Defining the mathematical family tree with a minimal programming language

Max Tegmark

Last updated: November 23, 2026

Abstract

I introduce MLISP, a Turing-complete language with nine single-character commands `a b c d f 0 i j k`, and use it to define 88 mathematical structures and a family tree of 65 theories that they model, with three aims:
Pedagogy: It presents the mathematical structures in a unified way, with standardized presentation tableaus that define them and axiomatized theories that describe them.
Connections: It highlights how they are related in the family tree via added symbols, axioms, *etc.* While the full tree is obviously infinite, its lower parts naturally separate into six main branches: the relational branch, the algebraic branch, the logic branch, the number branch, the set branch and the continuum branch.
Foundations: The definitions, key theorems and models of each entry are verified in *Lean*. Mathematical structures are defined as equivalence classes of presentations, which in a constructivist spirit are defined in a simple, explicit and self-contained way in standardized presentation tableaus with minimal human baggage. For finite structures, the generating functions are defined by lookup tables. For countably infinite ones, they are instead defined computationally using MLISP, since more complex languages such as *Lean* are massive overkill for rigorously defining standard arithmetic operations (for example, the program `jibjbfid` defines subtraction).
An interactive version is available at `https://tegmark.org/mathtree`. The Lean code is public, also verifies the correctness of MLISP and includes 43% of the theorems from Mathlib 2026-10-11.

1 Introduction

After millennia of progress, mathematics is currently developing faster than ever because of artificial intelligence, raising new opportunities and challenges [1]. For example, a recent declaration from leading mathematicians [2] emphasizes the importance of not merely proving formal results, but also making them human-understandable. In this spirit, the goal of this paper is pedagogical, defining and presenting a unified human-readable family tree of basic mathematical theories and structures. This tree is obviously highly incomplete (the full tree is infinite) and should be viewed as work in progress; it emphasizes mainly theories and structures that play a foundational role in university courses, mathematical Wikipedia pages and *Lean*'s Mathlib [3].

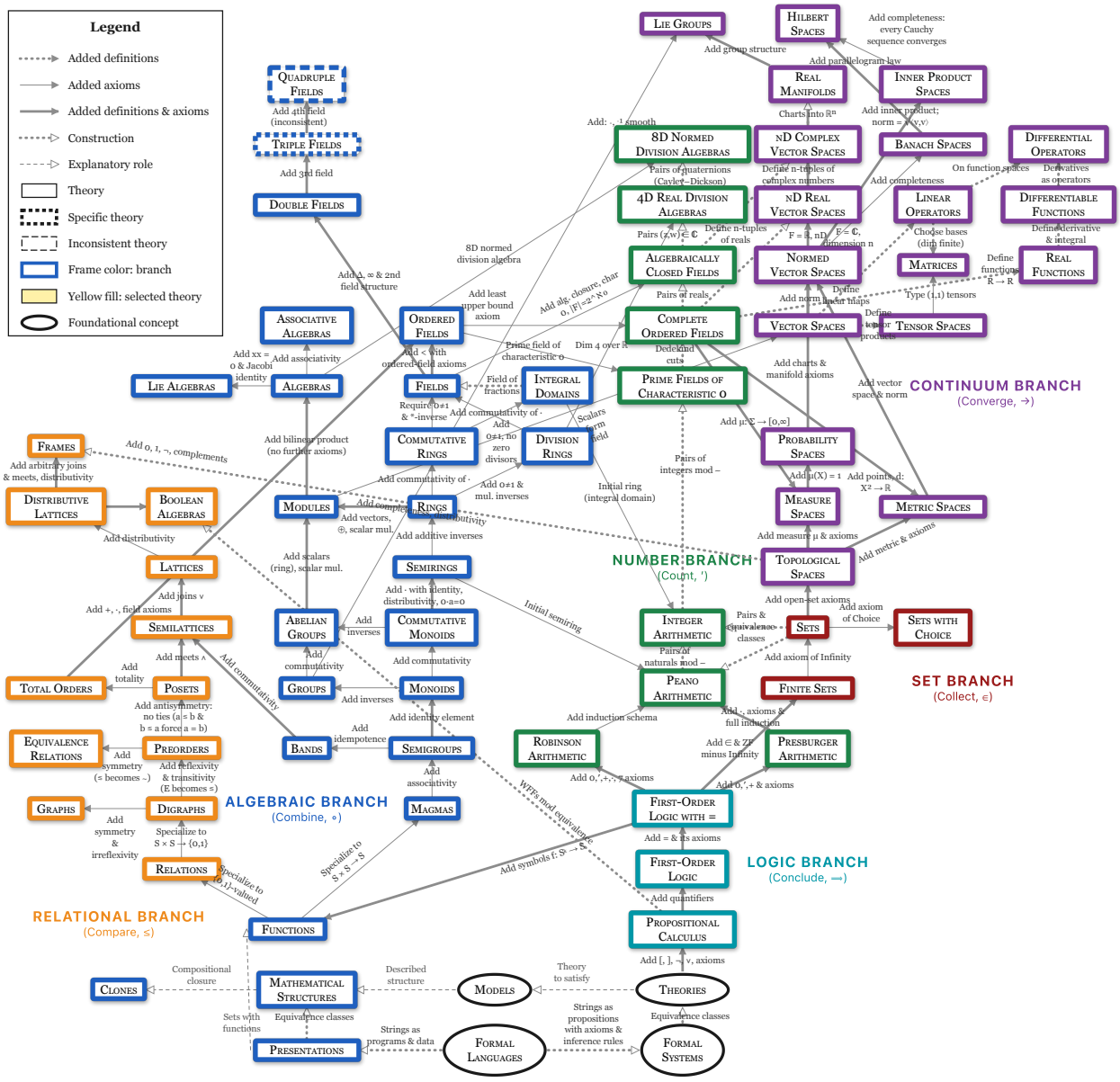


Figure 1: Family tree of mathematical theories. If you are reading this on a computer, you can click any of the boxes for details. You will find a fully interactive version at https://tegmark.org/mathtree. Boxes denote mathematical theories (equivalence classes of formal systems). Ovals, and the boxes Presentations, Mathematical Structures and Clones, denote foundational concepts that are not themselves theories.

1.1 Mathematical theories

Figure 1 shows our family tree[1] of basic mathematical theories, defined below as equivalence classes of formal systems that imply each other's axioms (after invertible translation). Each box is described in the online theory atlas at https://tegmark.org/mathtree, and various mathematical structures that are models of the theory are described in the online model atlas there. This tree graphically extends the 1997 family tree from [4], and the arrows agree with Mathlib's "extends" constructions.

The arrows and lines connecting the boxes represent various types of relationships between theories, such as *specialization* (added axioms, so same symbols, more theorems), *definitional extension* (added symbols and definitions, so no new theorems about the old symbols) and *construction* (the child's universe is built from the parent's via, *e.g.*, pairs, quotients or cuts; models are not inherited along these arrows). We refer to a mathematical theory as a *child* of another if it is a specialization, definitional extension or both (these parent-child relations have a gray rather than white arrowhead in the figure). Every model of the child is a model of the parent: forget the new symbols and the parent's axioms still hold.

While the full tree is infinite, its lower parts are seen to naturally separate into six main branches:

- **Relational branch**, centered around binary Boolean functions $S \times S \to \{0, 1\}$ interpreted as comparisons
- **Algebraic branch**, centered around binary operations $S \times S \to S$ interpreted as combining two elements
- **Logic branch**, centered around deduction rules for strings interpreted as well-formed formulas and theorems
- **Number branch**, centered around entities with arithmetic operations
- **Set branch**, centered around collecting elements into sets and subsets
- **Continuum branch**, centered around continuous spaces where limits can be taken

They can be alliteratively summarized as the "compare", "combine", "conclude", "count", "collect" and "converge" branches or, more succinctly, by their foundational symbols $\leq$, $\circ$, $\Longrightarrow$, $'$, $\in$ and $\to$, respectively. We have grouped theories into branches according to what their typical models are, not by the language of their axioms: for example, we color Vector Spaces purple (the Continuum branch) even though their axioms are algebraic.

[1] Technically, it is not a tree but a rooted directed acyclic graph — but the same can be true of conventional family trees.

1.2 Mathematical structures and terminology

We must not conflate the map with the territory, *i.e.*, the description with that which it describes. Just as physics theories are distinct from the physical systems that they (approximately) describe, mathematical theories must not be conflated with the mathematical structures that they (exactly) describe:

Description	Described
Map	Territory
Physics theories	Physical systems
Mathematical theories	Mathematical structures

For both theories and structures, we also wish to distinguish between their mathematical content and arbitrary notational choices used to specify them. We will therefore use the terminology of Table 1, as illustrated by the following examples:

- the sphere S^2 is a single mathematical structure regardless of which of the equivalent coordinate system atlases in Figure 2 is used to define it as a manifold;
- the 4-element cyclic group C_4 is a single mathematical structure regardless of which of the equivalent group multiplication tables (Cayley tables) from Figure 2 is used to specify it;
- Propositional Calculus is a single mathematical theory regardless of which of the equivalent formal systems is used to define it: the usual four axioms for NOT and OR or Nicod's single axiom for the Sheffer stroke (NAND).

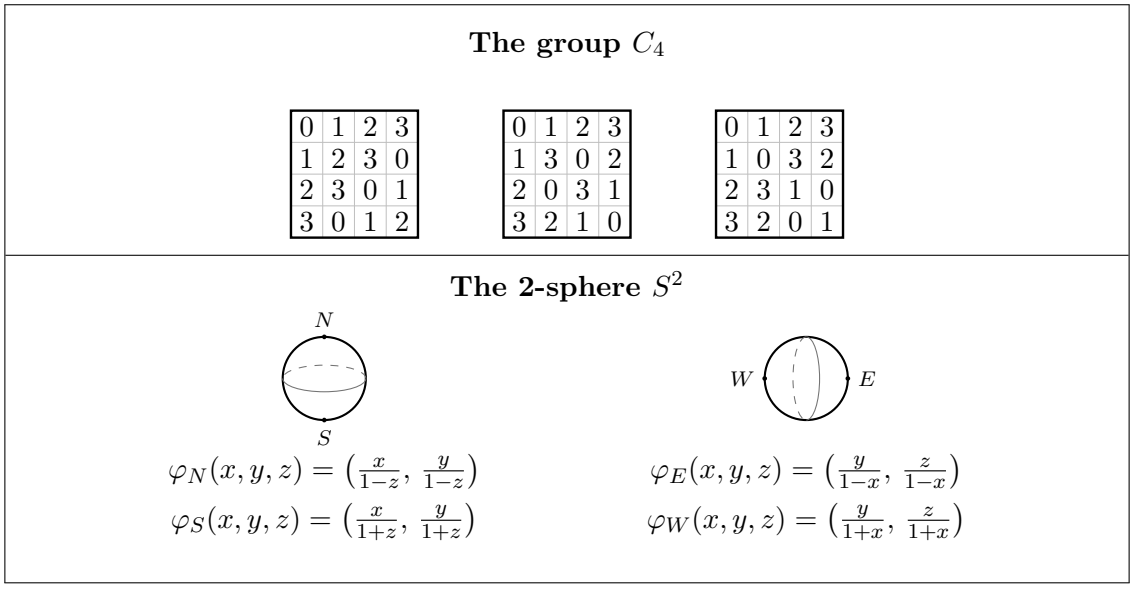


Figure 2: Equivalent specifications of the cyclic group of 4 elements (top) and the manifold S^2 (the sphere, bottom). We consider equivalent specifications as defining the same mathematical structure (here C_4 and S^2). The second and third group multiplication tables are related to the first by relabeling elements $2 \leftrightarrow 3$ and $1 \leftrightarrow 2$, respectively. Each chart φ_i is defined on the whole unit sphere except at its projection point ($z = 1$, $z = -1$, $x = 1$ and $x = -1$, respectively).

