Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Matematica

Collaboration and Innovation in the Equational Theories Project: Formalizing Mathematics with Lean 4

Tesi di Laurea Magistrale in Formalizzazione di Matematica

Relatore:
Chiar.mo Prof.
GIOVANNI PAOLINI

Correlatore:
Dott.
LORENZO LUCCIOLI

Presentata da:
MARCO PETRACCI

Sessione VI
Anno Accademico 2023-2024

A Davide,
a mamma e papà.

Abstract

This thesis explores the "Equational Theories" project, an innovative initiative launched by Terence Tao which combines collaboration between professional and amateur mathematicians with the use of artificial intelligence tools and proof-assistant languages, such as Lean 4. The project focuses on the study of equational theories in universal algebra, particularly on magmas (sets equipped with a binary operation), with the goal of determining the implications between different algebraic equations and constructing an implication graph.

A central aspect of this thesis is the *greedy construction*, an iterative algorithm that enables the construction of infinite magmas satisfying specific equations. This method has been applied in detail to prove the non-implication $1516 \not\vdash 255$, through the construction of a counterexample. The formalization in Lean revealed errors in the informal version of the proof, highlighting the importance of formalization in ensuring the correctness of the results.

This thesis also introduces *magma cohomology*, a new mathematical structure that emerged during the project, which generalizes group cohomology and has proven useful in resolving finite implications.

In summary, the project represents an innovative approach to mathematical research, based on open collaboration, advanced computational tools, and rigorous formalization, paving the way for future explorations in this field.

Contents

Introduction

In recent years, the advent of new interactive theorem provers has drawn increasing attention to the formalization and verification of mathematical proofs. These tools not only ensure the correctness of the results obtained but also support mathematicians in achieving their goals. This thesis focuses on one of the most widely used tools, Lean 4, and how it is employed by the mathematical community.

In particular, the thesis presents a description of the project titled **Equational Theories**, launched by Terence Tao. The goal of this project is to experiment with a new way of collaborating, combining the efforts of professional and amateur mathematicians with those of artificial intelligence tools and proof assistant languages, such as Lean. Typically, research projects are conducted by small groups of mathematicians who are familiar with all aspects of the subject. In this case, Terence Tao aimed to experiment with a large-scale project, given the minimal prerequisites required, leveraging proof assistant languages like Lean, which allow for broader participation and the use of AI tools.

The Equational Theories project focuses on the study of equational theories in universal algebra, particularly on magma structures, which are sets equipped with a binary operation. The main objective is to determine the implications between different algebraic equations, constructing an implication graph that describes the relationships between these equations. This graph is essential for understanding how different algebraic properties are interconnected and for identifying which equations imply or do not imply others.

In this thesis, we will focus particularly on the formalization of the non-implication $1516 \not\vdash 255$, a significant result achieved through the construction of a counterexample using the greedy algorithm. This work was undertaken by Lorenzo Luccioli, Pietro Monticone, and myself, leveraging pre-existing code developed by Bernhard Reinke for a different implication, and it highlighted the importance of formalization in Lean for identifying and correcting errors in the "on paper" version of the proof.

In summary, this thesis explores how mathematical formalization, combined with innovative collaboration and advanced computational tools, can lead to new results and discoveries. The Equational Theories project represents a significant step toward a new mode of mathematical research, where formal correctness and large-scale collaboration play a central role.

Chapter 1

Proof assistants

Proof assistants, or interactive theorem provers, are computer systems that allow a user to do mathematics on a computer: not so much the computing (numerical or symbolical) aspect of mathematics, but the aspects of proving and defining. So a user can set up a mathematical theory, define properties and do logical reasoning with them.
They are now mainly used by specialists who formalize mathematical theories in it and prove theorems. If you are already convinced of your proof, it is not convenient to use proof assistants to verify it: the formalization needs much more details than the "on paper" version. However, it often happens in mathematics that one is not fully convinced of a proof, or it is not widely accepted by the community: it is in these cases that proof assistants demonstrate their usefulness.

1.1 Lean 4

Lean 4 is a new open source interactive theorem prover, and a functional programming language, i.e. a software that allows you to formalize mathematical definitions, statements and demonstrations by verifying their correctness and, in some cases, helping with their construction.

The Lean project was launched by Leonardo de Moura when he was at

Microsoft Research in 2013. Lean 4 is the latest version, released in 2021, and it is hosted on GitHub[1].

It is not easy for newcomers to understand Lean code. For this reason, in the next section we provide the basic knowledge of Lean, inserting examples and walking through the comprehension of the code step-by-step.

1.2 Overview of Lean Code

First of all, we remember that Lean is a programming language, so a code editor needs to be downloaded: the most used is **Visual Studio Code**. Lean is a tool for building complex expressions in a formal language known as **dependent type theory**. Every expression has a type, and you can use the `#check` command to print it. Some expressions have types like $\mathbb{N}$ or $\mathbb{N} \to \mathbb{N}$: these are mathematical objects.

```
#check 3 + 2

def g (x : N) := x + 2

#check g
```

where the results of these `#check` are:

```
3 + 2 : ℕ

g : ℕ → ℕ
```

In general, types can be thought of as if they were sets, but with some important differences: the most significant one is that, while in set theory an element can belong to multiple sets, in type theory an element can belong to only one type. One particularity is that even propositions are types. For example, mathematical statements have type `Prop`.

```
theorem add_comm (a b : N) := a + b = b + a

#check add_comm
```

[1] see `https://github.com/leanprover`

In this case, `#check`, in addition to obviously showing the statement, provides some technical information about the inputs required by the theorem, which we will examine in more detail in the next example:

```
1 add_comm.{u_1} {G : Type u_1} [AddCommMagma G] (a b : G) : a + b = b
    + a
```

Let us now examine in detail an example of a Lean definition, analyzing all the commands used.

```
1 def Function.Injective {A : Sort*} {B : Sort*}
2   (f : A → B) : Prop :=
3 ∀ (a b : A), f a = f b → a = b
```

In this way we have just given the definition of injective function. We now enter the code to see step by step which commands are used:

- `def` is the first command met. It is a keyword that allows us to define a new object in Lean. There are other keywords that can be used for the same goal, for example `structure` can be used for definitions that group multiple variables or properties within it;

- `Function.Injective` is the name of the new object that we are defining. There are no rigid rules to follow when choosing a name, you can use whatever you prefer: however, it is advisable to choose something related to what we are defining.

The next three items are the arguments to define the object. Each time you want to say that a function is injective, Lean will require as input that information, namely the domain and the codomain of the function, and the function itself. The arguments differ in the type of brackets that enclose them, and now we are going to compare them.

- `{A : Sort*} {B : Sort*}` indicates the type to which the two sets A and B belong, which will then be taken as the domain and codomain of the function. Arguments in curly brackets `{ }` are called *implicit arguments*, because Lean does not explicitly require them when we use the definition,

especially when they can be easily deduced from the context. Implicit arguments can also be enclosed within square brackets `[ ]`, but they work differently: for example if we had had `[add G]`, Lean would have tried to independently deduce that G has an addition operation, based on previously proved instances of the code;

- the last argument is `(f : A → B)`, the function that is required to be injective. The ones contained between the round brackets `( )` are called *explicit arguments*, because we must recall them as input every time we use this definition;

- the part of the code between `:` and `:=` represents the type of the object we are defining: in this case it is the proposition that the function f is injective, so its type will be `Prop`;

- at the end we have the body of the definition, that is for all a, b elements of the set A, if $f(a)$ is equal to $f(b)$, then a is equal to b.

Let's look at a new example. This time we see how to prove a rather simple theorem. The difficulty of programming is that often simple proofs on paper require more attention than expected for formalization in lean.

```
1 theorem prove1 (a b c : ℕ) : a * b * c = b * (a * c) := by
2   rw [mul_comm a b]
3   rw [mul_assoc]
```

Again, we dive into the code to understand it better.

- `theorem` is the keyword to start the definition of a new theorem in Lean. We can also use `lemma`, or `example` if we don't need to give it a name to call it later in the code;

- `(a b c : ℕ)` are the arguments of the theorem, so every time we want to apply it we have to check that these hypotheses are verified. In this case we can apply it only if it has as input three elements of $\mathbb{N}$;

- after `:` there is the thesis, in the form of a proposition. In this case the theorem says that $a \cdot b \cdot c = b \cdot (a \cdot c)$ holds for every natural number.

Finally, after `:= by`, the proof of the theorem begins. Each step of the proof simplifies the thesis to be proved, until it becomes trivial to prove. When a cursor is in the middle of a tactic proof, Lean reports on the current proof state in the *Lean Infoview* window, along with the available hypotheses. As you move your cursor past each step, you can see the state change. Let us look how it works in this case:

- when the cursor is next to `by`, the *Lean Infoview* shows the initial state. There is only one hypotesis, that is a, b, c are natural numbers. The equality after $\vdash$ represents the thesis:

```
1 goal
a b c : ℕ
⊢ a * b * c = b * (a * c)
```

- the first step is `rw [mul_comm a b]`, where `mul_comm a b` claims the commutativity of the multiplication, specifically $a * b = b * a$, and `rw[ ]` is the rewrite tactic, that replaces the left-hand side of an identity (in this case it considers the identity given by `mul_comm a b`) by the right-hand side in the goal. So the goal changes, as the *Lean Infoview* shows:

```
1 goal
a b c : ℕ
⊢ b * a * c = b * (a * c)
```

- at the end we have `rw [mul_assoc]`. It works like the previous step, but in this case the equality that the rewrite tactic uses to make the substitution is $(b*a)*c = b*(a*c)$ given by `mul_assoc`, i.e. the associative property of multiplication. At this point the goal is transformed into `⊢ b * (a * c) = b * (a * c)`, which is trivial, so the proof is complete. Even the *Lean Infoview* communicates that there are no more goals:

```
No goals
```

The proof could also be done in a more concise way, using the command `rw [mul_comm a b,mul_assoc]`.

As can be seen from the example just given, Lean uses a type of backward reasoning, that is, it starts from the goal that you want to demonstrate and simplifies it until it becomes trivial. It is also possible to use forward reasoning, working on the hypotheses that you have in order to be able to then arrive at the thesis, but it is less common. The ideal method would be to use the two types of reasoning in a combined manner, simplifying the goal as much as possible and developing the hypotheses in order to be able to arrive at a trivial resolution.

Lean also helps in building proofs. It warns us if the proof we are building is not yet finished, that is, if the various steps are not yet sufficient to reach the goal. In this case, the keyword `sorry` is very useful, as it offers a partial resolution of the thesis, allowing us to delay the problem and use that result elsewhere in the code.

Moreover, Lean can help us complete the code: by using the commands `rw?` and `apply?`, suggestions for the next steps in the proof appear in the *Lean Infoview*, which also shows how the goal would change accordingly.

1.3 Mathlib

The Lean mathematical library, *Mathlib*, is a community-driven project born to build a unified library of mathematics formalized in the Lean proof assistant. Mathlib is essential to work efficiently with Lean 4, as it provides a huge amount of mathematical foundations ready to be used, so that you do not have to continually prove concepts and theorems from scratch. It contains a large amount of mathematical definitions and theorems in various fields, such as algebra, number theory, geometry, analysis, logic, category theory, and others[2]. This means that users can reuse already formalized results,

[2]see in more detail here `https://leanprover-community.github.io/mathlib-overview.html`

saving time in formalization and focusing on more advanced problems.

