	函子性的
空间	$A \mapsto S_1 S_2 \ldots S_n$	空间	空间 $* F *$ 空间 适用于任何 F
Mor	$F_1\ F_2 \ldots F_n \mapsto F_1 * \cdots * F_n$	态射	函子性的

表 4： 空间及其态射的示例。所列空间中，只有 *pass*、空值、类型 *n* 和 *Space* 是类型。中性元素分别是：*sum n* 的 *(n:0)*、*prod n* 的 *(n:1)*、*set n* 的空等价类，以及 *Sum n*、*Sum* 和 *Mor* 的 *pass*。其他所有空间的中性元素都是 *()*。

将一个空间视为执行两项操作：1）选择或构造感兴趣的数据；2）定义如何组合所选数据的有限序列，以在该空间中生成新数据。(type n) 的情况具有典型性。空间 (type n) 尽可能从其“输入”中构造自然数，并丢弃任何无法转换为自然数的内容；这些自然数以诸如 (n:5) 这样的 coda 表示。空间 (type n) 是分配的，因此也是幂等的，所以其输入中的自然数会得到保留。由于 (type n) 是分配的，(type n) 的态射就是从自然数到自然数的、保持序列的任意函数。数据 (sum n) 也是一个空间，它选择与 (type n) 相同的自然数序列，但通过加法组合这些序列，因此 sum n : (n:3) (n:5) 等于 (n:8)。请注意，一个空间决定了它的态射。(sum n) 的态射必须满足 (sum n) : F : $X = F$: (sum n) : X ，这意味着 F 必须是线性的。另一方面，(sort n) 的态射必须满足 (sort n) : F : $X = F$: (sort n) : X ，这意味着 (sort n) 的态射必须是保序函数。

创建具有不同附加性质的空间抽象理论似乎很有前景。例如，

- 带有反空间的空间是一个纯群。

这个名称是合理的，因为集合 $\{G : X | \mathbf{X}\ is\ pure\ data\}$ 在通常意义下构成一个群，其中复合定义为($G : X$) $\times$ ($G : Y$) $\to$ G : ($G : X$) ($G : Y$)，满足结合律，使(G:)成为单位元；并且(G^{-1}:X)是(G:X)的逆元，前提是 G^{-1} 是所承诺的 G的反空间。

一种关于空间的有趣可能性，可以称为“数学机器学习”。图 3 展示了一个简单示例：我们设想自己对两个不同的数学对象集合产生了兴趣，而这两个集合都只由原子 (:) 的有限序列组成。两个集合的区别在于，一个集合中每个序列都含有奇数个原子，另一个集合中每个序列都含有偶数个原子。我们通过搜索数据 A 来尝试将这两个样本分类，也就是说， A: X 对于这两个集合具有不同的逻辑值。第一次搜索从 14,098 个数据中找到了三个解，其中包括数据“aps not”。结果表明，对于偶数个 (:) 的情况，它为真；对于奇数个 (:) 的情况，它为假。这个解的一些特征颇为有趣。