	Mathematical theory	**Mathematical structure**	**Closed mathematical structure (clone)**
Defined by	Formal system	Presentation	Presentation
Equivalence definition	Prove each other's axioms (after invertible translation)	Relabeling	Generate each other's generator functions
Ingredients	Strings viewed as propositions	Strings viewed as programs	Strings viewed as programs
Starting point	Axioms	Functions	Functions
Expansion method	Theorem deduction	None	Function composition

Table 1: Our terminology for mathematical theories and structures. A translation is invertible if translating a formula there and back gives a formula provably equivalent to the original.

Specifically, we use the standard definition of a *formal system* in terms of a *specific* choice of symbols, axioms and deduction rules, and define a mathematical theory as an equivalence class of formal systems that prove each other's axioms after invertible translation. This equivalence class determines the deductive closure, the (normally infinite) set of theorems deducible from any formal system in the class; we let "mathematical theory" denote this class, while model theorists often use that word to denote that deductive closure.

Analogously, we define a *presentation* as a *specific* choice of functions on sets, and define a mathematical structure as an equivalence class of presentations under relabeling (two isomorphic presentations are deemed equivalent). This is a deliberate departure from the common model-theory nomenclature "structure" for what we call a presentation. Our definition is the same as the one Carnap used in 1936 [5], as discussed by Schiemer [6, pp. 388–390]. It is more in line with structuralist mathematical views [7, 8, 9, 10, 11] and with everyday usage by working mathematicians. For example, almost no mathematicians would say that the three Cayley tables in Figure 2 specify different groups, or that the two shown function pairs specify different manifolds (which are indeed defined as equivalence classes of compatible atlases). Most mathematicians would probably also say that the 2-sphere (or the group C_4) counts as a mathematical structure. According to our definitions, each group (counted in OEIS sequence A000001 [12]) is *one* mathematical structure.

1.3 Presentations and models

A presentation is said to be a *model* of a formal system if there exists an interpretation of each symbol of the formal system in terms of the sets and functions of the presentation such that every theorem holds in the presentation under the interpretation. In turn, a mathematical structure is said to be a *model* of a mathematical theory if a presentation in the equivalence class of the former is a model of a formal system in the equivalence class of the latter. All the model relations shown in the family tree are verified in Lean. We will elaborate on the notion of a *closed mathematical structure* in Section 3.

For *formal systems*, a very mature toolkit exists in the form of specialized computer languages such as Lean [13], Rocq [14] and Isabelle [15]. For *presentations*, less standardization has occurred, so we make a major effort in this direction. Section 2 presents our system for defining finite and countably infinite presentations as computations. In Section 2.4, in the spirit of constructivist mathematics [16], we define for this purpose a Turing-complete programming language *MLISP* that is dramatically simpler than traditional programming or proof languages, yet is able to define all functions in our presentations in simple halting programs that are often fewer than 10 characters long. A program implementing and running MLISP can be written in merely 195 characters — in MLISP. The advantage here is that one can prove things about presentations without having to make

assumptions about how traditional programming or proof languages implement their functions. For small finite sets, MLISP functions are also shown as human-friendly lookup tables.

The rest of this paper is organized as follows. We describe how we define presentations and computations in Section 2 and mathematical structures in Section 3. Section 3 also relates this to physics, Section 4 summarizes our conclusions and Appendix A describes how we counted closed mathematical structures. An interactive version of the family tree of mathematical theories and structures is available at https://tegmark.org/mathtree. All MLISP and Lean code is at https://github.com/mtegmark/mlisp9.

2 Computationally defining presentations

In this section, we clarify how we define the left base of the tree (presentations, mathematical structures), since there is much less convergence on that in the mathematics literature than there is for the right part (formal systems, theories). We take a constructivist computational approach, since more complex languages such as *Lean* are massive overkill for this task. Instead we use Lean to verify MLISP. Reducing the trusted computing base not only makes foundations more human-understandable, but also reduces vulnerability to bugs in programming languages that are used to define presentations. Numerous bugs have been found in various C compilers, and a bug was recently found in Lean itself that enabled proving false propositions [17].

2.1 Presentations

Definition: A presentation is a collection of sets and functions: sets $S_1, \ldots, S_n$ together with functions $f_a \colon S_{i_1} \times \cdots \times S_{i_k} \to S_j$ between them. We refer to $\{0,1\}$-valued functions in the presentation as *relations* or *Boolean functions*. In other words, a presentation is a concrete object, sets with functions defined by lookup tables or computations, before any axioms are imposed and before equivalent descriptions of the same thing are identified.

2.2 Finite presentations

For *finite presentations* (when all sets are finite), we simply need to specify each set followed by lookup tables defining each of its functions. For example, we could specify the presentation of the Klein 4-group like this:

1 set: $S_1 = \{e, a, b, c\}$

3 functions:

$$\begin{aligned} \circ &: S_1 \times S_1 \to S_1 \\ {}^{-1} &: S_1 \to S_1 \\ e &\in S_1 \end{aligned}$$

Lookup tables:

$g \circ h$	e	a	b	c
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

	g^{-1}
e	e
a	a
b	b
c	c

e
e

However, we are keen to standardize the notation, simplify away as many arbitrary notational choices as possible and draw focus to the actual mathematical properties of the presentation. We therefore relabel the elements of all finite sets as $\{0, 1, \dots\}$, drop inferable labels and specify each presentation by standardized human-friendly *presentation tableaus* as in these examples:

Klein 4-group:

4

$\circ$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

$^{-1}$
0
1
2
3

e
0

Total order of $\{0, \dots, 5\}$:

6 2

$\leq$

1	1	1	1	1	1
0	1	1	1	1	1
0	0	1	1	1	1
0	0	0	1	1	1
0	0	0	0	1	1
0	0	0	0	0	1

Transformation monoid T_2:

4

$\circ$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	2	2	2
3	3	3	3	3

e
0

Frame of open sets of 3-point space:

5

$\wedge$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	0	1	1
2	0	0	2	2	2
3	0	1	2	3	3
4	0	1	2	3	4

$\vee$	0	1	2	3	4
0	0	1	2	3	4
1	1	1	3	3	4
2	2	3	2	3	4
3	3	3	3	3	4
4	4	4	4	4	4

$\Rightarrow$	0	1	2	3	4
0	4	4	4	4	4
1	2	4	2	4	4
2	1	1	4	4	4
3	0	1	2	4	4
4	0	1	2	3	4

Rock–paper–scissors magma:

3

$\circ$	0	1	2
0	0	1	0
1	1	1	2
2	0	2	2

Diamond lattice M_3:

5

$\wedge$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	0	0	1
2	0	0	2	0	2
3	0	0	0	3	3
4	0	1	2	3	4

$\vee$	0	1	2	3	4
0	0	1	2	3	4
1	1	1	4	4	4
2	2	4	2	4	4
3	3	4	4	3	4
4	4	4	4	4	4

Ring $\mathbb{Z}/4$:

4

$+$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\cdot$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

$-$
0
3
2
1

0	1
0	1

Double field D_4:

4

Δ	0	1	2	3
3	2	1	0	
2	3	0	1	
1	0	3	2	
0	1	2	3	

$+$	0	1	2	3
0	1	2	3	
1	2	0	3	
2	0	1	3	
3	3	3	3	

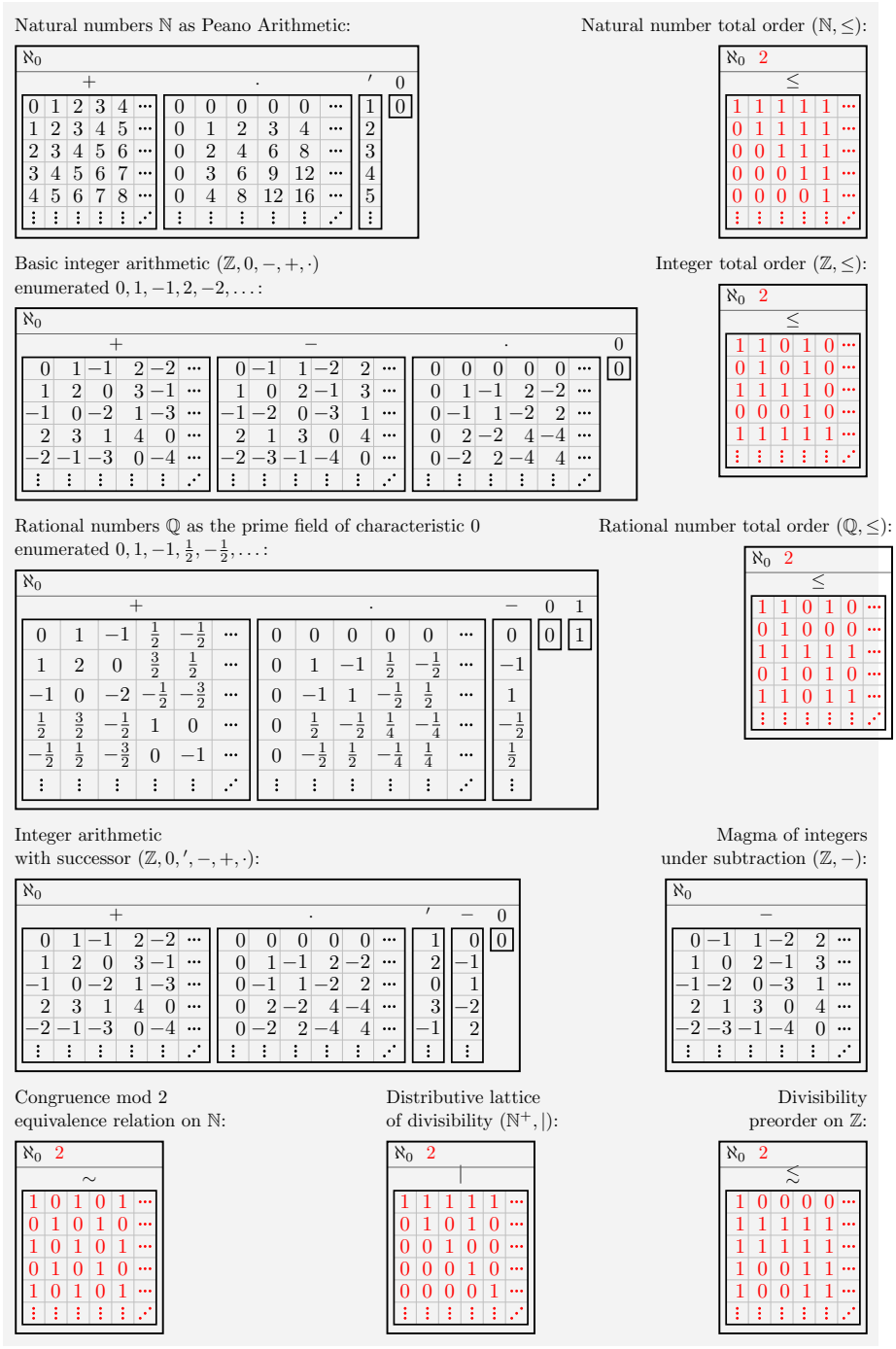
$\cdot$	0	1	2	3
0	0	0	0	3
0	1	2	3	
0	2	1	3	
3	3	3	3	