One of the main goals of Mathlib is to provide statements in the most general form possible, eliminating all unnecessary assumptions, so that they can be used in multiple contexts.

Being an open-source project, whose development is managed by the community and organized by maintainers, individual contributions are highly appreciated: however, the proposed code must undergo a review process conducted by a maintainer before being merged into the main branch, and a high standard of quality is required. This approach allows the library to grow rapidly, while keeping a high level of coherence and maintainability.

1.4 Importance of proof assistants

Proof assistants are becoming increasingly crucial for mathematicians, especially in contexts that require formal rigor, absolute precision, and the handling of complex theorems. The main function of proof assistants is to ensure that proofs are correct.

Often, you have to work with complex theorems that involve thousands of logical steps or advanced concepts. Using these new tools allows you to manage and formalize theorems that are difficult to do by hand, such as Fermat's Theorem or other proofs of topology or algebraic geometry, while also ensuring a rigorous formal approach.

It is important to note that proof assistants stimulate greater collaboration among mathematicians, as proofs can be written and verified in a way that is easily understandable and shareable. In particular, the open-source community around tools such as Coq, Lean, and Isabelle has created an ecosystem that favors teamwork and the collective construction of results. Complex theorems can be formalized by a group of researchers in different universities or even in different continents, or large-scale projects can be launched, just as happened with the **Equational theories project** launched by Terence Tao, which will be presented in the next chapter.

Chapter 2

Equational theories project

At the end of September 2024, Terence Tao announced a new project[1] called "**Equational theories**" on his blog. In the introduction Terence Tao explained how mathematicians tend to work in small trusted groups, while they rarely collaborate with the public or with AI tools, due to the risk that a single mistake could invalidate the entire work: interactive theorem provers can counteract this danger. Previously there have been several formalization projects that have dealt with existing results, but now he is trying to explore new problems.

The blog post announced a new pilot project with two goals: to study a specific mathematical problem but also to explore new ways to collaborate on mathematics in a public setting using **Lean formalization as the medium**. So the idea is to collaboratively work on the problem with the additional correctness guarantees from formalizing the results.

An example of this style of project is the Busy Beaver challenge[2], a massively collaborative research project dedicated to the Busy Beaver problem, which was recently completed. The goal was to formalize a result about Turing

[1]You can see here: `https://terrytao.wordpress.com/2024/09/25/a-pilot-project-in-universal-algebra-to-explore-new-ways-to-collaborate-and-use-machine-assistance/`

[2]More information on this can be found at this link: `https://bbchallenge.org/13493336`

machines, i.e. to prove the conjecture "$BB(5) = 47,176,870$ ", about the 5th busy beaver value; in this case the formal theorem prover Coq[3] was used.

But Terence Tao's idea was different: instead of working on an individual problem, he thought that a good approach for this type of project is to use crowdsourcing for a class of problems, because this makes it easier for many people to partecipate, especially for people without in-depth knowledge of a specific area. In particular the problem of Universal Algebra he chose is general enough to be accessible to many contributors with different backgrounds, from accomplished mathematicians to people with no graduate level training, or limited experience in Lean.

2.1 Preliminary definitions

Before describing the research goal of the project, we have to set down some definitions. While the algebraic structures most commonly studied are groups, rings, and fields, which exhibit recurring properties such as associativity and commutativity (as can be seen in [Her78]), the reference algebraic structure in this project is the magma, endowed with less conventional properties.

Definition 2.1.1 (Magma)**.** A magma is a set M with a binary operation $\diamond : M \times M \to M$.

Initially there are no specific rules that the operation $\diamond$ must satisfy, and in these cases, magmas are not very interesting objects. So we start adding some *equational axioms* on these sets, which are equalities involving the operation $\diamond$ and indeterminate variables in M.

Definition 2.1.2 (Equational Law)**.** An *equational law* is a formal symbolic expression A of the form

$$\varphi(\diamond; x_1, \ldots, x_k) = \psi(\diamond; x_1, \ldots, x_k) \tag{A}$$

[3]Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs

where $\varphi(\diamond; x_1, \ldots, x_k)$ and $\psi(\diamond; x_1, \ldots, x_k)$ are applications of the operation $\diamond$ to the variables $x_1, \ldots, x_k$.

We say that A has *order* n if the operation $\diamond$ appears n times, i.e. if the sum of the orders of φ and ψ is n.

Given a magma $\mathcal{M} = (M, \diamond)$, we say that $\mathcal{M}$ obeys A if the equality

$$\varphi(\diamond; a_1, \ldots, a_k) = \psi(\diamond; a_1, \ldots, a_k)$$

holds for all $a_1, \ldots, a_k \in M$. In this case we write $\mathcal{M} \models A$.

Two easy examples could be the *commutative law*:

$$x \diamond y = y \diamond x,$$

and the *associative law*:

$$(x \diamond y) \diamond z = x \diamond (y \diamond z),$$

that must be satisfied $\forall\ x, y, z \in M$. Axioms that include constants, such as $a \diamond x = x$ (where $a \in M$), are not considered *equational axioms*: equational axioms must contain only indeterminate variables.

We say that an Equation A *implies* an Equation B (denoted $A \vdash B$) if all magmas that obey A also obey B, otherwise we say that Equation A *refutes* Equation B. This definition will be crucial in this work, because the project analyzes magmas that initially obey a single equation A, and aims to study which equations are implied and which are not implied by it, as well as which ones imply A and which do not. We now provide two examples.

Given Equation $C : x \diamond y = x$ and Equation $D : x \diamond (y \diamond z) = (x \diamond w) \diamond u$, we want to prove the following proposition:

Proposition 2.1.3. $C \vdash D$.

Proof. We want to prove that:

$$x \diamond (y \diamond z) = (x \diamond w) \diamond u \tag{2.1}$$

From equation C we obtain that:

$$y = y \diamond z, \tag{2.2}$$

and

$$x = x \diamond w. \tag{2.3}$$

So by substituting Eq. (2.2) and Eq. (2.3) in Eq. (2.1) we get that:

$$x \diamond (y \diamond z) = x \diamond w. \tag{2.4}$$

We apply the equation once again C:

$$x \diamond w = (x \diamond w) \diamond u. \tag{2.5}$$

Substituting Eq. (2.5) into Eq. (2.4) we obtain the thesis. □

Now we want to provide an example of a non-implication between two equations: these cases are more complicated to prove, as it is necessary to find counterexamples. Consider the equations $E : x = x \diamond x$ and $E' : x = x \diamond y$, let us prove that:

Proposition 2.1.4. $E \nvdash E.'$

Proof. Let us consider a magma M with elements only in {0,1}, and with an operation $\diamond$ defined according to the Table 2.1. As we can easily see, the

$\diamond$	**0**	**1**
0	0	1
1	1	1

Table 2.1: Operator $\diamond$ on M

model satisfies E:

- $0 = 0 \diamond 0$;
- $1 = 1 \diamond 1$.

But does not obeys E' :

- $0 \neq 0 \diamond 1$.

Therefore, we have proven that equation E does not imply E'. □

2.2 Goal of the project

Since there are an infinite number of equational laws, it is useful to consider only a finite number of them, for example by putting a bound on the order of the laws. For this reason, let us focus on the set $\mathcal{E}$ containing all the *equational axioms* of order at most four, up to relabeling and the reflexive and symmetric axioms of equality. Before proceeding, let us explain what these three criteria consist of, which allow us to avoid counting the same property multiple times:

- **Relabeling**: we consider two laws, A and A', to be the same if, assuming that A is of the form $\varphi(\diamond; x_1, \ldots, x_k) = \psi(\diamond; x_1, \ldots, x_k)$, there exist some permutation σ of the alphabet of formal symbols such that A' is equal to $\varphi(\diamond; \sigma(x_1), \ldots, \sigma(x_k)) = \psi(\diamond; \sigma(x_1), \ldots, \sigma(x_k))$. For example, the equations $x \diamond y = x$ and $y \diamond x = y$ are the same up to relabeling;

- **Symmetry**: we consider two laws to be the same up to symmetry if, given A of the form $\varphi(\diamond; x_1, \ldots, x_k) = \psi(\diamond; x_1, \ldots, x_k)$, A' is $\psi(\diamond; x_1, \ldots, x_k) = \varphi(\diamond; x_1, \ldots, x_k)$, that is if their LHS and RHS are swapped;

- **Reflexivity**: we say that an equation is reflexive if the LHS and RHS coincide. In this case we consider it to be the same as the trivial law of order 0, $x = x$, which is the only trivial law that we count.

The **goal of the project** is to determine the implication graph of the set $\mathcal{E}$, proving which equations imply, or do not imply, which others. We now try to quantify how many implications and refutations we need to study.

Given the criteria just explained, let us now describe, in general, how to count all equations of a given order n.

Lemma 2.2.1. *The number of equational laws of order n, up to relabeling, is*

$$C_{n+1}B_{n+2},$$

where C_n is the $n-th$ Catalan number[4], and B_n is the $n-th$ Bell Number[5].

Proof. Let us consider an equation A of order n of the form $\varphi(\diamond; x_1, \ldots, x_k) = \psi(\diamond; x_1, \ldots, x_k)$, where φ has order n' and ψ has order n'', such that $n'+n'' = n$. The different shapes that φ can take are just the different ways we have of associating the n' applications of the binary operator $\diamond$ which are counted by the Catalan number $C_{n'}$, and the same for ψ with $C_{n''}$. Therefore, the total number of ways to associate n times the operator $\diamond$ in A is counted by:

$$\sum_{n'=0}^{n} C_{n'} C_{n-n'} = C_{n+1}.$$

Now, for each of those shapes we need to label the $n+2$ variables that appear in the expression. If we want to avoid counting multiple times expressions that are equivalent up to relabeling, we can notice that assigning the variables is equivalent to partitioning the set of $n+2$ positions into nonempty subsets, and then assign the same symbol to the positions in each subset, and different symbols to different subsets. The number of ways to partition a set of $n+2$ elements is given by B_{n+2}.

In conclusion, the formula $C_{n+1}B_{n+2}$ counts the number of equations of order n, up to relabeling. □

This is not yet the final number of equations, because we must avoid counting symmetric pairs of equations twice. To reach the exact number, we must first provide a result that will be useful later in the proof.

Lemma 2.2.2 (Burnside's Lemma)**.** *Let G be a finite group acting on a finite set X and let $|X/G|$ be the number of orbits of the action of G on X. Then we have*

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|,$$

[4]The Catalan numbers form a sequence of natural numbers that is very useful in combinatorics. They can be defined recursively by imposing $C_0 = 1$ and $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$ for $n \geq 1$. In this context, we use them to count the number of ways to associate n applications of a binary operation.

[5]The Bell numbers B_n are defined as the number of partitions of a set of n elements.

where X^g is the set of elements of X fixed by g, for every $g \in G$.

Proof. For the proof of this theorem, the reader is referred to [Rot95], Theorem 3.22. □

Lemma 2.2.3. *The number of equational laws of order n up to symmetry and relabeling is*

$$\begin{cases} \frac{1}{2}C_{n+1}B_{n+2} & \text{if } n \text{ is odd,} \\ \frac{1}{2}(C_{n+1}B_{n+2} + C_{n/2}(2D_{n+2} - B_{n+2})) & \text{if } n \text{ is even,} \end{cases}$$

where D_n is the number of partitions[6] of $[n] := \{1, 2 \ldots, n\}$ up to reflection.

