A Formal Abstract of the Classification of Finite Simple Groups

Floris van Doorn

University of Pittsburgh

June 13, 2019

j.w.w. Tom Hales, Jesse Han and Kody Vajjha

Outline

- Formal Abstracts
- Classification of Finite Simple Groups
- Implementation

(Potential) Automation in Mathematics

- Proving
- Proof checking
- Conjecture generation
- (Counter)example generation
- Error detection
- Computing
- Generalizing
- Searching
- Transforming the literature
- Machine translation
- Teaching
- Collaboration
- Exploration

Motivation

For many of these activities, it is important or even essential to have a machine-readable semantic representation of mathematical objects.

The goal of Formal Abstracts is to create a large database of mathematics containing theorems in both human readable and machine-readable form.

The database does not contain formal proofs.

Applications

- Searching: We can make theorems more easily searchable by human and machine.
- Translation: It can be used to translate mathematics between natural languages.
- Analysis: Our database can be used for machine-learning projects.
- Exploration: We could make a tool such as a Google Earth for mathematics, providing an intuitive visual map of the entire world of mathematics.
- We make formal methods more relevant for mathematicians.

$$\vdots$$

Classification of Finite Simple Groups

As an initial challenge, we write a formal statement of the classification of finite simple groups in the Lean proof assistant.

Definition

A group is simple if it has no nontrivial normal subgroup.

Theorem

Every finite simple group is in one of the following four families:

- *The cyclic groups $\mathbb{Z}_p$ of prime order;*
- *The alternating group A_n of degree $n \geq 5$;*
- *A group of Lie type (including the Tits group);*
- *One of the 26 sporadic groups.*

History of the Classification

- 1892: Hölder asks for a classification of finite simple groups
- 1963: Feit and Thompson prove the Odd Order Theorem
- 1976: The last sporadic groups (O'N and J4) are discovered
- 2004: The completion of the classification was announced
- 2008: Harada and Solomon fill a minor gap in the proof

Groups of Lie type

To define groups of Lie type, we need quite some algebraic geometry. We need the following concepts:

- Affine varieties
- Affine algebraic groups
- The general linear affine group
- Dynkin diagrams
- Frobenius maps

$$\vdots$$

Using these definitions we can specify a Lie theater

$$\Theta = (p, K, G, B, T, A, D, F, n, \phi, k, j, e, \rho).$$

We obtain a finite simple group of Lie type as a quotient of the fixed points of the Frobenius map F.

Sporadic Groups

The 26 sporadic groups are classified into four groups.

- 5 are in the First Happy Family, called Mathieu Groups.
  - They are defined as automorphism groups or stabilizers of Steiner Systems.
  - A Steiner system $S(t, k, v)$, where $t < k < v$ are positive integers is a finite set X of cardinality v, a collection of k element subsets of X (called blocks), such that each t element subset of X is contained in a unique block.

Sporadic Groups

The 26 sporadic groups are classified into four groups.

- 5 are in the First Happy Family, called Mathieu Groups.
- 7 are in the Second Happy Family
  - They are constructed from the Leech Lattice.
  - The Leech Lattice L is the unique even unimodular lattice in dimension 24 that does not have any vectors of norm 2.

Sporadic Groups

The 26 sporadic groups are classified into four groups.

- 5 are in the First Happy Family, called Mathieu Groups.
- 7 are in the Second Happy Family
- 8 are in the Third Happy Family
  - They are constructed as subquotients of the Monster Group.
  - We defined the monster group explicitly as the quotient of a Coxeter group (Y_{443}).

Sporadic Groups

The 26 sporadic groups are classified into four groups.

- 5 are in the First Happy Family, called Mathieu Groups.
- 7 are in the Second Happy Family
- 8 are in the Third Happy Family
- The 6 remaining groups are called Pariahs.
  - They are the sporadic groups that are not subquotients of the Monster Group.
  - We define most of them explicitly using a presentation.