0	1	∞
0	1	3

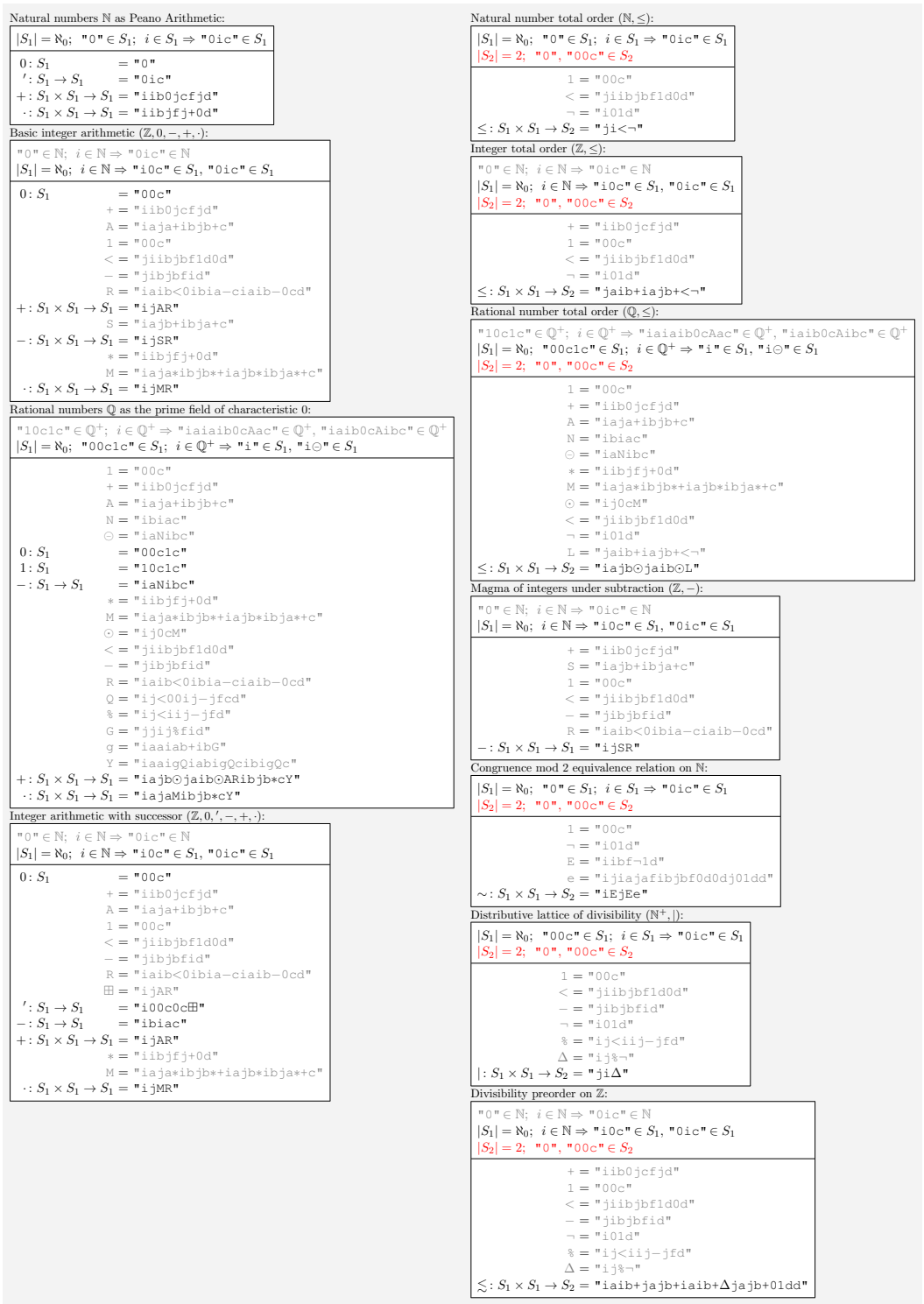
The cardinality of each set is written in a separate color in the set specification area, above the horizontal line. Note that the arity of each function can be inferred from the shape of its lookup table, and the identities of the input and output sets can be inferred from the colors of the lookup table boxes and the digits therein. The operation labels (*e.g.* e, $\circ$) are redundant, not part of the presentation itself, but we can optionally leave them in to make the tableaus more human-readable.

2.3 Countably infinite presentations

For presentations where at least one set is countably infinite, we could try to visualize them the same way, for example, by enumerating integers in the order $0, 1, -1, 2, -2, \dots$ and rationals in a signed extension of the Calkin–Wilf enumeration [18], $0, 1, -1, \frac{1}{2}, -\frac{1}{2}, \dots$ (zero first, then each positive rational in Calkin–Wilf order, immediately followed by its negative):



However, this is unacceptable, because it relies on unspecified human intuition to guess how to fill in the parts indicated by triple dots. Instead, we define them computationally, using standardized *computational* presentation tableaus as in these examples:



Here the mysterious-looking strings such as "`jibjbfid`" are ultra-short computer programs that fully define each function in a minimal LISP-like language called MLISP that I created for this purpose and will describe below. The additional definitions shown in gray are helper functions, not part of the set of functions defining the presentation.

2.4 Defining functions computationally with MLISP

MLISP is a minimal yet Turing-complete programming language helpful for defining functions. A program is a string of single-character symbols, each acting in turn on a stack of binary trees, as when parsing reverse Polish notation. MLISP has only the 9 basic 1-symbol operations[2] `a b c d f 0 i j k` defined as follows:

Operation	What it returns	Args
`a`	**a**-part: the first part of the pair i	1
`b`	**b**-part: the second part of the pair i	1
`c`	**c**ombine: the ordered pair (i, j)	2
`d`	**d**ecide: j if i is a pair, otherwise k (only the chosen branch is evaluated)	3
`f`	**f**unction being defined: for recursion; as many arguments as that function	
`0`	the atom 0	0
`i`	the first input	0
`j`	the second input	0
`k`	the third input	0

Table 2: The nine built-in MLISP commands

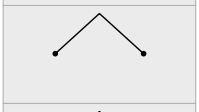
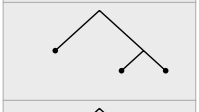
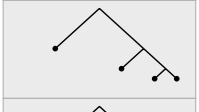
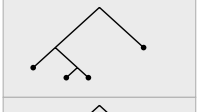
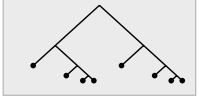
Human-friendly	Raw	Tree	Tree image
Natural number 0	`0`	`0`	
Natural number 1	`00c`	`(0,0)`	
Natural number 2	`000cc`	`(0,(0,0))`	
Natural number 3	`0000ccc`	`(0,(0,(0,0)))`	
Integer 2	`000cc0c`	`((0,(0,0)),0)`	
Integer −3	`00000cccc`	`(0,(0,(0,(0,0))))`	
Rational number −2/3	`0000ccc0000cccc`	`((0,(0,(0,0))),(0,(0,(0,0))))`	

Figure 3: MLISP variables are MLISP programs using 0 and c in reverse Polish notation ("raw" column), and can be interpreted as, *e.g.*, binary trees, natural numbers, integers and rational numbers.

[2]The even more minimal (but still Turing-complete) language mLISP using only 7 symbols is described and implemented at https://tegmark.org/mathtree/mlisp7.html; we stick with the 9-symbol version here because its programs tend to be slightly shorter and more human-readable.

The computational logic of the language is independent of how the binary trees are represented. As illustrated in Figure 3, it is intuitive to write them as nested lists such as `(0,(0,(0,0)))` and to draw them as actual trees. In practice, under the slogan "data is code", we instead represent them as the MLISP program generating them (the "raw" column). So the example above corresponds to "0000ccc", which is simply the nested list in reverse Polish notation. There are many convenient ways to interpret such variables `i`. We can interpret them as bit strings if we replace "c" by "1", making this example "0000111". We can interpret them as nested lists of zeros, with `ia` interpreted as the first list element and `ib` as the rest. For our purposes of defining presentations, we can further interpret a list of n zeros as the natural number n. We can also interpret a pair of natural numbers (i, j) as an integer $i - j$ whose standard form is $(i, 0) = i$ and $(0, i) = -i$. We can interpret a pair (i, j) of an integer i and a positive natural number j as a rational number $(i, j) = i/j$. Alternatively, as described in the next section, we can represent natural numbers as binary strings to allow more efficient computation. The possibilities are endless, but none of this is part of the definition of the MLISP language itself. At `https://tegmark.org/mathtree/mlisp.html`, you can run MLISP programs in an interactive calculator and download reference implementations in Python, C and Lean.

Any undefined symbols are ignored, but you can define new programs: for example, `+iib0jcfjd#` defines + to mean `iib0jcfjd`. The definitions in Table 3 let you do arithmetic with natural numbers $\mathbb{N}$ represented as nested pairs: `0`, `1=(0,0)`, `2=(0,(0,0))`, `3=(0,(0,(0,0)))`.

Symbol	Interpretation	Args	Input types	Output type	Definition
1	one, true	0		$\mathbb{N}$	00c
2	two	0		$\mathbb{N}$	01c
e	equal ((0,0) if i = j, else 0)	2	any, any	0 or 1	ijiajafibjbf0d0dj01dd
′	successor	1	$\mathbb{N}$	$\mathbb{N}$	0ic
−	minus (undefined for i < j)	2	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{N}$	jibjbfid
+	plus	2	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{N}$	iib0jcfjd
*	times	2	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{N}$	iibjfj+0d
^	power	2	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{N}$	jijbfi*1d
!	factorial	1	$\mathbb{N}$	$\mathbb{N}$	iiibf*1d
¬	not	1	0 or 1	0 or 1	i01d
&	and	2	0 or 1, 0 or 1	0 or 1	ij10d0d
∨	or	2	0 or 1, 0 or 1	0 or 1	i1j10dd
<	less than (i<j)	2	$\mathbb{N}$, $\mathbb{N}$	0 or 1	jiibjbf1d0d
%	remainder (i mod j)	2	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{N}$	ij<iij−jfd
Δ	divisible (i by j)	2	$\mathbb{N}$, $\mathbb{N}$	0 or 1	ij%¬
P	i indivisible by all of 2…j	2	$\mathbb{N}$, $\mathbb{N}$	0 or 1	jbijΔ¬ijbf&1d
Π	prime (i ≥ 2)	1	$\mathbb{N}$	0 or 1	iibP

Table 3: MLISP programs defining arithmetic with natural numbers. The first symbol of each program is defined by the rest.

Worked example: `0ic` (the successor $'$, one argument, with i = 3)

Symbol	Stack afterwards	Comment
`0`	`0`	push the atom
`i`	`0, 3`	push the first input
`c`	`(0,3)`	combine the top two into the pair (0,3), which is the number 4

Worked example: `jibjbfid` (minus, two arguments: i − j, with i = 3, j = 1)

Symbol	Stack afterwards	Comment
`j`	`1`	push the second input
`i`	`1, 3`	push the first input
`b`	`1, 2`	b-part of 3 = (0,2) is 2: the predecessor of i
`j`	`1, 2, 1`	
`b`	`1, 2, 0`	the predecessor of j
`f`	`1, f(2,0)`	the function itself applied to the top two values (a recursive call, not yet evaluated)
`i`	`1, f(2,0), 3`	
`d`	`d(1, f(2,0), 3)`	decide: the first of the three is 1, a pair, so the result is the second, f(2,0); with $j = 0$, i would have been returned instead

Note that MLISP programs, like all finite programs, can represent and compute with many exact real numbers (as opposed to finite approximations such as 64-bit floating-point numbers), but they cannot individually specify all real numbers: there are only countably many finite programs, whereas a generic real number requires infinitely many bits to specify. Indeed, constructivist mathematicians have questioned whether real numbers exist mathematically in the same sense that integers or rational numbers do [19, 20] and it has been claimed that there is no evidence that real numbers are needed to describe our physical universe [21].

3 Defining mathematical structures

Equivalence class constructions are ubiquitous in mathematics, as illustrated in Table 4. Generally, the goal is to focus on the mathematical properties by stripping away (via a quotient) irrelevant details related to notation. The essence of a rational number is its value, not whether we write it as $\frac{1}{2}$ or $\frac{3}{6}$. Analogously, the essence of a mathematical theory such as Propositional Calculus is which theorems it proves, not whether the NOT function is written with the symbol $\sim$ or $\neg$, or whether it is defined via the usual four axioms or via the equivalent single axiom of Nicod [22].