Proof. Let $A : \varphi = \psi$ and $A' : \psi = \varphi$ be two equations of order n. In Lemma 2.2.1, they can be counted only once if, in addition to being the symmetric of one another, they are also identical up to relabeling, or counted twice if they are not.

In the case where n is odd, A and A' can never be identical up to relabeling, because φ and ψ will certainly have different orders. Therefore, in the formula $C_{n+1}B_{n+2}$, each equation is counted twice due to its symmetry. To obtain the number of equations up to symmetry and relabeling, it will be enough to divide the result by 2.

We now proceed to examine the case where n is even. We can use a similar strategy of dividing by 2 the number of laws up to relabeling, but before doing that, we need to artificially add back A' whenever it has been removed from the previous count by being the same as A up to relabeling. So, if we call X the number of laws $A : \varphi = \psi$ that are the same as $A' : \psi = \varphi$ up to relabeling, the result we are looking for is

$$\frac{1}{2}(C_{n+1}B_{n+2} + X). \tag{2.6}$$

To calculate X, we must determine in how many ways we can construct an equation $A : \varphi = \psi$, such that it is identical to its symmetric counterpart

[6] D_n counts the number of ways to partition a set of n elements up to the reflection map $i \mapsto n + 1 - i$. For example, given $n = 3$, the partitions $\{\{1\}, \{2, 3\}\}$ and $\{\{3\}, \{2, 1\}\}$ are considered the same.

$A' : \psi = \varphi$ up to relabeling. First, φ and ψ must have the same order, that is, $n/2$, and the same shape, which we can choose in $C_{n/2}$ ways.

Then, we need to assign the variables. To do this, let us call $m := (n+2)/2$, L the set of partitions of $[2m]$, $\tau\colon [2m] \to [2m]$ the permutation that swaps the first half and the second half of the elements. With a slight abuse of notation, we can consider $\tau\colon L \to L$ as the map induced by τ, so that for every partition $\mathcal{P} \in L$, for every $P \in \mathcal{P}$, there is $\tau(P) \in \tau(\mathcal{P})$ such that for every $i \in [2m]$ we have $i \in P$ if and only if $\tau(i) \in \tau(P)$. Since τ is an involution over L, then $\{\mathrm{id}, \tau\}$ is a group acting on L, and we can apply Burnside's Lemma to obtain the following:

$$|L \setminus \{\mathrm{id}, \tau\}| = \tfrac{1}{2}(|L^{\mathrm{id}}| + |L^{\tau}|). \tag{2.7}$$

Clearly, all partitions are fixed by the identity, so $|L^{\mathrm{id}}| = |L| = B_{n+2}$. Moreover, the number of orbits $|L \setminus \{\mathrm{id}, \tau\}|$ is equal to the number of ways to partition $[2m]$ up to the permutation τ, which is D_{n+2}. Finally, since the numbers in $[2m]$ correspond to the positions of variables in φ and ψ, then the action of τ corresponds to swapping φ and ψ in A, so the partitions fixed by τ are exactly the ones such that A' is the same as A up to relabeling, so exactly the ones we are looking for. Therefore, Eq. (2.7) becomes

$$D_{n+2} = \tfrac{1}{2}(B_{n+2} + |L^{\tau}|),$$

and allows us to conclude that $X = C_{n/2}|L^{\tau}| = C_{n/2}(2D_{n+2} - B_{n+2})$. We obtain the final result by substituting the value of X just made into Eq. (2.6). □

At this point, we only need to apply the final criterion: reflexivity.

Theorem 2.2.4. *The number of equational laws of order n up to relabeling, symmetry, and reflexivity is*

$$\begin{cases} 2 & \text{if } n = 0, \\ \frac{1}{2}C_{n+1}B_{n+2} & \text{if } n \text{ is odd}, \\ \frac{1}{2}(C_{n+1}B_{n+2} + C_{n/2}(2D_{n+2} - B_{n+2})) - C_{n/2}B_{n/2+1} & \text{if } n > 0 \text{ is even}. \end{cases}$$

Proof. To prove this result, we can use Lemma 2.2.3 and subtract the number of trivial laws of order n.

A trivial law is of the form $A : \varphi = \varphi$, therefore, if n is odd, equations of this type cannot exist. Therefore, the result follows directly from Lemma 2.2.3.

Let us now consider the case where n is even. First, we study the case $n = 0$. It is easy to count the possible number of equations of order 0, as there are only two possibilities: the trivial law $x = x$ or the *singleton law* $x = y$.

For $n > 0$, we can count the number of trivial laws $E : \varphi = \varphi$ by first choosing the shape of φ, which we can do in $C_{n/2}$ ways, and then assigning its $n/2 + 1$ variables, which we can do in $B_{n/2+1}$ ways. □

The set $\mathcal{E}$ that we will study contains only equations up to order 4, so there will be exactly

$$2 + 5 + 39 + 364 + 4284 = 4694,$$

which means that it will be necessary to study a total of approximately 22 million implications or refutations. The list of all equations has been generated and numbered using a Python script[7].

2.3 How to minimize the number of proofs

Given the large number of proofs to be addressed, it was necessary to adopt techniques that would allow for a reduction in their number, in order to deduce the implication graph by studying the smallest possible number of implications or refutations. For example, starting from a base of already proven results, let us see how it is possible to expand the graph by exploiting the properties of transitivity and duality.

[7]The script to generate them can be found here: `https://github.com/teorth/equational_theories/blob/main/scripts/generate_eqs_list.py`. Instead here `https://github.com/teorth/equational_theories/blob/main/data/equations.txt` a list of all equations.

As for transitivity, it allows us to automatically prove that:

- $A \to B$ and $B \to C$ implies $A \to C$;
- $A \to B$ and $A \nrightarrow C$ implies $B \nrightarrow C$;
- $B \to C$ and $A \nrightarrow C$ implies $A \nrightarrow B$.

On the other hand the duality in a magma $(M, \diamond)$, where $\diamond$ is a binary operation on M, consists of defining another operation, called the **dual operation**, which can be obtained through a symmetric transformation of the structure, maintaining the same properties but in an inverted form. We now give the formal definition.

Definition 2.3.1 (Dual magma)**.** Given a magma $(M, \diamond)$, we define a new binary operation $* : M \times M \to M$ as follows:

$$x * y := y \diamond x.$$

Then, $(M, *)$ is the **dual magma** of $(M, \diamond)$.

With this definition, it is easy to deduce what it means for one equation to be the dual of another: given, for example, Equation $4 : x = x \diamond y$, its dual is $x = y \diamond x$, or given the Equation $11 : x = x \diamond (y \diamond y)$, its dual is $x = (y \diamond y) \diamond x$. Therefore, having already proved that $4 \to 11$, we automatically have $4^* \to 11^*$.

Despite these properties, there is still a huge number of proofs to be addressed, which is why proof automation is also employed. Two main types are used in this context:

- **Lean tactics**: these produce proofs in Lean and are tightly integrated with it. Some of the tactics used in the project are `aesop` [LF23], `duper` [Clu+24] and `egg` [RG24];
- **ATP** (Automated Theorem Provers): they are independent of Lean, having existed before it, and come in various types. They are then used in different ways:

  - some ATPs are directly implemented in Lean as tactics;
  - there are Lean tactics that use external ATPs, then translate their output into a Lean proof;
  - the last approach involves using an external script that employs an ATP to produce Lean code.

- An important tool used in collaboration with ATPs is **Mace4**[8]: it is very useful for finding counterexamples to a given set of axioms by searching for finite models that satisfy first-order and equational statements. Mace4 can be a valuable complement to Prover9 (an ATP), searching for counterexamples before (or simultaneously with) Prover9 searches for a proof. It can also be used to help debug input clauses and formulas for Prover9. The two tasks can be run in parallel, with Prover9 looking for a proof and Mace4 searching for a counterexample.

Thanks to these tools, we can greatly reduce the number of proofs, mainly using them for trivial, repetitive, and uninteresting ones that only result in a large waste of time.

2.4 Project Organization

The approach adopted by Terence Tao for this project consists of breaking the central problem into many smaller subproblems, allowing participants to independently choose which proofs to work on based on their expertise. All contributions are managed through a central repository on **GitHub**[9], where the formalized proofs in Lean are uploaded, and a file describing the current state of the project is updated concurrently.

[8] see `https://www.cs.unm.edu/~mccune/prover9/`.

[9] GitHub is a collaborative development platform based on Git. It allows developers to store, manage, and share their code, so that multiple people can easily work together on the same project. The main repository of the project can be found at the following link: `https://github.com/teorth/equational_theories`

An innovative aspect of the project organization is the method introduced by Pietro Monticone, which involves maintainers creating GitHub Issues, which participants can then claim. This system prevents multiple people from unknowingly working on the same task and also facilitates the overall management by keeping track of the work completed.

In this context, the **day-by-day diary**[10], continuously updated by Terence Tao, proves to be particularly useful. He summaries results and **Zulip** discussions[11] as they occurred, offering a historical perspective on the progression of the project. The diary serves as a key tool since it enables each contributor, especially for those who tends to focus solely on their task or do not follow Zulip (which could be very active), to gain an overview and stay informed about the developments.

In parallel with other formalization projects, a semi-formal **blueprint**[12] is also developed. The **blueprint** is a human-readable record of the results established, which accepts handwritten proofs from contributors without Lean experience, as well as contributions from automated proofs, whose outputs are typically in a different format from Lean.

The blueprint is written in a special form of LaTeX that supports some integration with Lean. In particular, definitions, theorems, and proofs of theorems can be tagged with additional macros:

- A macro `\lean{leanThm}` in the statement of a definition or theorem in the blueprint, will allow the blueprint to connect that definition or theorem to the corresponding Lean definition or theorem. It is possible to link a blueprint theorem to multiple Lean theorems;

- The macro `\uses{ref1, ref2}` in the statement of a definition or theorem,

[10]The diary can be viewed at the following link: `https://github.com/teorth/equational_theories/wiki/Terence-Tao%27s-personal-log#day-67-dec-1`.

[11]Zulip is a conversation platform widely used by the Lean community, designed to enable people to collaborate efficiently on complex projects. At this link you can find the Zulip discussions: `https://leanprover.zulipchat.com/#narrow/stream/458659-Equational/`.

[12]`https://teorth.github.io/equational_theories/blueprint/`

or a proof of that theorem, will indicate that the relevant statement or proof uses the results in the blueprint that have the indicated `\label{ref1}` and `\label{ref2}`. These will create edges in the dependency graph[13];

- The macro `\leanok` in the statement of a definition or theorem indicates that the statement has been formalized. The macro `\leanok` in the proof of a theorem indicates that the proof has been formalized. This will create various colors in the nodes of the dependency graph.

Contributors are welcome to make suggestions or additions to the blueprint, but all contributions are subject to continuous integration (CI) checks, which ensure that the blueprint compiles correctly. Additionally, all contributions must be approved by the maintainers, who verify their correctness.