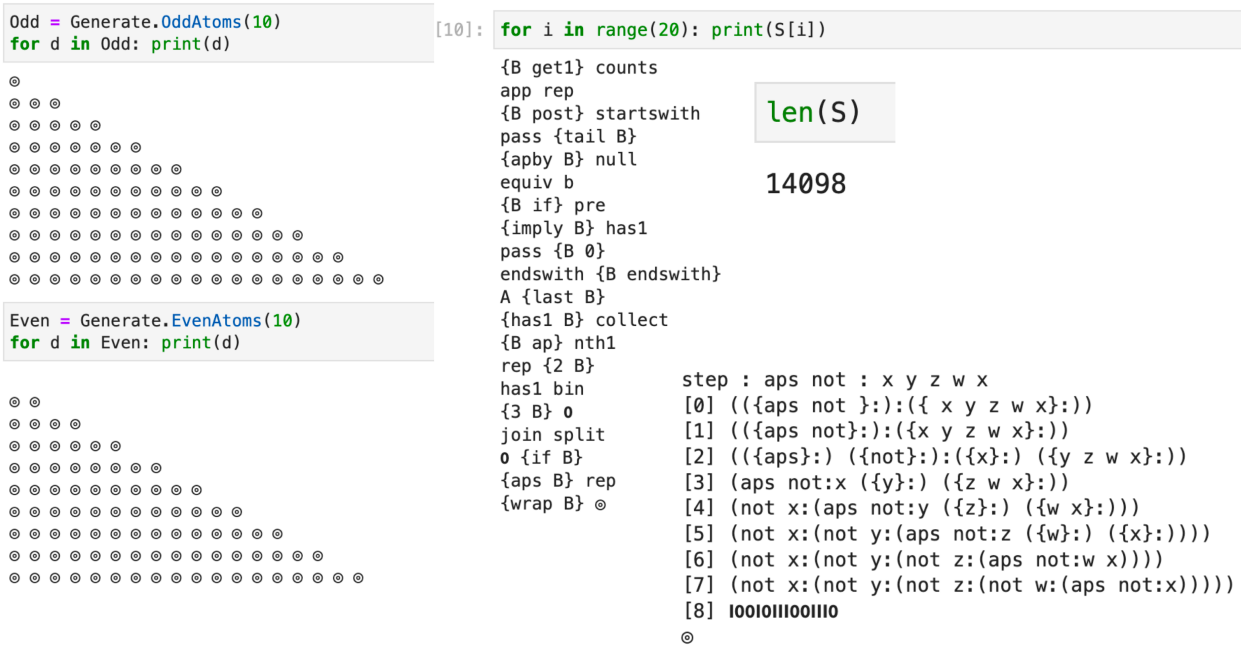


图 3： 摘自一场 *coda* 笔记本会话，展示了一个初步示例：尝试对一些我们感兴趣的数学对象进行分类。在这个例子中，我们搜索能够区分两组数据的数据：一组包含由奇数个 *(:)* 原子组成的数据（以内部带有一个小圆圈的圆圈表示），另一组则包含由偶数个 *(:)* 原子组成的数据。对 *14,098* 个任意生成的数据进行搜索（其中展示了部分数据），得到三个成功结果，包括正文中所述的“*aps not*”。

- 这个解法很巧妙。它将组合算子 ‘aps’ 与逻辑算子 ‘not’ 结合起来，作者在搜索之前并未想到这种方式。
- 这个解法具有普遍性，因为 ‘aps not’ 在所有情况下都会选出包含偶数个原子的 数据，而不仅仅是在应用于由 (:) 原子构成的序列时如此。
- 尽管‘aps not’不完全是空间，但稍作修改后的 ‘bool∗(apsnot)’ 按照上述含义是一个空间。
- 空间 ‘bool∗(aps not)’ 的态射很有意思。它们是保持数据原子奇偶性的函数。

因此，仅仅通过挑选出一些我们感兴趣的数学对象，我们就找到了一种选出相似对象的方法，以及与之相关的一种新数学结构。尽管这只是一个初步的玩具示例，但这似乎是一种具有普遍意义的策略。

尽管空间与态射的理论显然带有某种范畴论的色彩，但两者之间也存在显著差异。在范畴论中，只有当定义域和陪域相一致时，态射才能进行复合。在 coda 中，任意 数据 F、 G 和 H 都可以复合为$X \mapsto H\colon\ G\colon\ F\colon\ X$。例如，表 3 中的积 $F_1 * F_2 * \cdots * F_n$ 始终有定义。另一个主要差异在于空间与态射之间的关系。在空间理论中，空间 A 与空间 B 之间的态射由 A 和 B 决定。另一方面，在范畴中，态射总是可以有多种选择。例如，完全可以定义一个以任意函数作为态射的实向量空间范畴。

7 数学是一致的吗?

我们已将空数据定义为 真 ，并将原子数据定义为 假。然而，由于纯数据不可能同时为真和假，我们似乎已经证明了 coda 的一致性，这似乎与哥德尔第二不完备性定理相矛盾。为了探究这一问题，我们尝试直接用 coda 表达 coda 的一致性。以下数据

$$\text{ap } \{\text{xor } (\text{coda} : \text{B}) : (\text{not} : \text{coda} : \text{B})\} : \text{allByteSequences} : \tag{11}$$