Set	**Object**	**Equivalence relation**	**Normal form**
integers $\mathbb{Z}$	$(a, b) \in \mathbb{N} \times \mathbb{N}$	$a_1 + b_2 = a_2 + b_1$	smallest b
rationals $\mathbb{Q}$	$(a, b) \in \mathbb{Z} \times \mathbb{N}$, $b \neq 0$	$a_1 b_2 = a_2 b_1$	smallest b
reals $\mathbb{R}$	Cauchy sequence $a_i \in \mathbb{Q}$	$a_i - b_i \to 0$ in $\mathbb{Q}$	–
mathematical theories	formal system	prove each other's axioms after invertible translation	simplest?
mathematical structures	presentation	isomorphism	simplest?
closed mathematical structures (clones)	presentation	isomorphic closures	simplest?

Table 4: Many important mathematical objects are defined as equivalence classes in a set of other objects, that is, subsets whose members are all equivalent. It is often convenient to define a mapping to a standardized "normal form" serving as a representative of each equivalence class, for example "1/2" for all rational numbers (a, b) with $b = 2a$.

In this same spirit, we define a *mathematical structure* as an equivalence class of presentations. Now how exactly is the equivalence relation best defined — how broad should it be? The formal and colloquial usage of the term "mathematical structure" among mathematicians is not always consistent, so let us now discuss and motivate the choice made in our atlas.

3.1 Presentation isomorphism

There is universal agreement that the labels used to refer to set elements and functions have no mathematical meaning. Two presentations are therefore defined to be *isomorphic* if there is a one-to-one renaming of the elements of each set that turns every lookup table (or program output) of one presentation into the corresponding one of the other. In a human-friendly tableau, renaming set elements labeled $0, 1, \ldots$ permutes the digits inside the tables and reorders the rows and columns to match, as in the three isomorphic presentation tableaus defining the group C_4 in Figure 4 (top panel). Analogously, the bottom panel illustrates that the presentation of the Boolean functions (NOT, AND) is isomorphic to that for (NOT, OR), since they become identical if 0 and 1 are swapped (De Morgan's laws).

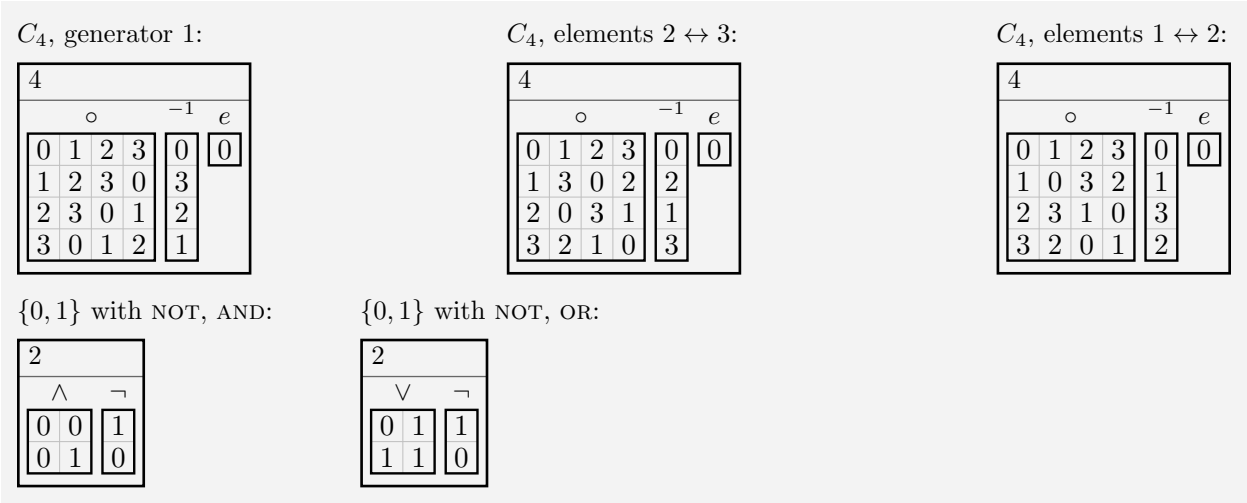


Figure 4: Equivalence by isomorphism. The top three presentations are isomorphic, differing in how their elements are labeled, and all represent the same mathematical structure: the cyclic group C_4. Analogously, the bottom two presentations are also isomorphic.

Figure 5 shows an analogous example of isomorphism for a countably infinite presentation of the natural numbers with the operations of Peano Arithmetic, representing n either in unary or in binary (essentially the difference between representing 13 as a string of length n like "1111111111111" versus a bit string of length of order $\log_2 n$ like "1101"). They are isomorphic because there are unary MLISP functions $f =$ "`iibf'0d`" and its inverse function $g =$ "`iibfibf+ia+0d`" that form a bijection between unary and binary number labels (strings) preserving all functions. For example, letting $\cdot$ and $\times$ denote multiplication in the two presentations, we have $f(i \cdot j) = f(i) \times f(j)$. The binary presentation is vastly more efficient computationally, requiring $\mathcal{O}\left((\log_2 n)^2\right)$ operations instead of $\mathcal{O}\left(n^2\right)$ to implement multiplication, but because it is isomorphic, it defines the exact same mathematical structure.

$(\mathbb{N}, 0, ', +, \cdot)$, elements as unary strings:

$\lvert S_1\rvert = \aleph_0$; `"0"` $\in S_1$; $i \in S_1 \Rightarrow$ `"0ic"` $\in S_1$	
$0 : S_1$	$=$ `"0"`
$' : S_1 \to S_1$	$=$ `"0ic"`
$+ : S_1 \times S_1 \to S_1$	$=$ `"iib0jcfjd"`
$\cdot : S_1 \times S_1 \to S_1$	$=$ `"iibjfj+0d"`

$(\mathbb{N}, 0, ', +, \cdot)$, elements as binary strings:

$\lvert S_1\rvert = \aleph_0$; `"0"` $\in S_1$; $i \in S_1 \Rightarrow$ `"1ic"` $\in S_1$; $i \in S_1$, $i \neq$`"0"` $\Rightarrow$ `"0ic"` $\in S_1$	
1	$=$ `"00c"`
$0 : S_1$	$=$ `"0"`
$' : S_1 \to S_1$	$=$ `"iia0ibfc1ibcd10cd"`
$+ : S_1 \times S_1 \to S_1$	$=$ `"ijiaja0ibjbf'c1ibjbfcdja1ibjbfc0ibjbfcddidjd"`
w	$=$ `"i0icid"`
$\cdot : S_1 \times S_1 \to S_1$	$=$ `"iiajibjfw+ibjfwd0d"`

Figure 5: Two computational presentations of the natural numbers $(\mathbb{N}, 0, ', +, \cdot)$, with the elements written as unary strings (top) and as binary strings (bottom).

3.2 Compositional equivalence and clones

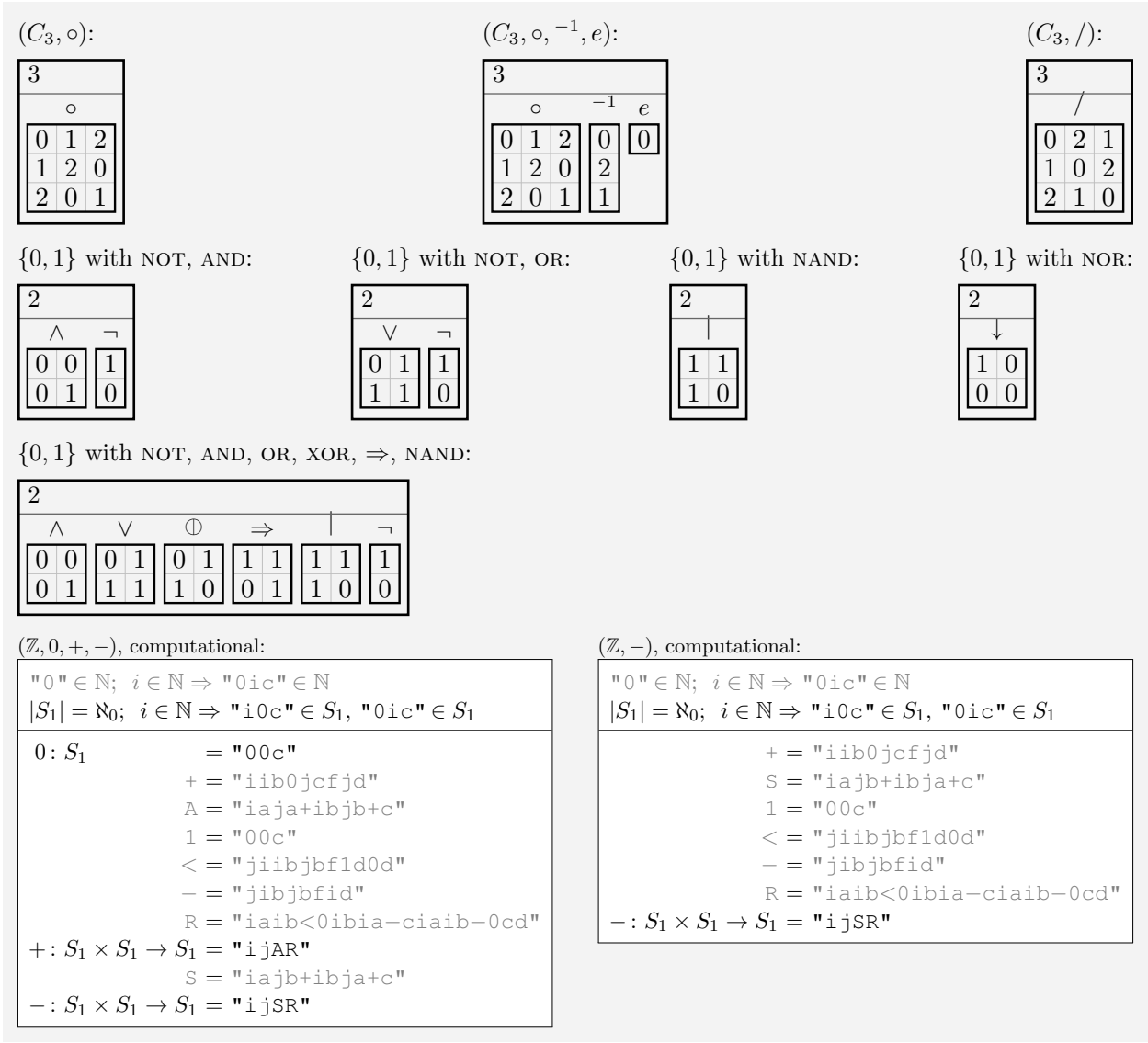


Figure 6: Equivalence by composition. Top row: three presentations of C_3, with the product alone, with product, inverse and identity ($a^{-1} = a \circ a$, $e = a \circ a \circ a$) and with division alone ($a \circ b = a/((b/b)/b)$). Middle rows: five presentations of the two-element Boolean algebra, each of whose functions is a composition of the others'. Bottom row: computational presentations of the integers with zero, addition and subtraction, and of the integers under subtraction alone ($0 = x - x$, $x + y = x - ((y - y) - y)$).

Model theory historically tends to call a presentation a "structure", sometimes identified up to isomorphism. As described in Section 1.2, we let "presentation" refer to a specific realization, and a "mathematical structure" to its equivalence class under isomorphism. However, it is interesting to further broaden the equivalence class definition, to also identify presentations that *generate each other via function composition*, because that is closer to the identification that many mathematicians make when they say that two presentations are "the same thing, just described

differently". Consider, for example, the first two presentations in the top row of Figure 6. The second one describes the cyclic group C_3, tabulating not only its binary (multiplication) function $f_1(g_1, g_2) = g_1 g_2$, but also its inverse function $f_2(g) = g^{-1}$ and identity $f_3 = e$. Is the first presentation, listing only the multiplication table f_1, not also describing the group C_3? Claiming that it does not would arguably go against standard mathematics terminology, since when finite groups are defined in the mathematics literature and on Wikipedia, they are normally described by merely presenting this multiplication table (known as their Cayley table). Moreover, the inverse and identity can be uniquely *defined* from the multiplication table alone, by composing the function f_1 with itself:

$$\begin{aligned} f_2(g) &= f_1(g,g) = g^2 = g^{-1}, &(1)\\ f_3 &= f_1(f_1(g,g),g) = g^3 = e. &(2)\end{aligned}$$

Indeed, for any finite group of order n, $g^{-1} = g^{n-1}$ and $e = g^n$. In other words, f_1 generates f_2 and f_3 via composition.