Other useful tools for the project are:

- As I mentioned before, the **Zulip** chat proves to be an invaluable tool for participant discussions, useful for resolving doubts regarding specific implications or for delving deeper into topics that have emerged during the proofs;

- **Equational Explorer**[14] is the primary tool to navigate the implication graph. All the equations of the project can be viewed, and each one has a dedicated page: here there are its inbound and outbound implications and refutations, other members of that equation's equivalent class, its dual, and also links to other useful tools, such as Graphiti or Finite Magma Explorer;

- **Graphiti**[15] allows you to view the implication graph as a diagram, where the edges going upwards are implications and equivalence classes are collapsed into a single node, which is distinguished by its rounded

[13] `https://teorth.github.io/equational_theories/blueprint/dep_graph_document.html`

[14] `https://teorth.github.io/equational_theories/implications/`

[15] `https://teorth.github.io/equational_theories/graphiti/`

edges. It also gives the option to view only the branches related to specific equations instead of displaying the entire graph, which can be really difficult to navigate. For example, below we can see the graph, generated using Graphiti, of the equations that imply Equation 43 "Commutative Law".

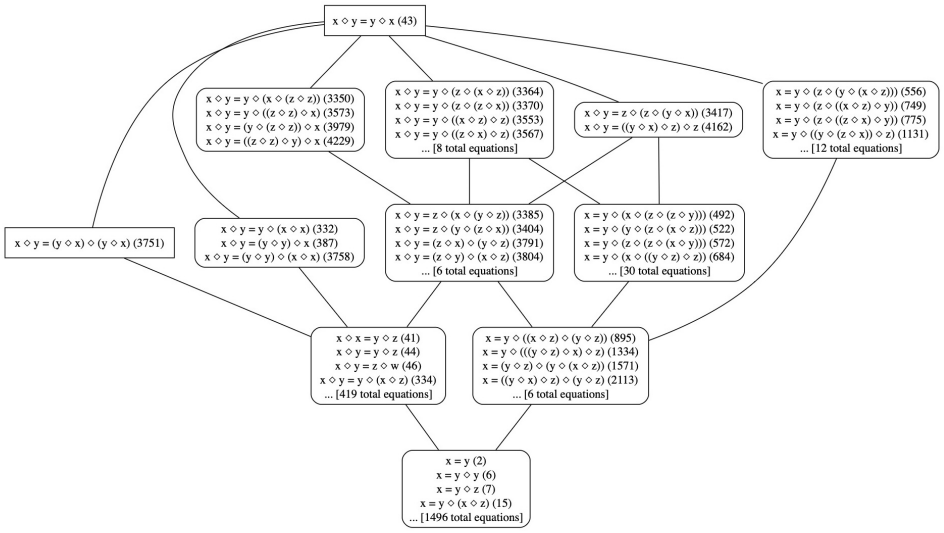

Figure 2.1: Graph representing the equations that imply the "Commutative Law"

Chapter 3

Useful Techniques and Mathematical Structures

In this chapter, we focus on the **Greedy Algorithm Construction**, a crucial technique widely used in various proofs within the project, including the refutation of implication $1516 \vdash 255$, which will be explored in greater detail in the following chapters. This algorithm plays a fundamental role in constructing infinite magmas that satisfy specific equations. Additionally, we will introduce a mathematical structure that has emerged as a result of these proofs: **Magma cohomology**.

In the first part of the chapter, we provide a detailed analysis of the Greedy Algorithm Construction, including an example to illustrate how it is applied in the construction of a magma. The second part of the chapter is dedicated to describing the structure of Magma cohomology and how it is connected to group cohomology.

3.1 Greedy Algorithm Construction

The Greedy Algorithm Construction is a method particularly useful for constructing algebraic structures, such as magmas, that satisfy specific equations. The idea is to work with magma operations that are already partially

defined, and then extend them to obtain the entire operation table. This is an iterative algorithm that consists of making the choice that seems the best at the moment, hence the term "greedy" (see [Cor+22], Section 15), without worrying too much about the long-term consequences. Let us explain in further detail how it functions.

Consider a partially defined magma operation $\diamond : S \to G$, on some carrier set G, with $S \subseteq G \times G$. This can be interpreted as a relation $R(x, y, z)$, with $x, y, z \in G$, that holds if and only if $x \diamond y$ is well-defined, i.e. $x, y \in S$, and equal to z. Furthermore, the relation R satisfies the following "vertical line test":

$$\text{if } R(x, y, z) \text{ and } R(x, y, z') \Rightarrow z = z'. \tag{VLT}$$

With an abuse of notation, since R is associated with the operation $\diamond$, we can say that R is a partially defined magma operation on S: our goal is to extend R so that it is defined on the entire $G \times G$.

Given an expression $w(x_1, \ldots, x_n)$[1] in variables $x_1, \ldots, x_n$, we will say that $w(x_1, \ldots, x_n)$ is *well-defined* with respect to the partial operation R if it can be entirely computed by R. For example, we will say that the expression $x \diamond (y \diamond z)$ is well-defined if there exist $u, v \in G$ such that $R(y, z, u)$ and $R(x, u, v)$ hold, which means $x \diamond (y \diamond z) = u$. Notice from VLT that this evaluation, when it exists, is unique. Instead, $w(x_1, \ldots, x_n)$ is said to be *almost well defined* if all the subexpressions of w are well defined. Returning to the previous example, $x \diamond (y \diamond z)$ is almost well-defined if there exists $u \in G$ such that $R(y, z, u)$ holds, but there does not exist $w \in G$ such that $R(x, u, w)$.

Definition 3.1.1. Given an equational law $w_1(x_1, \ldots, x_n) = w_2(x_1, \ldots, x_n)$, it is said to be *locally obeyed* by R if w_1 and w_2 are well-defined and evaluated to the same output y.

We now provide some examples to clarify this further.

[1] It is a notation to indicate an expression with $n-1$ applications of the binary operator $\diamond$ between the variables $x_1, \ldots, x_n$

Example 3.1.2 (Equation 65 "Asterix law")**.** Given Equation 65, called "Asterix law":

$$\underbrace{x = y \diamond (x \diamond \underbrace{(y \diamond x)}_{z})}_{u}, \tag{3.1}$$

it is said to be locally satisfied by R if the following holds:

$$\text{if } R(y, x, z) \text{ and } R(x, z, u) \Rightarrow R(y, u, x).$$

We now provide an example with a more complex expression on both sides of the equation.

Example 3.1.3. Now let us consider Equation 4673:

$$\underbrace{(x \diamond y)}_{u} \diamond z = \underbrace{(x \diamond z)}_{v} \diamond y.$$

In this case, R must be such that:

- if $R(x, y, u)$, $R(u, z, w)$ and $R(x, z, v) \Rightarrow R(v, y, w)$;
- if $R(x, z, v)$, $R(v, y, w)$ and $R(x, y, u) \Rightarrow R(u, z, w)$.

Note that in the case where R is associated with an operation $\diamond$ defined on the entire magma, then in that case, R satisfies the law $w_1 = w_2$ if and only if the operator $\diamond$ satisfies the same law.

Suppose we have a relation R that satisfies a certain theory Γ, that is a set of universal laws, but is finitely supported, meaning there are only a finite number of triples (x, y, z) such that $R(x, y, z)$ holds: this implies that the operator $\diamond$ associated with R is only partially defined on the magma. Therefore, we can find elements $a, b \in G$ such that $a \diamond b$ is not defined. If the set G is infinite, then it is possible to find a "fresh value" c that satisfies the following conditions:

(novel-1) $c \neq a, b$

(novel-2) if $R(x, y, z) \Rightarrow c \neq x, y, z$;

(undefined) $R(a, b, x)$ does not hold for any $x \in G$.

Definition 3.1.4 (Γ *greedily extensible*)**.** We say that a theory Γ is *greedily extensible* if, whenever R is a finitely supported ternary relation obeying Γ, and a, b, c are constants obeying (novel-1), (novel-2) and (undefined), there exists an extension R' of R such that:

- $\forall\ x, y, z \in G : R(x, y, z) \Rightarrow R'(x, y, z)$;
- R' is finitely supported and obeys Γ;
- $R'(a, b, c)$ holds.

Therefore, informally, the idea is to extend R with R' by defining the operation $\diamond$ between a and b as $a \diamond b := c$, ensuring that the axioms of R continue to be satisfied.

Note that if the theory Γ is *greedily extensible*, every partially defined ternary relation R that satisfies Γ on an infinite set G, by iterating the procedure just described, can be extended to a globally defined relation that continues to satisfy Γ. Indeed we can choose a pair (a, b) for which $a \diamond b$ is not yet defined, a "fresh value" c, which satisfies the conditions stated above, to define as $a \diamond b$, exploiting the greedily extensible property of Γ, and then we take a direct limit of the sequence of relations thus produced. In this way, we are able to construct a magma that satisfies a specific theory Γ, but does not satisfy other laws of the form $w_1(x_1, \ldots, x_n) = w_2(x_1, \ldots, x_n)$, where both sides are well-defined but do not yield the same output.

However, there are some theories that are not greedily extensible without making certain modifications: it is often not sufficient to simply take a fresh value and assign it as the result of the operation $\diamond$ between two elements for which the operation is not yet defined, but additional checks are required in our extension. For example, let us consider a theory Γ that includes VLT and "Asterix law" (Eq. (3.1)). Consider a ternary relation R with finite support, which we want to extend and ensure it is globally defined, while respecting Γ and $a, b, c \in G$ such that (novel-1), (novel-2) and (undefined) hold. At this

point, we attempt to construct R', a finite extension of R, that continues to satisfy Γ, as explained previously, by requiring that:

$$R'(x,y,z) \iff R(x,y,z) \text{ or } (x,y,z) = (a,b,c).$$

However, these conditions are not sufficient: indeed, if there exists a z such that $R(z,a,b)$, it follows that $R'(z,a,b)$. Substituting $x = a$, $y = z$ into Eq. (3.1) and knowing that $R'(a,b,c)$, it follows that R' satisfies Eq. (3.1) if and only if $R'(z,c,a)$. Therefore, we modify the construction of the extension R' by requiring that:

- $\forall\ x,y,z \in G : R(x,y,z) \Rightarrow R'(x,y,z)$;
- $R'(a,b,c)$;
- if $\exists\ z : R(z,a,b) \Rightarrow R'(z,c,a)$.

This is still not sufficient. In fact, if $R(b,a,b)$ holds, then from the new construction we have that $R'(b,c,a)$. Therefore, knowing $R'(a,b,c)$ and $R'(b,c,a)$, by substituting $x = a$ and $y = b$ into Eq. (3.1), it is necessary to have $R'(a,a,b)$ in order for R' to satisfy Eq. (3.1). We need to add an additional condition for R':

- if $R(b,a,b) \Rightarrow R'(a,a,b)$.

At this point, a possible violation of VLT could be checked: if we had $R(a,a,z)$ for some $z \neq b$, then from the construction of R' we would also have $R'(a,a,z)$, which would violate VLT. Therefore, this time we need to modify the theory Γ, adding an additional requirement:

- if $R(y,x,y) \Rightarrow R(x,x,y)$. (3.2)

Upon checking if there are any further modifications to be made, we notice that from the new construction of R' and the addition of Eq. (3.2) in Γ:

- from Eq. (3.2): $R(b,a,b) \Rightarrow R(a,a,b)$;

- knowing that if $\exists\ z : R(z, a, b) \Rightarrow R'(z, c, a)$, by setting $z = b$, we obtain that $R(b, a, b) \Rightarrow R'(b, c, a)$.