Statement

We build the library on top of the Lean mathematical library `mathlib`.[1]

We allow the use of the following axiom, which allows us to omit proofs of theorems.

```
axiom omitted {P : Prop} : P
```

The formalized statement of the CFSG is:

```
theorem classification_of_finite_simple_groups
  {G : Group} (h₁ : is_finite G) (h₂ : simple_group G) :
  is_cyclic_of_prime_order G ∨
  is_simple_alternating_group G ∨
  is_of_lie_type G ∨
  is_sporadic_group G :=
omitted
```

[1] `https://github.com/leanprover-community/mathlib`

Specifications

We allow ourselves to construct data that is unique (up to isomorphism) by giving a specification.

```
def leech_lattice_spec : ∃ Λ : euclidean_lattice 24,
  even Λ ∧ unimodular Λ ∧
  ∀ x ∈ Λ.1, x ≠ 0 → ⟪x,x⟫ ≥ 4 :=
omitted

noncomputable def Λ_24 :=
classical.some leech_lattice_spec
```

`classical.some` picks an element that satisfies the specification.

We should take care when writing `classical.some` that we don't accidentally underspecify (or overspecify) a notion.

Docstrings

We generously use docstrings to explain definitions and theorems in human-readable language.

```
/-- The conjugation map `H₁ × H₂ ⟶ G` is given by
  `(h₁,h₂) ↦ h₁ * h₂ * h₁⁻¹`-/
def conjugation (H₁ H₂ : set G.obj.type)
  [is_closed_subgroup H₁] [is_closed_subgroup H₂] :
  (sub H₁).obj × (sub H₂).obj ⟶ G.obj :=
((G.incl H₁).map ≫ diag) ×.map (G.incl H₂).map ≫
  product_assoc.hom ≫
  𝟙 G.obj ×.map (product_comm.hom ≫ G.mul) ≫ G.mul
```

This is not a great way for humans to learn material.

Writing Definitions

When doing formal proofs, much time is spent on getting the definitions exactly "right."

One definition can be much easier to work with than another definition, even if the definitions are equivalent.

- There is a (small) difference between writing $a = b - c$ or $a + c = b$.
- Is $\mathbb{Z}$ a quotient of $\mathbb{N} \times \mathbb{N}$, a quotient of $\mathbb{N} + \mathbb{N}$, equal to $\mathbb{N} + \mathbb{N}$ or something else?
- A vector of n elements can be defined as `fin n → A` or `{ l : list A // l.length = n }` or as a primitive inductive family.

In mathematics, these differences are completely ignored.

When doing formal abstracts, we can ignore these definitions, just as in mathematics.

Consistency of Definitions

Similarly, a high level of consistency of definitions is required when doing formal proofs.
For example, the following are ways to state that a set is non-empty in Lean:

- `s ≠ ∅`
- `∃ x, x ∈ s`
- `nonempty {x // x ∈ s}`

It is convenient when doing formal proofs if there is a single consistent choice across the library.
When doing formal abstracts, using equivalent definitions interchangeably will be less of a problem.

Observations 1

Building a library in Formal Abstracts takes much less time than formalizing.

We omit proofs, and we can also omit a lot of trivial lemmas.

Observations 2

Hopefully Formal Abstracts can be used as a basis for formal libraries.

- Definitions would need to be refined
- Statements need to be rewritten
  - Formal Abstracts contains quite some existential statements of objects you want to construct explicitly in a formal library.

Observations 3

Type class inference is used in Lean to synthesize some implicit information.

```
lemma mul_comm : ∀ {α} [comm_semigroup α] (a b : α),
  a * b = b * a
```

Type class is in some cases unreliable/annoying. This might become a problem with even bigger when scaling the libraries more.

Conclusions

- In some ways formal abstracts is closer to writing in mathematical papers than formalization.
- Building large libraries of research mathematics seems feasible.
- We need to think about the best method to make it as useful for mathematicians as possible.

Thank you!