The rest of Figure 6 shows more such examples. The five shown presentations of Boolean functions on $\{0,1\}$ are all equivalent under composition, because they generate each other's generator functions. For example, the binary function $f_|$ (NAND) generates all of the others:

$$\begin{aligned} f_\neg(x) &= f_|(x,x), &(3)\\ f_\wedge(x,y) &= f_|(f_|(x,y), f_|(x,y)),\\ f_\vee(x,y) &= f_|(f_|(x,x), f_|(y,y)),\\ f_\Rightarrow(x,y) &= f_|(x, f_|(y,y)),\\ f_\oplus(x,y) &= f_|(f_|(x, f_|(x,y)), f_|(y, f_|(x,y))),\\ f_\downarrow(x,y) &= f_\neg(f_\vee(x,y)).\end{aligned}$$

They in fact have the *same compositional closure*: they all generate *all* Boolean functions of any number of variables. Working mathematicians typically refer to this simply as Boolean algebra, regardless of which presentation was used to specify it, which further supports including compositional closure in the equivalence definition.

The final example in Figure 6 shows two different presentations of integers: $(\mathbb{Z}, 0, +, -)$ and $(\mathbb{Z}, -)$, respectively. They are compositionally equivalent because $f_0 = f_-(x,x) = x - x = 0$ and $f_+(x,y) = x + y = x - ((y - y) - y)$.

In model theory and universal algebra, the compositional closure is called the *clone*, and equivalence under compositional closure is called "term equivalence". We will also refer to a clone as a *closed mathematical structure*. So a closed mathematical structure consists of abstract sets of elements and functions (the clone) [23, 24, 25, 26], closed under composition. We can think of it as all the functions on sets that are defined (via composition) by the presentation.

In our atlas, we ultimately choose to define mathematical structures up to isomorphism rather than up to clone isomorphism, because the latter sometimes lumps together two objects that mathematicians conventionally view as different. For example, there are two different groups of order 260 that have the same compositional closure [27] (see Appendix A): with our definitions, these two groups are different mathematical structures, but define the same closed mathematical structure.

As illustrated in Figure 7, closed mathematical structures generated by binary functions are partially ordered by inclusion, where each generates its children. In other words, the clone of a child is a subset of the clone of a parent. For example, for a single binary operation on 3 elements, there are $3^{3^2} = 19683$ different presentations, which form 3330 mathematical structures (magmas

up to isomorphism). 24 of them are also semigroups, but some generate each other, leaving only 18 clones. 15 of these are truly binary, 2 depend only on one variable and 1 is a constant. In this example, each class consists of a semigroup and its opposite (which may already be isomorphic to it), since $x \circ y$ and $y \circ x$ generate each other. More generally, classes can be quite large.

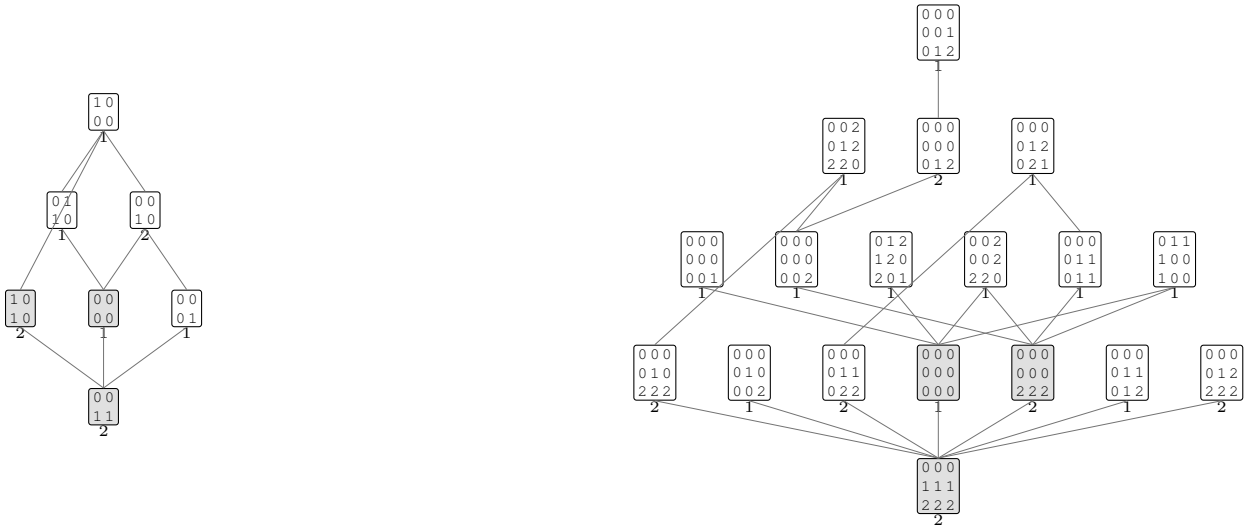


Figure 7: Hasse diagram of closed mathematical structures generated by a single binary operation on 2 elements (left) and by a semigroup on three elements (right), ordered by inclusion. Each generates its children. Numbers show the number of mathematical structures in each equivalence class.

3.3 Counting mathematical structures

Table 5 shows the number of mathematical structures with n elements that are models of various theories; most of these counts are listed in OEIS [12]. Monounary algebras are simply a single unary function $f\colon S \to S$. Finite distributive lattices are the same thing as finite frames. The rows are seen to scale quite differently: the group rows are relatively flat and arithmetic in nature, the semigroup rows grow roughly like n^{cn^2} and the magma row is $n^{n^2}/n!$ up to a factor tending to 1.

Table 6 compares the numbers of mathematical structures and closed mathematical structures with n elements, computed as described in Appendix A. The numbers of mathematical structures (the "str" columns) are known from the literature (cited in Table 5). The new entries are the numbers of closed mathematical structures (the "cl" columns). Some of these might not be in the existing literature, for example the 411 clones generated by a single binary operation on three elements.

The last column of the table shows n_{diff}, defined as the smallest n for which cl $<$ str, that is, for which two non-isomorphic models of the theory generate each other, as detailed in Appendix A. A "–" means that this never happens for any n.

n	1	2	3	4	5	6	7	8	Source
Monounary algebras	1	3	7	19	47	130	343	951	A001372
Rooted trees	1	1	2	4	9	20	48	115	A000081
Digraphs	2	10	104	3044	291 968	96 928 992	$1.1 \cdot 10^{11}$	$4.6 \cdot 10^{14}$	A000595
Graphs	1	2	4	11	34	156	1044	12 346	A000088
Preorders	1	3	9	33	139	718	4535	35 979	A001930
Equivalence relations	1	2	3	5	7	11	15	22	A000041
Total orders	1	1	1	1	1	1	1	1	one per n
Magmas	1	10	3330	178 981 952	$2.5 \cdot 10^{15}$	$1.4 \cdot 10^{25}$	$5.1 \cdot 10^{37}$	$1.6 \cdot 10^{53}$	A001329
Semigroups	1	5	24	188	1915	28 634	1 627 672	3 684 030 417	A027851
Bands	1	3	10	46	251	1682	13 213	119 826	A058112
Monoids	1	2	7	35	228	2237	31 559	1 668 997	A058129
Commutative monoids	1	2	5	19	78	421	2637	20 486	A058131
Groups	1	1	1	2	1	2	1	5	A000001
Abelian groups	1	1	1	2	1	1	1	3	A000688
Semilattices	1	1	2	5	15	53	222	1078	A006966(n+1)
Semirings	1	2	6	40	295	3246	59 314	–	arXiv:2507.03709
Rings (with 1)	1	1	1	4	1	1	1	11	A037291
Commutative rings	1	1	1	4	1	1	1	10	A127707
Integral domains	0	1	1	1	1	0	1	1	prime powers
Division rings	0	1	1	1	1	0	1	1	prime powers
Fields	0	1	1	1	1	0	1	1	prime powers
Lattices	1	1	1	2	5	15	53	222	A006966
Distributive lattices	1	1	1	2	3	5	8	15	A006982
Boolean algebras	1	1	0	1	0	0	0	1	powers of 2

Table 5: Mathematical structures with n elements

Theory	$n=2$ str	cl	$n=2$ str_{tb}	cl_{tb}	$n=3$ str	cl	str_{tb}	cl_{tb}	$n=4$ str	cl	str_{tb}	cl_{tb}	n_{diff}
Monounary algebras	3	3			7	7			19	19			6
Rooted trees	1	1			2	2			4	4			–
Magmas	10	7	5	4	3330	411	3317	404	178 981 952	–[3]	178 981 915	–	2
Semigroups	5	4	2	2	24	18	19	15	188	126	179	121	2
Bands	3	2	1	1	10	6	8	5	46	26	44	25	2
Monoids	2	2	2	2	7	6	7	6	35	27	35	27	3
Commutative monoids	2	2	2	2	5	5	5	5	19	19	19	19	–
Groups	1	1	1	1	1	1	1	1	2	2	2	2	24–260
Abelian groups	1	1	1	1	1	1	1	1	2	2	2	2	–
Semilattices	1	1	1	1	2	2	2	2	5	5	5	5	–
Semirings	2	2	2	2	6	6	6	6	40	38	40	38	4
Rings (with 1)	1	1	1	1	1	1	1	1	4	4	4	4	16
Commutative rings	1	1	1	1	1	1	1	1	4	4	4	4	16
Integral domains	1	1	1	1	1	1	1	1	1	1	1	1	–
Division rings	1	1	1	1	1	1	1	1	1	1	1	1	–
Fields	1	1	1	1	1	1	1	1	1	1	1	1	–
Lattices	1	1	1	1	1	1	1	1	2	2	2	2	5
Boolean algebras	1	1	1	1	0	0	0	0	1	1	1	1	–

Table 6: Number of mathematical structures "str" (presentations up to isomorphism) and closed mathematical structures (clones) "cl" for small n; the second pair of columns restricts to ones whose binary operations are all truly binary by depending on both input variables, so it does not apply to the two unary rows. The last column gives the smallest n for which cl < str, with a dash if this never happens and a range if the value is not yet determined; see Appendix A.

3.4 Determining compositional equivalence in practice

There is an interesting analogy between mathematical theories and closed mathematical structures. Both are specified by a list of "base cases" (axioms and generating functions, respectively; the list is often finite) and rules for generating new ones by combining old ones (via deduction and composition, respectively). Although the sets of theorems and functions are generally infinite, equivalence between two theories/presentations can be proven by showing that the base cases of one formal system/presentation can be generated from the axioms/functions of the other. For two finite presentations there is a halting algorithm determining whether they generate each other; one never needs to explore the whole infinite clone, merely the generated functions up to the highest arity d occurring among the generators of the two presentations. Consider first a single set with n elements. To check whether a generator of arity $k \leq d$ of one presentation equals some composition of functions from the other presentation, we only need to consider compositions with k input variables, and these compute at most $n^{n^k} \leq n^{n^d}$ different functions, the number of all k-ary functions on a set with n elements. So we start from the k projections $(x_1, \ldots, x_k) \mapsto x_i$ and iteratively compose the functions generated so far with the generators until no new function is created. A constant counts as generated if some generated function is constant with that value. For presentations with several sets the argument is the same, with each variable and each function

[3]The magma clone counts for $n = 4$ are left open. There are 178 981 952 magmas on four elements up to isomorphism (178 981 915 of them truly binary), and each needs a canonical form of its clone, whose binary part can hold up to $4^{16} \approx 4.3 \cdot 10^9$ functions; with the closure algorithm used for the other rows a typical magma takes hours and tens of gigabytes, so the row would cost far more than 10^5 CPU-years. Completing it needs a different method, such as Rosenberg's completeness criterion for the Sheffer operations and a relational invariant for the rest.

carrying the sets of its inputs and of its output, and with compositions allowed only when these match; the number of functions with a given input and output profile is again finite. This is how the "cl" numbers in Table 6 were computed (described in greater detail in Appendix A). For reasons that are not yet fully understood, most of the functions used in science and mathematics are unary and binary, so $d = 2$ often suffices in practice.