Consequently, also from the construction of R':

- as in the previous point, knowing that if $\exists z : R(z, a, b) \Rightarrow R'(z, c, a)$, by setting $z = a$, we obtain that $R(a, a, b) \Rightarrow R'(a, c, a)$.

Since R' must satisfy Γ, it must also satisfy Eq. (3.2), so:

- $R'(a, c, a) \Rightarrow R'(c, c, a)$.

Therefore, we construct R' in such a way that it satisfies the following requirements:

- $\forall\ x, y, z \in G : R(x, y, z) \Rightarrow R'(x, y, z)$;
- $R'(a, b, c)$;
- if $\exists\ z : R(z, a, b) \Rightarrow R'(z, c, a)$;
- if $R(b, a, b) \Rightarrow R'(b, c, a), R'(a, c, a), R'(c, c, a)$.

Finally, it can be proven, for example, with the help of automated theorem provers, that if R is finitely supported and satisfies Γ, which includes VLT, Eq. (3.1), and Eq. (3.2), and a, b, c satisfy (novel-1), (novel-2) and (undefined), then the R' just constructed is finitely supported and satisfies Γ. This shows that Γ, with some modification, is greedily extensible.

In the next subsection, we will provide an informal idea of how to visualize the expansion of a magma using the greedy construction, while in the next chapter, we will use this algorithm to prove $1516 \not\vdash 255$, also analyzing the formalization in Lean.

3.1.1 An example of a Greedy construction

We aim to construct a solution for Equation 1648:

$$x = (x \diamond y) \diamond ((x \diamond y) \diamond y).$$

We define a *partial solution* as a partial function $\diamond : E \to \mathbb{N}$, where $E \subset \mathbb{N} \times \mathbb{N}$ is a finite subset (which we can also think of as a ternary relation R with finite support), such that:

i) if $x \diamond y = z$ and $z \diamond y = w$, then $z \diamond w = x$;

ii) if $z \diamond x = y,\ w \diamond x = y,$ then $z = w$.

That is, in point i), we required that the operation $\diamond$ locally obeys Equation 1648, according to Definition 3.1.1, while in point ii) we require that $\diamond$ be left-injective. We aim to extend this solution so that it is globally defined on $\mathbb{N} \times \mathbb{N}$ while continuing to satisfy conditions i) and ii). To illustrate this, we demonstrate how the greedy algorithm construction works in this case, showing how the operation $\diamond$ is gradually extended. We begin with a partial finite model defined in Table 3.1.

$\diamond$	**1**	**2**	**3**
1	2	3	3
2			
3			

Table 3.1: operation $\diamond$

The operation $\diamond$ provides a partial solution to Equation 1648, satisfying both conditions i) and ii). We now seek three elements a, b, and c that satisfy (novel-1), (novel-2), and (undefined) to define $a \diamond b := c$. By choosing $c = 4$, $a = 2$, and $b = 1$, we extend $\diamond$ to define $2 \diamond 1 := 4$.

Next, we verify that $\diamond$ remains a partial solution, particularly ensuring it continues to satisfy condition i). We proceed by defining new operations involving the "fresh value" c, with c appearing in the conclusion of i), while only previously defined operations appear in the hypotheses. By setting $w = 4$, we modify i) as follows:

$$x \diamond y = z,\ z \diamond y = 4 \Rightarrow z \diamond 4 = x. \tag{3.3}$$

Taking $z = 2$ and $y = 1$ (since $2 \diamond 1 = 4$), we now look for x such that $x \diamond 1 = 2$. Referring to Table 3.1, we see that $1 \diamond 1 = 2$, so $x = 1$. Substituting this into Eq. (3.3), we obtain:

$$1 \diamond 1 = 2,\ 2 \diamond 1 = 4 \Rightarrow 2 \diamond 4 = 1. \tag{3.4}$$

Thus, to ensure $\diamond$ respects i), we must define $2 \diamond 4 := 1$. The expansion of the model is shown in Table 3.2. We continue by defining additional fresh values.

$\diamond$	**1**	**2**	**3**	**4**
1	2	3	3	
2	4			1
3				
4				

Table 3.2: First $\diamond$ extension

First, we choose $c = 5$ and define $2 \diamond 2 := 5$, and then $c = 6$ with $3 \diamond 1 := 6$. However, in these cases, we are unable to use condition i) to define other operations involving 5 and 6. Next, we explore the case where $c = 7$, and extend $\diamond$ by defining $1 \diamond 4 := 7$. To maintain consistency with i), we proceed as follows:

- We choose parameters such that only previously defined operations appear in the hypotheses, and the new value 7 appears in the conclusion. Taking $w = 7$, $z = 1$, and $y = 4$, we modify i) as:

$$x \diamond 4 = 1,\ 1 \diamond 4 = 7 \Rightarrow 1 \diamond 7 = x. \tag{3.5}$$

- Since $2 \diamond 4 = 1$ is already defined, we choose $x = 2$. From Eq. (3.5), we then define $1 \diamond 7 := 2$ to ensure $\diamond$ remains a partial solution.

After several extensions, the operation $\diamond$ is globally defined as follows:

$\diamond$	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	2	3	3	7			2
2	4	5		1			
3	6						
4							
5							
6							
7							

Table 3.3:

By iterating this process, the operation $\diamond$ can gradually be defined globally on $\mathbb{N} \times \mathbb{N}$, such that it satisfies Equation 1648.

3.2 Linear Magmas

Before proceeding, it is necessary to introduce *linear magmas*, which serve as a valuable resource for counterexamples and will be extensively used in the subsequent discussion.

Definition 3.2.1 (Linear Magma)**.** We say that $\mathcal{M} = (M, \diamond)$ is a *linear magma* if the carrier M is a ring $(M, +)$ (which may be commutative or non-commutative, finite or infinite), and the operation $\diamond$ is given by $x \diamond y := ax + by$, where $a, b \in \mathrm{End}(M)$.

Let us now take an abelian group $M = (M, +)$, on which we define a magma linear operation as follows:

$$s \diamond t := as + bt, \;\; a, b \in \mathrm{End}(M).$$

We want to choose a, b such that $\diamond$ satisfies Equation $E : w_{E,1}(x_1, \ldots, x_n) = w_{E,2}(x_1, \ldots, x_n)$. Notice that in the magma M, every expression $w(x_1, \ldots, x_n)$ is of the form:

$$w(x_1, \ldots, x_n) = \sum_{i=1}^{n} P_{w,i}(a, b)x_i,$$

where $P_{w,i}(a,b)$ are **non-commutative** polynomials with natural coefficients in the variables a and b. Let us now see some examples to understand better:

$$s \diamond s = (a+b)s$$
$$t \diamond (s \diamond s) = at + b(s \diamond s) = at + (ba + b^2)s$$
$$(t \diamond (s \diamond s)) \diamond t = a^2t + (aba + b^2a)s + bt = (a^2 + b)t + (aba + b^2a)s.$$

Therefore, Equation E will be of the form:

$$\sum_{i=1}^{n} P_{w_{E,1},i}(a,b)x_i = \sum_{i=1}^{n} P_{w_{E,2},i}(a,b)x_i,$$

so M obeys E if:

$$P_{w_{E,1},i}(a,b) = P_{w_{E,2},i}(a,b), \quad \text{for} \quad i = 1, \ldots, n. \tag{3.6}$$

For example, given Equation 1110

$$x = y \diamond ((y \diamond (x \diamond x)) \diamond y),$$

we develop the right-hand side according to the definition of $\diamond$ on M and obtain

$$x = (baba + bab^2)x + (ba^2 + b^2 + a)y,$$

from which it follows that M satisfies this law if and only if:

$$\begin{cases} baba + bab^2 = 1, \\ ba^2 + b^2 + a = 0. \end{cases} \tag{3.7}$$

Example 3.2.2 (Commutative counterexample)**.** Linear magmas, on various fields, are primarily used to establish counterexamples. So given Equation 1286:

$$x = y \diamond (((x \diamond y) \diamond x) \diamond y), \tag{3.8}$$

and Equation 3 ("Idempotent law"):

$$x = x \diamond x, \tag{3.9}$$

we want to prove that $1286 \not\vdash 3$ by constructing a commutative counterexample using linear magmas. We must therefore choose a, b in a ring M such that Eq. (3.8) holds but Eq. (3.9) does not, i.e., a and b must obey the following conditions:

$$\begin{cases} a + ba^3 + bab = 1, \\ a + ba^2b + b^2 = 0. \end{cases} \tag{3.10}$$

and

$$a + b \neq 1. \tag{3.11}$$

Thus, we choose $M := \mathbb{Z}/p\mathbb{Z}$ with $p = 11$, setting $a := 1$ and $b := 7$. In this way, we have defined a commutative counterexample $\mathcal{M} = (M, \diamond)$, where $x \diamond y := 1x + 7y$, with $M = \mathbb{Z}/11\mathbb{Z}$.

3.3 Magma Cohomology

Magma cohomology is a variant of group cohomology (or rather, it can be thought of as a generalization of it) that was discovered while working on some proofs for the project, and later explored in more detail. More precisely, a new "magma extension" strategy was discovered, which allowed for the resolution of many finite implications. This strategy involves extending a magma G by taking the set $G \times M$, where M is a ring or an abelian group, and defining an operation $(x, s) \diamond (y, t) := (x \diamond y, as + bt + f(x, y))$, where a, b are coefficients of some linear magma model $s \diamond t = as + bt$, and $f : M \times M \to G$ is a "cocycle" that satisfies a certain "cocycle type equation" (we will explain it in detail later on). We now aim to analyze this construction to understand why it is referred to as cohomology. We begin by briefly reviewing the concept of group cohomology and explore how it can be extended to the case of magmas.

Let G be an abelian group and M be a G-module. Define the set $C^n(G, M) := \mathrm{Hom}_{\mathrm{Set}}(G^n, M)$, i.e. the set of functions $f : G^n \to M$.

Remark 3.3.1. Let e be the identity element of the abelian group G, and $G^0 = \{e\}$. Therefore, we have:

$$C^0(G, M) = \mathrm{Hom}(\{e\}, M) \simeq M.$$

For all n, define the map $d^n : C^n(G, M) \to C^{n+1}(G, M)$ as follows:

$$\begin{aligned} d^n\varphi(g_1, \ldots, g_{n+1}) := \; & g_1\varphi(g_2, \ldots, g_{n+1}) + \sum_{i=1}^{n}(-1)^i\varphi(g_1, \ldots, g_ig_{i+1}, \ldots, g_{n+1}) \\ & + (-1)^{n+1}\varphi(g_1, \ldots, g_n). \end{aligned}$$

It is easy to see that $d^{n+1} \circ d^n = 0$ for all n, so we have the complex:

$$0 \to C^0(G, M) \xrightarrow{d^0} C^1(G, M) \xrightarrow{d^1} C^2(G, M) \xrightarrow{d^2} C^3(G, M) \to \ldots$$

and the cohomology group is defined as $H^n(G, M) := \frac{\ker d^n}{\mathrm{Im}(d^{n-1})}$. For further details, the reader is referred to [Bro12].