在逻辑上表达了 coda 语言中的 coda 一致性。这里的“coda”运算通过满射 $s \mapsto (\{s\}:)$ 将有限字节序列映射为数据。这意味着，对于每个字节序列 s ，都会对 (coda : s) 和 (not : coda : s) 求值；如果它们的逻辑值有任何一次相同，(11) 的整体值就是 false。尽管在 (11) 中，‘xor (coda : B) : (not : coda : B)’ 的每次应用结果都为 true，但我们不能就此断定 (11) 为 true，因为 (allByteSequences:) 最终会产生整个字节序列 (11)，而后它会被无限递归地重新求值。因此，我们遇到的情况是，(11) 等于

$$()()()()()\ldots() \ (some\ undecided\ data) \tag{12}$$

由无穷无尽的空序列与永远处于未决状态的数据串联而成。我们知道，未决数据永远不会产生一个原子，但我们也知道，它永远不会消失，这意味着 (11) 永远无法完全等于 ()，而这正是真（‘truth’）的定义。这似乎是一个令人满意的答案。类似哥德尔的局限性似乎出现了，但与此同时，也很清楚，coda 在本质上是一致的，因为不可能出现任何实际的矛盾，尽管我们无法在 coda 的内部意义上证明“不会出现矛盾”这一表达式为真。

8 悖论

在此，我们将考察逻辑悖论，将其作为对所提出的逻辑观进行压力测试的一种方式。从 coda 的视角来看，任何命题，无论是否构成悖论，都应表示为纯数据。任何纯数据都可以在给定的上下文中进行求值，因此，coda 必须针对任何此类命题给出一个 coda 逻辑结果。

8.1 哥¨德尔

回顾哥德尔第二不完备性定理[9]的整体结构。哥德尔巧妙地在皮亚诺算术（PA）中构造了一个“哥德尔句子”G，该句子断言其自身在皮亚诺算术（PA）中不可证。假设皮亚诺算术（PA）是一致的，即在皮亚诺算术（PA）中证明的句子实际上在柏拉图意义上都为真，那么我们便可以按照以下方式，在柏拉图式逻辑中于皮亚诺算术（PA）之外进行推理。

1. 假设 G 是假的；

2. 接着，G 在 PA 中是可证明的；3. 由于 PA 是一致的，根据假设，G 在柏拉图意义上为真；4. 因此，G 在 PA 中 not 可证明。 $\Rightarrow\Leftarrow$；5. 因此，G 为真。

因此，我们得出结论：G 在柏拉图意义上为真，但 G 在皮亚诺算术（PA）中不可证。 .

在 coda 中，数据 X 可证明，当且仅当 $X =()$，因此我们只需计算一个“G¨odel coda”$G?$，其定义为 $G? \mapsto \text{not} : G?$。这个定义由

- 令 G : not : G?

用 coda 语言。对 $G?$求值几次后，我们得出结论： $G?$ 等于未决数据。

$$(\text{not} : (\text{not} : (\text{not} : (\text{not} : (\text{not} : (\text{not} : (\text{not} : (\text{not} : ((\text{not} :) : (G? :)))))))))) \tag{13}$$

它既非真（空），也非假（原子）。由于对（13）进行进一步求值只会产生更多相同的结果，因此， $G?$ 是一种<escape>不可判定</escape>的数据。在这一点上，可以重复 G¨odel 的论证，从而得出结论： $G?$ 在柏拉图主义意义上为真，但在 coda 中不可证明。与 G¨odel 的论证相比，这种解释似乎不那么令人信服，仅仅是因为问题的自指性来源显得更加明显。另一种选择是从柏拉图主义的角度接受 coda 逻辑，并直接认为 $G?$ 既非真，也非假。

8.2 贝里

对于说谎者悖论之类的问题，一种可能的态度是，干脆将这个悖论排除在数学考量之外。毕竟，并非所有英语句子在数学上都有意义，也许说谎者悖论不过是另一句这样的句子。面对贝里悖论 [11] ，这种观点就不那么令人信服了，因为贝里的提议似乎是一个合理的数学问题。什么

- 无法用少于十二个词定义的最小正整数?

问题就在这里。显然，对于任意正整数 N， N 要么可以用少于十二个词定义，要么不能。因此，必定存在一个无法用少于十二个词定义的最小整数。但如果这样的整数存在，我们就已经成功地用少于十二个词定义了它。这就产生了矛盾。

我们约定，如果某个正整数出现在某个字节串 s 所对应的序列 ($\{s\}$:) 中，就称它为“可用 coda 定义”。令 (bytes:N) 生成所有长度小于或等于 N 的字节序列，再令 (posint:B) 筛选出其中恰好为正整数的项。令 (berry:B) 表示不在 B 中的最小正整数。那么

$$\text{berry} : \text{posint} : \text{coda} : \text{bytes} : 26 \tag{14}$$

计算的是在 coda 中无法用少于 26 个字节定义的最小正整数。然而，由于 (14) 的长度为 26 个字节，整个字符串 ‘berry:posint:coda:bytes:26’ 最终会再次作为输入传递给 ‘coda’，从而保证 (14) 永远无法计算出一个正整数。因此，(14) 属于 coda 逻辑类别 不可判定。