3.5 Relation to physics: What should be considered equivalent?

Historically, mathematics has often been developed to model physical systems. Considering a physical system as described by a mathematical structure (or optionally even being one, as controversially explored in [21]) provides an interesting perspective both on infinite structures and on which presentations should be considered equivalent, as summarized in Table 7.

Equivalence	**Two presentations are equivalent if...**	**An observer who...**
Isomorphic	they differ only in how elements are labeled	cannot see element labels
Compositional	each one's functions are compositions of the other's	can also compose measurements
Computational	the same holds once equality tests "=" and branching "?" are allowed	can also tell whether two measurement results agree and act on it
Definitional	each one's functions can be defined from the other's using $\forall$ and $\exists$	can also search through all elements

Table 7: Four successively broader notions of equivalence between presentations. Each row also allows everything in the rows above it. The last column describes an observer for whom presentations equivalent in that sense are indistinguishable.

Consider an observer who inhabits a physical world described by a mathematical structure, and is able to perform certain measurements and reasoning steps. The world is made up of elements (examples may include points, magnetic field vectors, *etc.*) and various relations that hold between them. For example, in classical physics, three nearby points in space may satisfy the relation that they lie on a line, or one magnetic field vector may be 2 times another. Which relations or other functions would the observer consider well-defined and inherent in the physical system? The answer depends on what she is able to do, and Table 7 lists four successively more capable observers. The more she can do, the more functions she can obtain from the ones that nature hands her, and the more presentations therefore become equivalent to her. Let us go through the four rows in turn.

Isomorphism There is not a shred of evidence that notation matters in physics. In our physical world, nature never seems to reveal specific coordinate systems or other entity labels to us. For example, we do not physically observe a point in space being labeled $\mathbf{r}$, or our universe having any labeled x-axis. This motivates describing the physical system as a mathematical structure (an equivalence class of isomorphic presentations), since those unobservable labels are the only difference between the presentations in the equivalence class.

Compositional equivalence If the observer is able to measure certain quantities (*e.g.*, distances between points) and use them as inputs determining subsequent experimental measurements, then

she is able to *compose* the functions that nature gives her access to. This motivates describing the physical system using a closed mathematical structure.

Computational equivalence Suppose that the observer can also tell whether two measurement results agree, and let the answer decide what she does next. Then she can also define functions by cases. This amounts to adding two functions to the presentation for each of its sets S:

1. the equality test $=: S \times S \to \{0,1\}$, which returns 1 if its two inputs are the same element and 0 otherwise, and
2. the "if statement" $?: \{0,1\} \times S \times S \to S$, which returns its second input if the first is 1 and its third input otherwise (analogously to MLISP's `d`-command).

Definitional equivalence Finally, if the observer can also quantify over a full set of elements, then even more presentations become equivalent. This is what model theory calls *definitional equivalence*. As an example, consider the integers with addition alone, $(\mathbb{Z}, +)$, and the integers with addition and zero, $(\mathbb{Z}, +, 0)$. There is no procedure for producing zero by merely adding different integers together (there would be if $-$ were also defined, as $x - x$ for any x). But an observer able to search through all integers can identify and point to 0 as the unique element z satisfying $x + z = x$ for all x.

For a presentation with a single finite set, this fourth level has a simple description in terms of symmetries, *i.e.*, relabelings of the elements that leave all functions of the presentation unchanged. If some symmetry exchanges two elements, then no property whatsoever can distinguish them. Conversely, on a finite set, anything that all symmetries preserve can be defined using quantifiers [28, 23]: one says "there is a relabeling of the elements that respects all the lookup tables and carries one of the listed cases to this one". Two presentations on the same n elements are therefore equivalent in this sense exactly when they have the same symmetries, and two arbitrary ones exactly when their symmetry groups are conjugate subgroups of the permutation group S_n. For example, a total order on n elements has no symmetries, since its smallest element, its second smallest element and so on can each be singled out. It is therefore equivalent to a set of n elements that each have their own name (a constant), and indeed to every other presentation without symmetries. Conversely, given a presentation where each element is labeled by a constant function, the order relation $\leq$ can be defined via nested if-statements. At this level, the number of different structures with n elements thus drops to the number of conjugacy classes of subgroups of S_n, which is merely 1, 2, 4, 11 and 19 for $n = 1, \ldots, 5$.

In the physical universe we inhabit, intelligent observers can arguably be approximated as "computational", with the caveat that their ability to measure and compute is limited. In contrast, fully "definitional" observers appear nonexistent because various forms of locality deny us access to more than a small fraction of what physically exists.

3.6 Infinity in physics and mathematics

How large is the set of mathematical structures? For starters, the number of finite mathematical structures is infinite but countable, since any presentation tableau can be described in a finite number of bits of information and can therefore be encoded as a natural number whose binary representation encodes the presentation bit-string. The same clearly holds for infinite *computable* structures, whose functions are defined by computer programs as in our presentation tableaus. But what about structures whose relations cannot all be defined by halting computations? To sharpen this question, consider these four candidate mathematical structures, which each include exactly

one function on a single set (the set of natural numbers for the first three structures and the set of real numbers for the fourth one):

1. $P(n)$: Given a natural number n, $P(n)$ denotes the smallest prime greater than n.
2. $T(n)$: Given a natural number n, $T(n)$ denotes the smallest twin prime greater than n (a twin prime is a prime p for which $p+2$ is also prime).
3. $H(m,n)$: Given two natural numbers m and n, $H(m,n)$ equals 0 if the mth computer program on an enumerated list of all computer programs will keep running forever when fed the bits of n as input, and $H(m,n)$ equals 1 if the program instead halts after a finite number of steps.
4. $D(x)$: Given a real number x, $D(x)$ denotes $2x$.

Which of these four should qualify as a valid mathematical structure, and which get disqualified for being insufficiently well-defined? The first function, $P(n)$, is a piece of cake: it is easy to write a program that starts checking whether the numbers following n are prime and stops as soon as it finds one, and we are guaranteed that this program will halt after a finite number of steps because Euclid proved that there are infinitely many primes. So $P(n)$ is an example of what we call a *computable function*.

The second function, $T(n)$, is trickier: it is again easy to write a program that checks each number following n to see whether it is a twin prime, but if you plug in a number n greater than $2996863034895\times 2^{1290000}-1$ (the largest twin prime known as of writing this), there is no guarantee that the program will ever stop and deliver an answer, because despite the best efforts of our most talented mathematicians, we still do not know whether there are infinitely many twin primes. So for now, we do not know whether $T(n)$ is defined for all n, and it is debatable whether a mathematical structure containing such a sloppily specified function qualifies as well-defined.

The third function, $H(m,n)$, is even more nefarious: Alonzo Church and Alan Turing established that there is *no* program that can compute $H(m,n)$ for arbitrary input numbers m and n in a finite number of steps, so $H(m,n)$ is an example of what we call an *uncomputable function*. In other words, there is no program that can determine which other programs will eventually halt. Of course, any given program will or will not halt, but the catch is, just as with the twin primes, that you might have to wait forever to find out. The Church–Turing discovery of uncomputable functions is closely related to the discovery by Kurt Gödel that some statements of arithmetic are undecidable, meaning that they can neither be proved nor disproved in a finite number of steps in a given formal system.

The fourth function, $D(x)$, is even worse: whereas $H(m,n)$ is computable for some but not all inputs, $D(x)$ is uncomputable for almost all inputs (for all real numbers except a subset of Lebesgue measure zero). The reason is that a generic real number has infinitely many decimals that go on without any discernible pattern, and therefore requires an infinite number of bits of information to describe: the problem is not merely performing an infinite number of computational steps on a finite input as for the H-example, but that of inputting and outputting an infinite amount of information.

So should a mathematical structure be considered well-defined even if it contains a function such as H or D that cannot be evaluated even on an arbitrarily powerful computer? I am not going to answer this question here, but find it interesting that closely related issues have been hotly debated among mathematicians without any reference to physics. According to the finitist school of mathematicians, which included Leopold Kronecker, Hermann Weyl and Reuben Goodstein, a mathematical object does not exist unless it can be constructed from whole numbers in

a finite number of steps. With this criterion, an existing mathematical structure can contain rational numbers and algebraic numbers, but not generic real numbers or other generic members of continuous/uncountable sets.

Turning now to physics, how well-defined do mathematical structures need to be to correspond to something in the physical world? The above examples suggest a hierarchy of possible answers:

1. Finite structures
2. Computable structures (defined by halting computations)
3. Structures defined by computations that might not halt
4. Uncountable structures (involving *e.g.* real numbers)

Although physics textbooks are full of uncountable structures, it is striking that physics has thus far provided no solid experimental evidence that anything infinite (let alone uncountably infinite) exists physically. The contrast is particularly noteworthy for real numbers, where we cannot define generic ones (or functions thereof) computationally. Most physicists rarely bother distinguishing between a real number and a `float64` variable; indeed, computer languages including Fortran and Pascal even refer to the float datatype as "real". This is probably because *the distinction never seems to matter*. A `float64` variable represents a number of the form $x = \pm a \times 2^b$, with 52 bits for a and 11 for b, and can approximate any number between 10^{-308} and 10^{308} to about 15 significant digits. In contrast, physicists never need such a large dynamic range in practice, since the largest accessible distance (the diameter of our observable universe) is "only" about 10^{62} times the smallest meaningful distance (the Planck length), and there are "only" about 10^{88} particles in our observable universe. Indeed, the popular but controversial holographic principle [29] suggests that our universe can store only about 10^{123} bits of information. No measurement in physics has reached a relative precision better than about one part in 10^{22} (length changes in gravitational wave detectors), for which `float128` still suffices. So when physicists casually talk about computing "real numbers", usually they *de facto* mean not members of the uncountable set $\mathbb{R}$, but something that they can have their computer approximate to any reasonable precision they desire.

Whereas many early physicists assumed that the physical laws they discovered were exact, modern views tend to align more with Steven Weinberg's [30] humbler stance that all of today's physics theories are probably mere effective theories, approximations of something deeper and more accurate. Some have even speculated that the deepest theory does not involve the continuum or any form of infinity [31, 32, 33, 21].

Approximating something discrete as continuous is often convenient, greatly simplifying calculations. For example, consider the air in front of you. Keeping track of the positions and speeds of octillions of atoms would be hopelessly complicated. But if you ignore the fact that air is made of atoms and instead approximate it as a continuum, a smooth substance that has a density, pressure and velocity at each point, you find that this idealized air obeys a beautifully simple partial differential equation that explains many important phenomena: how sound waves propagate through air, how winds work, *etc.* Yet despite all that convenience, air is not truly continuous. Many physicists guess that it may be the same way for space, time and all the other seemingly continuous building blocks of our physical world, even though fully workable non-continuous alternatives are still lacking. Humility aside, this guess is appealing because some of the thorniest open problems in modern physics rest on assumptions about infinity and the continuum, from the measure problem in eternal inflation [34, 35] to black hole and Big Bang singularities [36, 37]. On the theoretical physics side, it will be interesting to see whether discrete or finite theories are found that match the experimental successes of today's leading physics theories. On the experimental side, it will be interesting to

see whether any observations can determine whether any assumed physical continuum satisfies the least-upper-bound property.