Let us see how this construction can be generalized in the case of magmas. Let G be a magma and M a linear magma, we now consider the extensions of G through M, i.e. the magmas with values in the set $G \times M$, with an operation $\diamond$ defined as follows:

$$(x, s) \diamond (y, t) := (x \diamond y, as + bt + f(x, y)), \tag{3.12}$$

where $f : G \times G \to M$. It can be proven inductively that every expression $w((x_1, s_1), \ldots, (x_n, s_n))$ is of the form:

$$\begin{aligned} w((x_1, s_1), \ldots, (x_n, s_n)) = \Big(& w(x_1, \ldots, x_n), \sum_{i=1}^{n} P_{w,i}(a, b)x_i \\ & + \sum_{w_1 \diamond w_2 \leq w} Q_{w,w_1\diamond w_2}(a, b) f\big(w_1(x_1, \ldots, x_n), w_2(x_1, \ldots, x_n)\big)\Big), \end{aligned}$$

where $Q_{w,w_1\diamond w_2}(a, b)$ is a non-commutative monomial in variables a, b. Consider now Equation E:

$$w_{E,1}((x_1, s_1), \ldots, (x_n, s_n)) = w_{E,2}((x_1, s_1), \ldots, (x_n, s_n)),$$

where both sides are expanded as shown above. Suppose that G and M obey E, so the condition Eq. (3.6) holds. Then, the operation $\diamond$ on $G \times M$ obeys E if and only if:

$$\begin{aligned} & \sum_{w_1 \diamond w_2 \leq w_{E,1}} Q_{w_{E,1},w_1\diamond w_2}(a, b) f(w_1(x_1, \ldots, x_n), w_2(x_1, \ldots, x_n)) \\ = & \sum_{w_1 \diamond w_2 \leq w_{E,2}} Q_{w_{E,2},w_1\diamond w_2}(a, b) f(w_1(x_1, \ldots, x_n), w_2(x_1, \ldots, x_n)). \end{aligned} \tag{3.13}$$

For example, let us examine the conditions under which this extension obeys Equation 1110:

$$(x,s) = (y,t) \diamond (((y,t) \diamond ((x,s) \diamond (x,s))) \diamond (y,t)). \tag{3.14}$$

We now proceed with the calculations on the right-hand side:

$$\begin{aligned}(y,t) \diamond (((y,t) \diamond ((x,s) \diamond (x,s))) \diamond (y,t)) = (y \diamond ((y \diamond (x \diamond x)) \diamond y),\\ (baba + bab^2)s + (ba^2 + b^2 + a)t + babf(x,x) + baf(y, x \diamond x)\\ + bf(y \diamond (x \diamond x), y) + f(y, (y \diamond (x \diamond x)) \diamond y).\end{aligned} \tag{3.15}$$

By substituting Eq. (3.7) (since we are assuming that before the extension, M obeys 1110) and Eq. (3.15) into Eq. (3.14), and knowing that G obeys 1110 (therefore we have that $x = y \diamond ((y \diamond (x \diamond x)) \diamond y)$), we obtain:

$$(x,s) = (x, s + babf(x,x) + baf(y, x \diamond x) + bf(y \diamond (x \diamond x), y) + f(y, (y \diamond (x \diamond x)) \diamond y)).$$

Therefore, Equation 1110 is obeyed if and only if the condition Eq. (3.13) holds, which in this case is equivalent to:

$$babf(x,x) + baf(y, x \diamond x) + bf(y \diamond (x \diamond x), y) + f(y, (y \diamond (x \diamond x)) \diamond y) = 0,$$

for all $x, y \in G$.

Definition 3.3.2 (*E-cocycle*)**.** Given an Equation $E : w_{E,1}((x_1,s_1), \ldots$ $\ldots, (x_n,s_n)) = w_{E,2}((x_1,s_1), \ldots, (x_n,s_n))$, we say that $f : G \times G \to M$ is a *E-cocycle* if Eq. (3.13) holds, and we denote with $Z^2_E(G,M)$ the space of such *E-cocycles*.

Remark 3.3.3. $Z^2_E(G,M)$ is an abelian group, and each *E-cocycle* defines a magma on $G \times M$ obeying E.

Remark 3.3.4. If $f = 0$, the defined magma is precisely the direct product of G and M.

Given a function $g : G \to M$, we define the bijection $\varphi : G \times M \to G \times M$ by $(x,s) \mapsto (x, s + g(x))$, which conjugates Eq. (3.12) with the following law:

$$(x,s) \diamond (y,t) = (x \diamond y, as + bt + f(x,y) + g(x \diamond y) - ag(x) - bg(y)). \tag{3.16}$$

Lemma 3.3.5. *Eq.* (3.12) *and Eq.* (3.16) *are conjugate.*

Proof. We define the maps $\psi, \tilde{\psi} : (G \times M) \times (G \times M) \to G \times M$ by:

$$\psi((x,s),(y,t)) := (x \diamond y, as + bt + f(x,y)),$$

$$\tilde{\psi}((x,s),(y,t)) := (x \diamond y, as + bt + f(x,y) + g(x \diamond y) - ag(x) - bg(y)),$$

which correspond to Eq. (3.12) and Eq. (3.16), respectively. We want to prove that the following diagram

$$\begin{array}{ccc} (G \times M) \times (G \times M) & \xrightarrow{\tilde{\psi}} & G \times M \\ \downarrow{\scriptstyle \varphi^{-1}} & & \uparrow{\scriptstyle \varphi} \\ (G \times M) \times (G \times M) & \xrightarrow[\psi]{} & G \times M \end{array}$$

commutes, where $\varphi^{-1}((x,s),(y,t)) = ((x, s - g(x)), (y, t - g(y))$. Then we apply ψ and we have that:

$$\psi((x, s - g(x)), (y, t - g(y)) = (x \diamond y, as - ag(x) + bt - bg(y) + f(x,y)).$$

Only applying φ remains, and we obtain $(x \diamond y, as + bt + f(x,y) + g(x \diamond y) - ag(x) - bg(y))$, that is equal to $\tilde{\psi}((x,s),(y,t))$. Therefore, we have shown that

$$\tilde{\psi} = \varphi^{-1} \circ \psi \circ \varphi. \qquad \square$$

Since they are conjugate, the new operation satisfies E if and only if the original operation does.

Definition 3.3.6. We call a function $f : G \times G \to M$ a *coboundary* if it is of the form $f(x,y) = g(x \diamond y) - ag(x) - bg(y)$, for some $g : G \to M$. We denote with $B^2(G,M)$ the space of coboundaries.

Remark 3.3.7. We can add a *coboundary* to an *E-cocycle* and still obtain an *E-cocycle*.

Remark 3.3.8. $B^2(G,M)$ is a subgroup of $Z^2_E(G,M)$.

Finally, we can define the *E-cohomology* $H^2_E(G, M)$ as:

$$H^2_E(G, M) := \frac{Z^2_E(G, M)}{B^2(G, M)}.$$

Lemma 3.3.9. *If $E \vdash E'$, then $H^2_E(G, M) \subseteq H^2_{E'}(G, M)$.*

Proof. We just have to prove that if f is an E-cocycle, is also an E'-cocycle. But being f an E-cocycle, this means that E holds, so thanks to the hypotesis also E' holds. But E' holds if and only if f is an E'-cocycle. □

A very useful application of this theory is that, to prove that $E \nvdash E'$, it is sufficient to find a magma G and a linear magma M, which satisfy both E and E', such that:

$$H^2_E(G, M) \not\subseteq H^2_{E'}(G, M).$$

This leads to a computational approach to refutations, as these groups can be computed by linear algebra.

Remark 3.3.10. The *E-cohomology* group just defined can also be viewed in terms of a partial chain complex

$$0 \to C^0(G, M) \xrightarrow{d^0} C^1(G, M) \xrightarrow{d^1} C^2(G, M) \xrightarrow{d^2_E} C^n(G, M),$$

where $d^0 : C^0(G, M) \to C^1(G, M)$ is the zero map, the first coboundary map $d^1 : C^1(G, M) \to C^2(G, M)$, which does not depend on Equation E, is defined as:

$$d^1 f(x, y) := f(x \diamond y) - (f(x) \diamond f(y)) = f(x \diamond y) - af(x) - bf(y), \quad (3.17)$$

and the second map $d^2_E : C^2(G, M) \xrightarrow{d^2_E} C^n(G, M)$, which instead depends on Equation E, is defined as:

$$\begin{aligned} d^2_E f(x_1, \ldots, x_n) := &\sum_{w_1 \diamond w_2 \leq w_{E,1}} Q_{w_{E,1}, w_1 \diamond w_2}(a, b) f(w_1(x_1, \ldots, x_n), w_2(x_1, \ldots, x_n)) \\ &- \sum_{w_1 \diamond w_2 \leq w_{E,2}} Q_{w_{E,2}, w_1 \diamond w_2}(a, b) f(w_1(x_1, \ldots, x_n), w_2(x_1, \ldots, x_n)), \end{aligned} \quad (3.18)$$

where $f \in C^2(G, M),$ and $(x_1, \ldots, x_n) \in G^n$. Obviously, from the definition of d^0, we have $d^1 \circ d^0 = 0$, while from the fact that the coboundary maps, which exactly correspond to Im d^1, are also E-cocycles, we obtain that: $d^2_E \circ d^1 = 0$. Therefore, observing that $\ker d^2_E$ corresponds to the space $Z^2_E(G, M)$ of E-cocycles, we have that the E-cohomology group $H^2_E(G, M)$ is equal to $\frac{\ker d^2_E}{\text{Im } d^1}$, that is the second cohomology group of the complex.

Chapter 4

Exploring Equations

In this chapter, we analyze the most significant equations of the project, including those on which I have focused the most. Specifically, after briefly introducing the *commutative* and *associative* laws, we examine Equation 677, where the concepts of *linear magma* and *free magma* emerge. The non-implication $677 \not\vdash 255$, for linear magmas, remains the only one that is conjectured but not yet proven. Finally, we prove that $1516 \not\vdash 255$, by constructing a counterexample using the greedy algorithm construction technique introduced in the previous chapter. We will also discuss the formalization work of this non-implication that we have undertaken.

4.1 Commutative and Associative law

The *commutative law*: $x \diamond y = y \diamond x$ and the *associative law*: $x \diamond (y \diamond z) = (x \diamond y) \diamond z$ are considered fundamental equations in the study of algebraic structures, and they are probably the most important among all the equations in the project. They are not equivalent to any other equation in the project, and below we can visualize which equations are implied by them.