8.3 柯里

柯里悖论[8] 是另一个自指悖论。

$$\text{If } (15) \text{ is true, then Germany borders China.} \tag{15}$$

我们的推理如下。假设 (15) 为真。那么德国与中国接壤。因此我们证明了“如果 (15) 为真，那么德国与中国接壤”。这正是 (15)。

在 coda 中，我们可以定义

- 令库里句：库里句蕴含？：德国与中国接壤？ _ _

这里（蕴含 A:B）在 A 和 B 为真或假时，具有标准的蕴含真值表；在其他情况下则处于未决状态。在 coda 中对“库里句？”进行求值时，正如预期的那样，它永远是未决数据。它是不可判定的。

8.4 Yablo

既然说谎者悖论、贝里悖论和库里悖论都包含自指句子，人们很容易以为自指句子就是问题的根源。Yablo[10] 指出了一个没有自指句子的简单悖论，从而证明事实并非如此。考虑以下无限句子序列 Y_1，Y_2，.....

1. Y_i 对所有 $i > 1$都为假。2. Y_i 对所有 $i > 2$都为假。3. Y_i 对所有 $i > 3$都为假。4. .....

假设某个 Y_n 为真。于是，所有大于 Y_n 的句子都为假。但这意味着，所有大于 Y_{n+1} 的句子也都为假，这就意味着 Y_{n+1} 为真，与 Y_n 为真的假设相矛盾。因此，对于任何 n，Y_n 都不可能为真。但如果所有 Y_n 都为假，那么Y_0 就为真。这是一个矛盾。

与前面的情况一样，我们可以采用具体的方法。下面给出 Yablo 的定义。

- 定义 Yablo : {ap not : Yablo : skip 1 : nat : B}

使得（Yablo : n）当（Yablo : $n+1$）、（Yablo : $n+2$）.....都为假时为真。图 4 显示了第 hikwalaho e

```
#
#   Compute Yablo's paradox.
#
Yablo : 1
```

```
(ap not:(ap not:(ap not:(ap not:(ap not:(({ap not}:(({skip}:(skip 1:(nat:(skip
1:(nat:(skip 1:(nat:(skip 1:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 (na
t:20)))))))))) ({1}:(skip 1:(nat:(skip 1:(nat:(skip 1:(nat:(skip 1:1 2 3 4 5 6 7
8 9 10 11 12 13 14 15 16 17 18 19 (nat:20)))))))))):(({nat }:(skip 1:(nat:(skip
1:(nat:(skip 1:(nat:(skip 1:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 (na
t:20)))))))))):({ B}:(skip 1:(nat:(skip 1:(nat:(skip 1:(nat:(skip 1:1 2 3 4 5 6
7 8 9 10 11 12 13 14 15 16 17 18 19 (nat:20))))))))))):({Yablo : skip 1 : nat
: B}:(({skip}:(skip 1:(nat:(skip 1:(nat:(skip 1:(nat:(skip 1:1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 (nat:20)))))))))) ({1}:(skip 1:(nat:(skip 1:(nat:
(skip 1:(nat:(skip 1:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 (nat:2
0)))))))))):(({nat }:(skip 1:(nat:(skip 1:(nat:(skip 1:(nat:(skip 1:1 2 3 4 5 6
7 8 9 10 11 12 13 14 15 16 17 18 19 (nat:20)))))))))):({ B}:(skip 1:(nat:(skip
1:(nat:(skip 1:(nat:(skip 1:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 (na
t:20))))))))))))))))) (ap not:(ap not:(ap not:(ap not:(ap not:)))))
```

图 4： 在 *coda* 笔记本中对亚布洛命题（*Yablo:1*）进行求值，结果表明（*Yablo:1*）在深度达到 *20* 时仍未得到判定，这正如预期的那样。

标准赋值 深度为 20 时 (Yablo:1) 的 n。在 G¨odel 情形中，从 th e

输出可“明显”看出，评估到更深的层次不会改变任何结果。我们没有尝试实际证明这一点，但根据经验，(Yablo:1) 在深度 20 时仍未得到判定，而且我们可以预期，增加深度也不会带来任何变化。