4 Conclusions

We have introduced MLISP, a tiny Turing-complete language, and used it to define a family tree of mathematical theories and associated models. Emphasis was placed on consistently applying a standardized terminology throughout.

We used a consistent four-way "map versus territory" terminology: formal systems and theories are descriptions; presentations and mathematical structures are what they describe. Theories and structures are both equivalence classes that strip away notation. Every structure is defined by a presentation tableau, with lookup tables for finite sets and short halting programs for countably infinite ones. This standardizes the side of the family tree where the literature has converged least. For finite presentations on a set with n elements, equivalence is decided by a halting algorithm that never explores more than the n^{n^d} functions of the highest arity d among the generators. The definitions, key theorems, model relations and the correctness of MLISP are checked in Lean. Lean checks MLISP rather than defines the presentations, which keeps the trusted base small.

This work naturally raises many open questions. Here are a few examples:

- What is the lowest order (between 24 and 260) at which two different groups are compositionally equivalent?
- How can the closed mathematical structures of various types and sizes be counted more efficiently? These integer sequences arguably deserve to be added to OEIS like their non-closed counterparts.
- Is there generally a canonical normal form or "simplest" presentation defining a mathematical structure or a closed mathematical structure?
- The partial tree presented is biased toward what appears in university courses, Wikipedia and Mathlib; is AI now capable of automatically extending it within, *e.g.*, the τ Ceti project [38]?

As AI proposes and proves ever more mathematical theorems, compact human-readable maps of what the theorems are about become more valuable. This project is a small step in that direction, which will hopefully inspire others to continue further.

Acknowledgements: The author would like to thank Meia Chita-Tegmark for encouragement with this long-running project. Codex and Claude Code assisted with plots, tables, Lean proofs, Appendix A and proofreading. The mathematics tree was organized without AI, starting in 1996. The author created MLISP and the programs in Table 3 around 2018.

References

[1] Current and former Caltech mathematicians. Open letter about the mathathon. Proofs and Prompts, https://proofsandprompts.com/2026/09/10/open-letter-about-the-mathathon/, September 2026. Open letter of 10 September 2026 opposing the Caltech Mathathon, a hackathon on open research problems with LLMs; 771 signatories at publication.

[2] Twenty-eight Fields Medalists. A severe misalignment of AI in mathematics. Declaration, https://mathandai.org/, September 2026. Signed by 28 Fields Medal recipients, including Pierre Deligne, Peter Scholze, Maryna Viazovska, Martin Hairer, Cédric Villani, Manjul Bhargava and Terence Tao; published 11 September 2026 and open to further signatures.

[3] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pages 367–381. ACM, 2020.

[4] Max Tegmark. Is "the theory of everything" merely the ultimate ensemble theory? *Annals of Physics*, 270(1):1–51, 1998. arXiv:gr-qc/9704009.

[5] Rudolf Carnap and Friedrich Bachmann. Über Extremalaxiome. *Erkenntnis*, 6:166–188, 1936.

[6] Georg Schiemer. Carnap's structuralist thesis. In Erich H. Reck and Georg Schiemer, editors, *The Prehistory of Mathematical Structuralism*, pages 383–420. Oxford University Press, Oxford, 2020. https://doi.org/10.1093/oso/9780190641221.003.0015.

[7] Nicolas Bourbaki. The architecture of mathematics. *The American Mathematical Monthly*, 57(4):221–232, 1950.

[8] David Hilbert. Axiomatisches Denken. *Mathematische Annalen*, 78:405–415, 1918.

[9] Saunders Mac Lane. *Mathematics: Form and Function*. Springer, New York, 1986.

[10] Leo Corry. *Modern Algebra and the Rise of Mathematical Structures*, volume 17 of *Science Networks*. Birkhäuser, Basel, 1996.

[11] Timothy Gowers, June Barrow-Green, and Imre Leader, editors. *The Princeton Companion to Mathematics*. Princeton University Press, Princeton, 2008.

[12] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, 2026. https://oeis.org.

[13] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021.

[14] The Rocq Development Team. The Rocq prover (formerly Coq), reference manual. https://rocq-prover.org/, 2025.

[15] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, Berlin, 2002.

[16] Errett Bishop. *Foundations of Constructive Analysis*. McGraw-Hill, New York, 1967.

[17] Leonardo de Moura. Postmortem for kernel soundness bug #14576. https://leodemoura.github.io/blog/2026-8-1-postmortem-for-kernel-soundness-bug-14576/, August 2026.

[18] Neil Calkin and Herbert S. Wilf. Recounting the rationals. *The American Mathematical Monthly*, 107(4):360–363, 2000.

[19] Errett Bishop and Douglas Bridges. *Constructive Analysis*, volume 279 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin, 1985.

[20] N. J. Wildberger. *Divine Proportions: Rational Trigonometry to Universal Geometry*. Wild Egg, Sydney, 2005.

[21] Max Tegmark. *Our Mathematical Universe: My Quest for the Ultimate Nature of Reality*. Knopf, New York, 2014.

[22] Jean Nicod. A reduction in the number of the primitive propositions of logic. *Proceedings of the Cambridge Philosophical Society*, 19:32–41, 1917.

[23] Wilfrid Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1993.

[24] Emil L. Post. *The Two-Valued Iterative Systems of Mathematical Logic*, volume 5 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, 1941.

[25] Philip Hall. Unpublished lectures on universal algebra, 1958. The term "clone" for a composition-closed set of operations containing the projections is attributed to Hall by Cohn [26].

[26] Paul M. Cohn. *Universal Algebra*. Harper & Row, New York, 1965.

[27] Keith A. Kearnes and Ágnes Szendrei. Groups with identical subgroup lattices in all powers. *Journal of Group Theory*, 7(3):385–402, 2004.

[28] Marc Krasner. Une généralisation de la notion de corps. *Journal de Mathématiques Pures et Appliquées*, 17:367–385, 1938.

[29] Ning Bao, Sean M. Carroll, and Ashmeet Singh. The Hilbert Space of Quantum Gravity Is Locally Finite-Dimensional. *Int. J. Mod. Phys. D*, 26(12):1743013, 2017. arXiv:1704.00066.

[30] Steven Weinberg. What is quantum field theory, and what did we think it is? In Tian Yu Cao, editor, *Conceptual Foundations of Quantum Field Theory*, pages 241–251. Cambridge University Press, Cambridge, 1999. arXiv:hep-th/9702027.

[31] Nicolas Gisin. Indeterminism in Physics, Classical Chaos and Bohmian Mechanics: Are Real Numbers Really Real? *Erkenntnis*, 86(6):1469–1481, 2021. arXiv:1803.06824.

[32] Flavio Del Santo and Nicolas Gisin. Physics without determinism: Alternative interpretations of classical physics. *Phys. Rev. A*, 100(6):062107, 2019. arXiv:1909.03697.

[33] George F. R. Ellis, Krzysztof A. Meissner, and Hermann Nicolai. The physics of infinity. *Nature Phys.*, 14(8):770–772, 2018.

[34] Andrei Linde. A brief history of the multiverse. *Rept. Prog. Phys.*, 80(2):022001, 2017. arXiv:1512.01203.

[35] Arthur Hebecker. *Naturalness, String Landscape and Multiverse: A Modern Introduction with Exercises*, volume 979 of *Lecture Notes in Physics*. Springer, March 2021. arXiv:2008.10625.

[36] José M. M. Senovilla and David Garfinkle. The 1965 Penrose singularity theorem. *Class. Quant. Grav.*, 32(12):124008, 2015. arXiv:1410.5226.

[37] Karen Crowther and Sebastian De Haro. Four Attitudes Towards Singularities in the Search for a Theory of Quantum Gravity. In Antonio Vassallo, editor, *The Foundations of Spacetime Physics: Philosophical Perspectives*, pages 223–250. Routledge, 2022. arXiv:2112.08531.

[38] Tau Ceti Project. Tau Ceti: an AI-built Lean library of formalized mathematics downstream of Mathlib. https://github.com/TauCetiProject/TauCeti, 2026.

[39] S. V. Jablonskij. Functional constructions in a k-valued logic. *Trudy Mat. Inst. Steklov.*, 51:5–142, 1958. In Russian.

[40] Jerzy Słupecki. Kryterium pełności wielowartościowych systemów logiki zdań (A criterion of fullness of many-valued systems of propositional logic). *Comptes Rendus des Séances de la Société des Sciences et des Lettres de Varsovie, Classe III*, 32:102–110, 1939.

[41] László Zádori. Categorical equivalence of finite groups. *Bulletin of the Australian Mathematical Society*, 56(3):403–408, 1997.

A Counting closed mathematical structures

This appendix describes how the entries of Table 6 were computed. The str columns reproduce numbers known from the literature (see Table 5); some of the cl entries may be new. The str and cl counts were obtained by exhaustive computer enumeration; Section A.6 explains the last column, which also rests on proofs and on searches at larger n. The Python scripts that produced the numbers are named below in parentheses.

A.1 What is counted

Fix a set $S = \{0, \dots, n-1\}$. A presentation of one of the theories in the table is a list of constants, unary functions $S \to S$ and binary functions $S \times S \to S$ satisfying the axioms of the theory.

- **str** is the number of presentations up to isomorphism, *i.e.*, up to relabeling the elements of S by a permutation.
- **cl** is the number of distinct clones that these presentations generate, again up to relabeling. Two presentations are counted once if, after some relabeling, every function of each is a composition of functions of the other.
- The columns $\mathbf{str}_{\mathrm{tb}}$ and $\mathbf{cl}_{\mathrm{tb}}$ repeat the two counts for the presentations all of whose binary functions are *truly binary*, *i.e.*, depend essentially on both arguments: $f(x, y) \neq f(x, z)$ for some x, y, z and $f(x, y) \neq f(z, y)$ for some x, y, z.

A.2 Step 1: enumerating the presentations

For each n, we generate every labeled presentation of each theory and keep one representative per isomorphism class. The canonical form of a presentation is the lexicographically smallest of its $n!$ relabelings (the tables of all its functions written out as one tuple), and two presentations are isomorphic exactly when their canonical forms agree. The number of distinct canonical forms is str. The labeled presentations are generated as follows.

- **Monounary algebras:** all n^n maps $f : S \to S$. **Rooted trees:** those maps with exactly one fixed point that every element reaches under iteration of f.
- **Magmas:** all n^{n^2} multiplication tables (16 for $n = 2$ and 19 683 for $n = 3$).
- **Semigroups:** all associative tables, found by backtracking: the table is filled in one entry at a time and a branch is abandoned as soon as an instance of $(xy)z = x(yz)$ with all entries known fails. This gives 8, 113 and 3492 labeled semigroups for $n = 2, 3, 4$ (`semigroups.py`).
- **Bands, semilattices, monoids, commutative monoids, groups, Abelian groups:** these are selected from the list of semigroups by testing the additional axioms (idempotency; idempotency and commutativity; existence of an identity; and so on). A monoid is stored with its identity as a constant, and a group with its identity and its inverse function, so that these count as generators in Step 2.
- **Semirings:** for every labeled commutative monoid $(S, +, 0)$ from the list above and every choice of the element 1, all multiplication tables are generated by backtracking subject to associativity, 1 being an identity, 0 being absorbing and distributivity over $+$ (`semirings.py`). **Rings** are the semirings in which every element has an additive inverse; **commutative rings, integral domains, division rings** and **fields** are selected from the rings by testing their axioms.
- **Lattices:** for every semilattice $(S, \wedge)$ from the list above, with its order $x \leq y \Leftrightarrow x \wedge y = x$, the join $x \vee y$ is computed as the least upper bound of x and y; the semilattice is kept, as the pair $(\wedge, \vee)$, if all joins exist. **Boolean algebras** are the lattices that are distributive and

in which every element has a complement; they are stored with 0, 1 and the complement as additional generators.