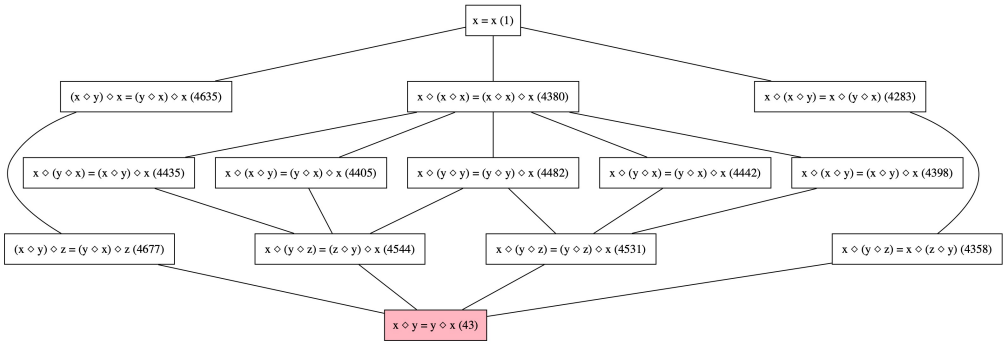

Figure 4.1: Commutative law

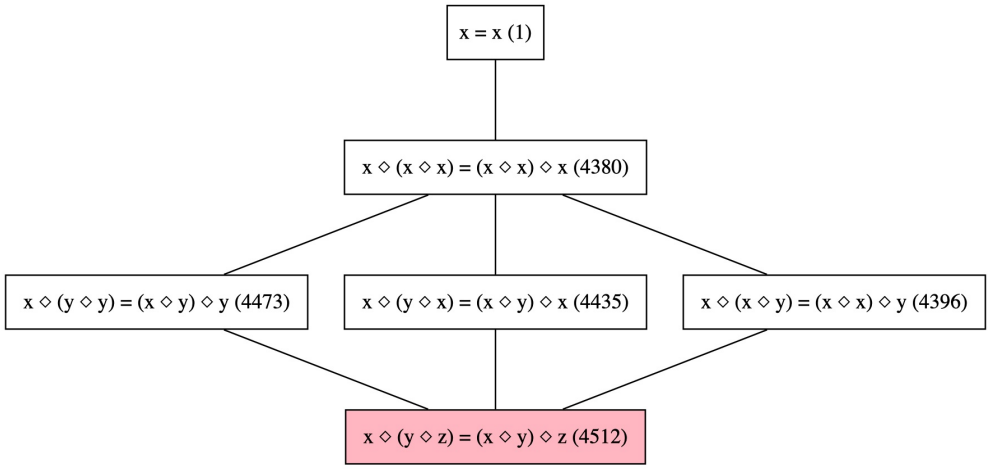

Figure 4.2: Associative law

4.2 Equation 677

Equation 677 is still being studied by many mathematicians working on the project. In fact, the only non-implication that still lacks a proof, either formal or informal, is $677 \not\vdash 255$ in the case of finite magmas. For general magmas, an infinite counterexample has been found, and it has been conjectured that 677 does not imply 255 also for finite magmas, but no finite counterexample

has been found yet. Automatic tools, such as *Mace4*, *Prover9*, and *Magma*[1], are being used to generate finite magmas with different characteristics and sizes that satisfy 677 but not 255, but for now none has succeeded.

First, we present Equation 677:

$$x = y \diamond (x \diamond ((y \diamond x) \diamond y)), \tag{4.1}$$

and Equation 255:

$$x = ((x \diamond x) \diamond x) \diamond x. \tag{4.2}$$

Using the notation $L_y x := y \diamond x,\ R_y x := x \diamond y,\ Sx := x \diamond x$, we can rewrite 677 as:

$$x = L_y L_x L_{L_y x} y = L_y(x \diamond R_y L_y x), \tag{4.3}$$

and 255 as:

$$x = (Sx \diamond x) \diamond x. \tag{4.4}$$

Now we provide and prove some basic properties of 677 magmas.

Lemma 4.2.1. *Let M be a finite* 677 *magma. Then:*

i) the left multiplication operators $L_y : M \to M$ are all invertible, and $L_y^{-1} x = x \diamond R_y L_y x$;

ii) If $x, y \in M$ and $y \diamond x = x$, then $y = Sx \diamond x$. In particular, Equation 255 *holds if and only if $y \diamond x = x$ is solvable for every x;*

iii) for all $x, y \in M$, we have $x = L_y x \diamond R_y L_y^2 x$.

Proof. We can easily see from Eq. (4.3) that L_y is surjective, and thus injective because we are working with finite magmas, meaning that L_y is invertible.

[1]Magma is a Computer Algebra system designed to solve problems in algebra, number theory, geometry and combinatorics that may involve sophisticated mathematics and which are computationally hard. A key feature is the ability to construct canonical representations of structures, thereby making possible such operations as membership testing, the determination of structural properties and isomorphism testing. We provide the following link for further information: `https://magma.maths.usyd.edu.au/magma/`.

Furthermore, from Eq. (4.3) we have that $L_y^{-1}x = x \diamond R_y L_y x$. For ii), we have that $L_y x = x$. Therefore, i) becomes $x = x \diamond R_y x = L_x L_x y$, and using the fact that L_x is invertible from i), we obtain:

$$y = L_x^{-1}L_x^{-1}x \overset{i)}{=} L_x^{-1}(x \diamond R_x L_x x) = L_x^{-1}(L_x R_x L_x x) = R_x L_x x = Sx \diamond x.$$

For iii), we use i) by substituting x with $L_y x$. □

Now we provide some characterizations of 255, which will be useful later on.

Lemma 4.2.2. *Let M be a finite 677 magma, and let $x \in M$. Then the following statements are equivalent:*

i) $R_x(Sx \diamond x) = x$;

ii) $R_x y = x$ *has the unique solution* $y = Sx \diamond x$;

iii) $R_x y = x$ *has a solution;*

iv) $R_x L_x z = x$ *has a solution;*

v) $L_x R_x z = z$ *has a solution;*

vi) $R_x L_x y = y$ *has a solution;*

vii) $L_x S y = y$ *has a solution.*

Proof. **ii) → i):** the proof is straightforward.

i) → iii): From i), we know that 255 holds. Therefore, from ii) in Lemma 4.2.1, it follows that $\forall x, \exists y : y \diamond x = x$, which is equivalent to iii).

iii) → ii): This result is exactly Item (ii).

iii) ↔ iv): the proof follows immediately by exploiting the surjectivity of L_x.

v) → vi): from v), $\exists\, z : L_x R_x z = z$, and since L_x is invertible, $\exists\, y : z = L_x y$. Thus, exploiting again the invertibility of L_x, we have that:

$$L_x R_x z = z \iff \underbrace{R_x z}_{R_x L_x y} = \underbrace{L_x^{-1} z}_{y} \iff R_x L_x y = y,$$

from which we obtain vi).

vi) → v): we repeat the same reasoning as above, proceeding backwards.

vii) → vi): knowing that $Sy = L_y y$, we can rewrite vii) as $L_x L_y y = y$. Moreover, we know that:

$$y \overset{(677)}{=} L_x L_y L_{L_x y} x = L_x L_y \underbrace{(L_x y \diamond x)}_{R_x L_x y} = L_x L_y R_x L_x y.$$

Thus, we have:

$$L_x L_y y = y = L_x L_y R_x L_x y,$$

and by exploiting left-injectivity, we obtain vi).

vi)→ vii): we repeat the same reasoning as above, proceeding backwards.

iv) ↔ v): the proof is similar to the one carried out above. □

This characterization will be useful to prove that linear 677 magmas cannot be used as counterexamples to refute the implication $677 \not\vdash 255$, since the following lemma holds.

Lemma 4.2.3. *Let M be a finite magma obeying* 677*, which is linear in the sense that M is an abelian group and $x \diamond y = \alpha x + \beta y + c$, for some endomorphisms $\alpha, \beta : M \to M$ and a constant c. Then M obeys Equation* 255*.*

Proof. From ii) of Lemma 4.2.1, it is sufficient to prove that $y \diamond x = x$ is solvable for every x, so it is enough to prove that R_x is surjective. However, in the case of finite magmas, it is equivalent to prove that R_x is injective.

Suppose that $R_x y = R_x y'$, which is equivalent to $L_y x = L_{y'} x$. Thus, using the definition of $\diamond$ on M, we have:

$$y \diamond x = y' \diamond x \iff \alpha y + \beta x + c = \alpha y' + \beta x + c \iff \alpha y = \alpha y',$$

from which we deduce that $L_y = L_{y'}$. Therefore, applying Eq. (4.3), we get:

$$L_y L_x L_{L_y x} y = x = L_{y'} L_x L_{L_{y'} x} y' \overset{L_y = L_{y'}}{=} L_y L_x L_{L_y x} y',$$

and by exploiting the injectivity of L_y, L_x, and $L_{L_y x}$, we obtain that $y = y'$. $\square$

Another technique often used to construct magmas is to start from a magma that satisfies both equations and extend it linearly, hoping that the new magma continues to satisfy the first equation but no longer satisfies the second. However, this approach also has no hope of yielding a counterexample for $677 \vdash 255$, as the following lemma demonstrates.

Lemma 4.2.4. *Let us consider a* 677 *magma with carrier $G \times M$, where G is a magma obeying* 677 *and* 255*, M is an abelian group, and the operation $\diamond$ on $G \times M$ is defined by:*

$$(x, s) \diamond (y, t) := (x \diamond y, \alpha_{x,y} s + \beta_{x,y} t + c_{x,y}),$$

for some endomorphisms $\alpha_{x,y}, \beta_{x,y} : M \to M$ and costants $c_{x,y}$. Then $G \times M$ obeys 255*.*

Proof. From Lemma 4.2.1, it is sufficient to prove that for every $(y, t) \in G \times M$, the equation $(x, s) \diamond (y, t) = (y, t)$ is solvable. From the definition of $\diamond$, we have that $(x, s) \diamond (y, t) := (x \diamond y, \alpha_{x,y} s + \beta_{x,y} t + c_{x,y})$, so we need to prove that there exists (x, s) such that:

$$(x \diamond y, \alpha_{x,y} s + \beta_{x,y} t + c_{x,y}) = (y, t).$$

Knowing that G obeys 255, we have that there exists x such that $x \diamond y = y$. Now, we only need to prove that the map $s \mapsto \alpha_{x,y} s + \beta_{x,y} t + c_{x,y}$ is surjective. In the case of finite magmas, this is equivalent to proving that it

is injective. Suppose, for the sake of contradiction, that given $s \neq s'$, we have $\alpha_{x,y}s + \beta_{x,y}t + c_{x,y} = \alpha_{x,y}s' + \beta_{x,y}t + c_{x,y}$, that is, $\alpha_{x,y}s = \alpha_{x,y}s'$. Consequently, we have:

$$L_{(x,s)}(y,t') = L_{(x,s')}(y,t'), \quad \forall\, t' \in M. \tag{4.5}$$

Since G obeys 255 and 677, it follows that:

$$x \diamond y = y \overset{4.1}{=} x \diamond (y \diamond ((x \diamond y) \diamond x)),$$

From left-invertibility, we obtain $y = y \diamond ((x \diamond y) \diamond x)$, so $L_{(y,t)}L_{L_{(x,s)}(y,t)}(x,s) = (y,t) \diamond ((x,s) \diamond (y,t) \diamond (x,s))$ will be of the form (y,t') for some t'. Knowing that $G \times M$ obeys 677, we have:

$$L_{(x,s)}L_{(y,t)}L_{L_{(x,s)}(y,t)}(x,s) = (y,t) = L_{(x,s')}L_{(y,t)}L_{L_{(x,s')}(y,t)}(x,s') \overset{4.5}{=}$$
$$L_{(x,s)}L_{(y,t)}L_{L_{(x,s)}(y,t)}(x,s'),$$

and using left-invertibility again, we obtain $s = s'$, which is a contradiction. □

4.2.1 A finite non-right-cancellative example

Now, we provide an example of how to construct 677 magmas, specifically non-right-cancellative magmas. This does not directly bring us any closer to refuting $677 \vdash 255$, but it may offer some broader intuition as to what 677 finite magmas might look like.