8.5 没有答案的问题

数学中的问题在句法上可能看起来合理，但实际上可能没有答案。例如，在尚未定义 x 和 y 时提出“$x = y$ 是吗？”，或者“最后一个自然数是什么？”、“所有自然数之和是多少？”，以及本节考察的任何悖论。在 coda 中，所有这些问题都属于逻辑类别 未决，而除第一个问题外，其余问题也都属于 不可判定。未决数据是指“目前没有答案的问题”，而不可判定数据是指“永远不可能有答案的问题”。从某种意义上说，说谎者悖论没有答案，就如同自然数没有最后一个元素一样。

9 总结

提出了一种数学公理基础，以有限序列为起点，而非从逻辑、集合或类型出发。在这个称为“coda”的框架中，所有数学量都是纯数据。Coda 只有一条公理，即定义公理，它规定了什么构成有效定义。该公理导出对纯数据的稳定分类，我们将其解释为一个逻辑系统，其中 $empty/atomic/neither$ 数据分别解释为 $true/false/undecided$逻辑值。该公理还引入了一种语言，为纯数据操作提供程序化控制。所有字节序列都是有效的语言表达式，因此不需要语言公理，也不需要用于定义有效和无效语法的公理。证明和计算均以数据相等性来定义。与具有柯里－霍华德对应的依赖类型系统不同，在 coda 中，证明和计算是一回事。文中还引入了空间和态射理论，它似乎发挥着类似于范畴论在现代数学中所起的作用。我们首次尝试开展“数学机器学习”，从分类问题中推导出空间。作为对所提出逻辑的一项检验，我们以计算方式探讨数学的一致性问题，以及由 G¨odel、贝里、柯里和 Yablo 引发的悖论。我们得出了数学具有一致性的肯定结论，并对这些悖论得出看似令人满意的结论：每个悖论都会在不可判定类别中产生数据。

显然，仍有大量工作值得开展。迄今为止，人们只尝试了粗略的求值策略，而这一领域很可能有很大的提升空间。这些提升对于开发数学探索和证明辅助工具很可能至关重要。第 6 节中数学机器学习的例子，充其量只是对可能性的提示。空间的理论似乎内容丰富、前景可期，而目前才刚刚起步。

参考文献

[1] Dybjer, Peter 和 Erik Palmgren，《直觉主义类型论》，《斯坦福哲学百科全书》（2023年春季版），Edward N. Zalta 和 Uri Nodelman（编）[2] Steve Awodey 和 Thierry Coquand，《单值基础与数学的大规模形式化》，高等研究院，2013年。[3] 同伦类型论：数学的单值基础，高等研究院，2013年。

[4] S.Watt，依赖类型与范畴编程，TRICS，西安大略大学，2012年，网址：https://cs.uwaterloo.ca/∼smwatt/talks/2012-trics-categorical.pdf；载于《计算机代数手册》第265–270页，J. Grabmeier、E. Kaltofen、V. Weispfenning（编），施普林格出版社，海德堡，2003年，ISBN 3-540-65466-6；S.Youssef，*Aldor* 中范畴论的前景，2008年，https://physics.bu.edu/∼youssef/aldor.pdf。[5] Bagaria，Joan，集合论，《斯坦福哲学百科全书》（2023年春季版），Edward N. Zalta 与 Uri Nodelman（编）[6] 可通过 https://github.com/Saul-Youssef 获取 coda 的 Python 实现。该软件包括 Python API、命令行界面和 Jupyter Notebook 插件。[7] coda 语言是 *egg* 的衍生语言。参见 Saul Youssef、John Brunelle、John Huth、David C. Parkes 和 Margo Seltzer，最小经济分布式计算，arXiv:0902.4730。J. Brunelle 等，载于 GECON 2006：《第三届网格经济学与商业模式国际研讨会论文集》，新加坡，2006年5月16日，H. Lee 和 S. Miller 编。新加坡；新泽西州哈肯萨克：世界科学出版社。[8] Shapiro，Lionel 和 Jc Beall，柯里悖论，《斯坦福哲学百科全书》（2021年冬季版），Edward N. Zalta（编）[9] Raatikainen，Panu，$G\ddot{}$哥德尔不完备定理，《斯坦福哲学百科全书》（2022年春季版），Edward N. Zalta（编）[10]S. Yablo（1985年）。真理与反思。《哲学逻辑杂志》。14（2）：297–348。doi:10.1007/BF00249368。[11] Nicholas Griffin（2003年6月23日）。《剑桥罗素指南》。剑桥大学出版社。第63页。ISBN 978-0-521-63634-6。[12]Barry Mazur，何时一个事物等于另一个事物？，哈佛大学，2007年，https://people.math.harvard.edu/∼mazur/preprints/when is one.pdf。