A.3 Step 2: computing the clone of a presentation

Every function in these presentations has at most two arguments. In that case two presentations on S generate the same clone if and only if they generate the same set of *binary* functions: each generator of one is then among the binary functions generated by the other (a unary function or a constant being regarded as a binary function that ignores one or both of its arguments), and compositions of generated functions are generated. It therefore suffices to compute, for each presentation, the set B of all binary functions that it generates.

A binary function on S is an $n \times n$ table, so B is a set of tables. We compute it as a closure (`clones.py`):

1. Start with the table X of the function $(x, y) \mapsto x$, the table Y of $(x, y) \mapsto y$ and one constant table for each constant of the presentation.
2. Apply every generator entrywise to the tables found so far: a unary generator u turns a table F into the table with entries $u(F_{ij})$, and a binary generator g turns a pair of tables F, G into the table with entries $g(F_{ij}, G_{ij})$. Add every table not seen before.
3. Repeat step 2 until no new table appears.

This halts because there are only n^{n^2} tables, and it yields exactly B: every table produced is the table of a composition of generators applied to x and y, and every such composition is produced, by induction on its length.

Three shortcuts avoid the closure in the cases where B is large:

- **All functions, $n = 3$:** A single binary operation on three elements generates all functions if and only if it belongs to none of the 18 maximal clones on three elements, which were classified by Jablonskij [39]. We test each of the 19 683 operations for membership in these 18 clones once (`sheffer3.py`); 3774 of them belong to none and are assigned the clone of all functions without running the closure.
- **All functions, $n \geq 3$:** By Słupecki's criterion [40], a presentation generates all functions if it generates all n^n unary functions and has a generator that is surjective and truly binary. The unary functions generated are computed by the same closure applied to tables of unary functions, which is cheap.
- **Semirings and their descendants, $n = 4$:** By distributivity, every term of a semiring is a sum of products of variables and constants. We therefore first close $\{x, y, 0, 1\}$ under multiplication and then close the result under addition; this gives the same set B as the general closure but takes far fewer steps (`ringlike_clones.py`). A ring, division ring or field generates the same clone as its semiring, because $-x$ is a sum of copies of x and, in a field with q elements, the inverse (with the convention $0^{-1} = 0$) is $x^{-1} = x^{q-2}$ for $q > 2$ and $x^{-1} = x$ for $q = 2$. For $n = 2$ and 3 we verified that this method and the general closure produce identical sets.

A.4 Step 3: counting clones up to relabeling

A permutation σ of S maps a table F to the table F^σ with $F^\sigma(x, y) = \sigma(F(\sigma^{-1}x, \sigma^{-1}y))$, and maps the set B to $B^\sigma = \{F^\sigma : F \in B\}$. The canonical form of B is the lexicographically smallest of the $n!$ sorted lists B^σ. Two presentations generate the same closed mathematical structure exactly when the canonical forms of their sets B coincide, so cl is the number of distinct canonical forms

occurring in a row. For $\mathrm{cl}_{\mathrm{tb}}$, only the presentations all of whose binary generators are truly binary are included.

A.5 Checks

The str columns reproduce the published sequences listed in Table 5 for every row where one exists. An operation and its opposite $x \circ' y = y \circ x$ always generate each other, so the classes of presentations generating the same clone are unions of classes under isomorphism or anti-isomorphism. For semigroups, the cl entries $4, 18, 126$ for $n = 2, 3, 4$ equal the published numbers of semigroups up to isomorphism or anti-isomorphism (OEIS sequence A001423 [12]); for bands $(2, 6, 26)$ and monoids $(2, 6, 27)$ they equal the corresponding numbers, which we computed ourselves. Since the counts agree, there are no further identifications: for $n \leq 4$ the only distinct associative operations generating each other are an operation and its opposite. For magmas with $n = 3$, the 3330 isomorphism classes give 411 clones, of which the clone of all functions accounts for 629 classes. The magma entries for $n = 4$ were not computed, for the reason given in the footnote to the table.

A.6 When compositional closure makes a difference

In many rows of Table 6, cl equals str for every n shown. The last column of the table records the smallest n, denoted n_{diff}, for which cl $<$ str, *i.e.*, for which two non-isomorphic models of the theory generate each other. Throughout, only finite models are considered. For some theories this never happens, and for others the equality is a coincidence of small n. In each case the argument is the same: list the binary functions that a model generates, and ask which of them could be the operation of another model of the same theory.

Theories where it never makes a difference

- **Commutative monoids** $(M, +, 0)$: The binary functions generated are $(x, y) \mapsto ax + by$, where x and y range over the elements of M and a, b are natural numbers; here ax denotes x added to itself a times. Suppose that $x \circ y = ax + by$ is another commutative monoid operation, with identity e. The only constant that the monoid generates is 0, so $e = 0$. Then $x \circ 0 = x$ gives $ax = x$ for all x, and $0 \circ y = y$ gives $by = y$ for all y, so $x \circ y = x + y$. A commutative monoid thus generates no commutative monoid other than itself.
- **Abelian groups:** The same argument applies, with a and b integers.
- **Semilattices:** The only binary functions generated by $\wedge$ are x, y and $x \wedge y$. For $n \geq 2$ only the last is commutative, and for $n = 1$ there is only one semilattice.
- **Rooted trees:** A unary function f generates exactly its iterates $x, f, f^2, \ldots$, so f and g generate each other only if each is an iterate of the other. Unless $f = g$, this forces $f = f^m$ for some $m > 1$. A rooted tree with $f = f^m$ is a star, in which every node points to the root, and all iterates of a star are equal to it.
- **Integral domains, division rings, fields and Boolean algebras:** For every finite n there is at most one of each with n elements, so there is nothing to identify.

Theories where it makes a difference from the start A non-commutative operation $x \circ y$ generates its opposite $y \circ x$. This accounts for all identifications in the semigroup, band and monoid rows for $n \leq 4$: the first n with a model that is not isomorphic to its opposite is $n = 2$ for semigroups and bands and $n = 3$ for monoids. Magmas ($n_{\mathrm{diff}} = 2$) and semirings ($n_{\mathrm{diff}} = 4$) also have identifications of other kinds.

Theories where the difference appears only later

- **Lattices** ($n_{\text{diff}} = 5$): The binary functions generated by $\wedge$ and $\vee$ are x, y, $x \wedge y$ and $x \vee y$, so the only other lattice that a lattice generates is its dual, obtained by exchanging $\wedge$ and $\vee$. Hence cl counts lattices up to duality. Every lattice with at most four elements is isomorphic to its dual; of the five lattices with five elements, two are each other's duals, giving four clones.
- **Monounary algebras** ($n_{\text{diff}} = 6$): If two non-isomorphic functions f and g are iterates of each other, then $f = f^m$ for some $m > 1$, so f permutes its image: it consists of cycles with further elements (leaves) attached directly to cycle elements, and $g = f^k$ runs through the same cycles in a different order. Usually g is then isomorphic to f. The smallest exception has six elements: a cycle $0 \to 1 \to 2 \to 0$ with two leaves attached to 0 and one leaf attached to 1. Here f^2 runs through the cycle backwards and $(f^2)^2 = f^4 = f$, so f and f^2 generate each other, but they are not isomorphic, because the numbers of leaves met along the direction of the cycle are $(2, 1, 0)$ for f and $(1, 2, 0)$ for f^2. An enumeration of all n^n functions confirms that this is the first exception: cl = str $= 1, 3, 7, 19, 47$ for $n \leq 5$, whereas str = 130 and cl = 129 for $n = 6$ (`monounary_clones.py`).
- **Rings** ($n_{\text{diff}} = 16$): A ring generates its opposite ring, with multiplication $x \cdot' y = yx$, so cl $<$ str as soon as some ring is not isomorphic to its opposite. The ring of triples (a, b, c) with $a \in \mathbb{Z}_4$ and $b, c \in \mathbb{Z}_2$, with componentwise addition and the product

$$(a, b, c)(a', b', c') = (aa',\ \bar{a}b' + bc',\ cc'),$$

  where $\bar{a}$ denotes a reduced modulo 2 (the upper triangular matrices $\left(\begin{smallmatrix} a & b \\ 0 & c \end{smallmatrix}\right)$), has 16 elements and is such a ring. For an element x, let $r(x)$ be the number of y with $xy = 0$ and $l(x)$ the number of y with $yx = 0$. Over the 16 elements, r takes the values $1, 2, 4, 8, 16$ respectively $4, 4, 3, 4, 1$ times, whereas l takes them $4, 2, 6, 3, 1$ times; an isomorphism with the opposite ring would exchange the two lists (`ring_invariants.py`). For $n \leq 15$ nothing is identified. For $5 \leq n \leq 15$, the only n with more than one ring with identity are 8 (eleven rings), 9 (four) and 12 (four), and for each of these n the rings are distinguished by quantities that two rings generating each other must share: the number of constants generated, the number of unary functions generated, the number of subrings and the numbers of ideals and of minimal ideals (`ring_invariants.py`).
- **Commutative rings** ($n_{\text{diff}} = 16$): A commutative ring is its own opposite, so the mechanism above is not available, and the computation just described shows that nothing is identified for $n \leq 15$. For $n = 16$, however, the two rings

$$R_1 = \mathbb{Z}_4[g]/(g^2 - 2) \quad \text{and} \quad R_2 = \mathbb{Z}_4[g]/(g^2 - 2g - 2)$$

  generate each other. They have the same elements $u + vg$ with $u, v \in \mathbb{Z}_4$, the same addition and the same 0 and 1, and differ only in the product $g \cdot g$, which is 2 in R_1 and $2 + 2g$ in R_2. In each of the two rings, the product of the other ring is

$$x \otimes y = xy + (x^2 - x^4)(y + y^4),$$

  which we verified by evaluating both sides on all 256 pairs (x, y) in each ring (`comm_rings16_pairs.py`). The rings are not isomorphic: R_1 contains a square root of 2, whereas R_2 is the ring $\mathbb{Z}_4[i]$ of Gaussian integers modulo 4 (with $i = g - 1$, so that $i^2 = -1$), in which $(u+vi)^2 = u^2 - v^2 + 2uvi$ and $u^2 - v^2$ is never 2 modulo 4. We found this pair by enumerating all 37 commutative rings with identity with 16 elements (`comm_rings16.py`) and comparing the quantities listed above; these distinguish all of the rings except this pair and the pair $\mathbb{Z}_4[g]/(g^2)$, $\mathbb{Z}_4[g]/(g^2 - 2g)$, for which we do not know whether they generate each other.

- **Groups** ($24 \leq n_{\text{diff}} \leq 260$): Two non-isomorphic groups with 260 elements that generate each other are known: the smallest member of a family constructed by Kearnes and Szendrei [27, Example 2.14]. Let $G_1 = \mathbb{Z}_4 \ltimes (\mathbb{Z}_5 \times \mathbb{Z}_{13})$, where the generator of $\mathbb{Z}_4$ acts on $\mathbb{Z}_5$ as multiplication by 2 and on $\mathbb{Z}_{13}$ as multiplication by 5, and let G_2 be the same group with 5 replaced by its inverse 8. Then $x \circ y = xy\,[y,x]^{40}$, with $[y,x] = y^{-1}x^{-1}yx$, is a group operation on the set G_1 that is isomorphic to G_2 and not to G_1, and the product of G_1 is obtained back from $\circ$ by the same formula; we verified these statements by computer (`termeq_counterexample.py`). This is equivalence of the generated functions; the earlier result of Zádori [41] concerns the weaker categorical equivalence of finite groups. For a lower bound, suppose that the groups G and G' on the same set generate each other. The powers of an element are the same in both, so each element has the same order in G and in G'; and an Abelian group generates no group other than itself, so G' is non-Abelian if G is. For $n \leq 23$ with $n \neq 16$, any two non-isomorphic non-Abelian groups with n elements differ in the number of elements of some order, and for $n = 16$ we verified by computing, for each of the 14 groups, all group operations that it generates, that each of them is isomorphic to the original group (`termeq_groups.py`). Hence $n_{\text{diff}} \geq 24$.