Let us start from a 677 magma G and a set M. We want to extend the magma structure from G to $G \times M$, ensuring it continues to obey 677. Therefore, for each pair x, y, we define a binary operation $\diamond_{x,y} : M \times M \to M$ such that:

$$s = t \diamond_{y,L_y^{-1}x} \left(s \diamond_{x,(y \diamond x) \diamond y} \left((t \diamond_{y,x} s) \diamond_{y \diamond x, y} t\right)\right). \tag{4.6}$$

Consequently, we define the operation $\diamond$ on $G \times M$ as:

$$(x,s) \diamond (y,t) := (x \diamond y, t \diamond_{x,y} s).$$

It can be easily seen, using Eq. (4.6), that it satisfies 677. Moreover, $\diamond$: $(G \times M) \times (G \times M) \to (G \times M)$ is right-injective if the operation $\diamond$ defined on

G and $\diamond_{x,y}$ for all $x, y \in M$ are both right-injective. Let M be a field that admits a primitive cube root of unity ω as well as a primitive fifth root of unity ζ (for example, M could be a field of order 16), we define the following operations:

- $s \diamond^0 t := s - \zeta(t - s)$;
- $s \diamond^+ t := t$;
- $s \diamond^- t := s - \omega(t - s)$.

It can be easily seen that for all $t, s \in M$, the following identities hold:

i) $s = t \diamond^0 (s \diamond^0 ((t \diamond^0 s) \diamond^0 t))$;

ii) $s = t \diamond^+ (s \diamond^+ ((t \diamond^- s) \diamond^- t))$;

iii) $s = t \diamond^- (s \diamond^- ((t \diamond^+ s) \diamond^+ t))$.

Now suppose that G is also a field with a primitive fifth root of unity β, but -1 and $\beta - 1$ are non-zero quadratic non-residues[2] (for example, one can think of G as a field of order 31, with $\beta = 2$). Defining the operation $\diamond$ on G as $x \diamond y = x - \beta(y - x)$, and $\diamond_{x,y}$ as:

$$\diamond_{x,y} = \begin{cases} \diamond^0 & \text{if } x = y, \\ \diamond^+ & \text{if } y - x \text{ is a non-zero quadratic residue,} \\ \diamond^- & \text{if } y - x \text{ is a non-zero quadratic non-residue,} \end{cases}$$

it can be shown that Eq. (4.6) holds, and thus the extension $G \times M$, with $\diamond$ defined as above, obeys 677. Furthermore, it can be easily seen that $\diamond^+$ is not right-cancellative, so this tells us that we have constructed a finite 677 magma that is not right-cancellative.

[2]We say that x is a quadratic residue if $x^4 = x$.

4.2.2 Free 677 Magma

In this subsection, we aim to construct the free 677 magma $\mathcal{M}_{X,677}$ generated by a set of generators X. First, we consider M_X as the free magma generated by a set X through the pairing of its elements $x, y \mapsto (x, y)$. The elements of M_X can be thought of as finite trees with leaves in X. Let us now introduce some notation. Let $w = (x, y) \in M_X$, where we denote by w_L and w_R the left and right components, meaning in this case $w_L = x$ and $w_R = y$. Iteratively, we can define $w_{LL}, w_{RR}, \ldots$, for instance, if $w = ((x, y), z)$, we have $w_{LL} = x$, $w_{LR} = y$, and $w_R = z$. Additionally, we define a partial order relation $<$ on M_X, where $w < w'$ if w is a subtree of w', that is, if either $w \leq w'_L$ or $w \leq w'_R$ holds. At this point, we define an operation $\diamond$ on M_X recursively.

Definition 4.2.5. Let $x, y \in M_X$. If $x < y = (y_L, (x \diamond y_L) \diamond x)$, then define $x \diamond y := y_L$. Otherwise, define $x \diamond y := (x, y)$.

Remark 4.2.6. Note that to define $x \diamond y$ recursively, it is sufficient to know how to calculate $x' \diamond y'$, where $y' < y$. Moreover, since, as all the elements of M_X are finite trees, there are no infinite descending chains in the partial order $<$. Therefore, the operation $\diamond$ is well-defined.

Lemma 4.2.7 (Properties of operation)**.** *Let $x, y \in M_X$ such that $x \diamond y = z$. Then, one of the following statements holds:*

i) $x, z < y = (z, (x \diamond z) \diamond x)$;

ii) $x, y < z = (x, y)$.

Proof. According to Definition 4.2.5, there are two possibilities to define $x \diamond y$:

- If $x < y = (y_L, (x \diamond y_L) \diamond x)$, then $x \diamond y = y_L$, so $z = y_L$. This implies that y has the form $(z, (x \diamond z) \diamond x)$, and consequently $x, z < y$, thus i) holds.

- Otherwise, $x \diamond y = (x, y)$, so $z = (x, y)$ and obviously ii) holds. □

The goal is to prove that the operation $\diamond$ just defined on M_X obeys 677; in particular, it guarantees that if $x \diamond y$ falls into the first case of the definition, then y is of the form $y = (y_L, (x \diamond y_L) \diamond x) = y_L \diamond ((x \diamond y) \diamond x)$.

Lemma 4.2.8 (Additional property)**.** *Let $x, y \in M_X$, then:*

$$x \diamond ((y \diamond x) \diamond y) = (x, (y \diamond x) \diamond y).$$

Proof. First, let us define new variables $z := y \diamond x$, $u := z \diamond y$ and $v := x \diamond u$. We want to prove that $v = (x, u)$. From Lemma 4.2.7, we know that y is (strictly) upper bounded by x or z, z by u or y, and x by u or v. Among all these elements, in light of what has been just stated, the only elements that can be maximal are u and v: if v were maximal, then it would imply that, since $v = x \diamond u$, from Lemma 4.2.7 we have $x, u < v$ (if this were not the case, we would have $x, v < u$, but this is impossible because v is maximal), therefore $v = (x, u)$, thus concluding the proof. It is sufficient to prove that u is not maximal. Suppose, for the sake of contradiction, that it is maximal. Then, using Lemma 4.2.7, we have the following:

- $z, y < u$, so $u = (z, y)$;
- $x, v < u$, hence u has the form $(u_L, (x \diamond u_L) \diamond x)$. From the previous part, we know that $u = (z, y)$, hence $u_L = z$ and $y = (x \diamond z) \diamond x$.

From Lemma 4.2.7, x is upper bounded by $x \diamond z$ or z. Therefore $x \diamond z$, being $y = (x \diamond z) \diamond x$, is upper bounded by x or y. Remembering that y is upper bounded by either z or x, among x, y, z, and $x \diamond z$, the only maximal element can be z. In particular, z is not bounded by x or y, so $z = (y, x)$. We now focus on $x \diamond z$. We know that z is not upper bounded by x or $x \diamond z$, so from Lemma 4.2.7 this means that $x, x \diamond z < z$. Therefore, $x \diamond z = z_L$. Since $z = (y, x)$, we have:

$$y = z_L = x \diamond z. \tag{4.7}$$

In conclusion, from the earlier result, we know that $y = (x \diamond z) \diamond x$. Substituting Eq. (4.7) and by the definition of z, we get:

$$y = y \diamond x = z,$$

which leads to a contradiction, as y cannot be maximal. □

Corollary 4.2.9. *The operation $\diamond$ obeys Eq.* (4.1).

Proof. Using the results obtained so far, we know that it suffices to prove that $y < (x, (y \diamond x) \diamond y)$. Indeed, from Lemma 4.2.8, we know that $x \diamond ((y \diamond x) \diamond y) = (x, (y \diamond x) \diamond y)$, so Eq. (4.1) is equivalent to:

$$x = y \diamond (x, (y \diamond x) \diamond y). \tag{4.8}$$

At this point, if $y < (x, (y \diamond x) \diamond y)$ holds, from the definition of $\diamond$, we obtain $y \diamond (x, (y \diamond x) \diamond y) = x$, and thus Eq. (4.8) is satisfied. Therefore, we simply need to prove that $y < (x, (y \diamond x) \diamond y)$.

Let us define z, u, and v as in the previous proof. We need to show that $y < v$. From Lemma 4.2.8, we know that $v = (x, u)$, so trivially, $x < v$. Since $z := y \diamond x$, from Lemma 4.2.7, y is upper bounded by either x or z:

- If it is upper bounded by x, we know that $x < v$, so $y < v$;
- If it is upper bounded by z, we can reason as follows. z is upper bounded by either y or u: however, we exclude y because we already know that $y < z$, so we cannot have $z < y$. Therefore, we have $y < z < u < v$, hence $y < v$. □

Before proceeding to the last theorem, it is necessary to provide the definition of a free magma for an equational law and an alphabet X.

Definition 4.2.10. Given a set X and an Equation E, the *free magma* for E generated by the alphabet X is a magma $\mathcal{M}_{X,E}$ obeying E, together with a map $i_{X,E} : X \to \mathcal{M}_{X,E}$ that satisfies the following universal property: for every magma $\mathcal{M}$ that obeys E, and for every function $f : X \to \mathcal{M}$, there exists a unique homomorphism $\varphi_{f,E} : \mathcal{M}_{X,E} \to \mathcal{M}$ such that $\varphi_{f,E} \circ i_{X,E} = f$.

Now, let us prove a result that will be fundamental for the last corollary. This lemma determines the carrier of the free magma for 677, which will be a subset of M_X. In fact, $(M_X, \diamond)$ is already a 677 magma, however it is not the free magma for 677 generated by X; the reason is that it cannot guarantee the uniqueness of the homomorphisms $\varphi_{f,677}$.

Lemma 4.2.11. *$\exists \tilde{M} \subseteq M_X$ such that:*

i) $X \subseteq \tilde{M}$;

ii) $\diamond : \tilde{M} \times \tilde{M} \to \tilde{M}$;

iii) $\forall x \in \tilde{M} \setminus X,\ x = x_L \diamond x_R$.

Proof. We define $\tilde{M}$ as the $\diamond$-closure of X, i.e. the smallest set containing X that is closed under the operation $\diamond$. Equivalently, it can be defined iteratively starting from the alphabet X: indeed, it is the free magma generated by X. Thus, an element of $\tilde{M}$ is either a letter of X, or of the form $w_1 \diamond w_2$, where $w_1, w_2 \in \tilde{M}$ and $\diamond$ is defined as in Definition 4.2.5. Therefore, according to this definition, $\tilde{M}$ contains all elements of M_X, except those of the form $(x, (y, (x \diamond y) \diamond x))$, with $x, y \in M_X$, and all elements that can be obtained from these elements using $\diamond$. Therefore, from the definition, it immediately follows that i) e ii) hold.

Let us now consider iii). Take $x = (x_L, x_R) \in \tilde{M}$. The only case to examine is if $x_R = (y, (x_L \diamond y) \diamond x_L)$, because in that case $x_L \diamond x_R = y$. However, x_R cannot be of that form, since $(x_L, (y, (x_L \diamond y) \diamond x_L)) \notin \tilde{M}$. □

Let $\mathcal{M}_{X,677}$ denote the magma $(\tilde{M}, \diamond)$. We are finally ready to prove the central result of this section.

Theorem 4.2.12. *$\mathcal{M}_{X,677}$ is the free magma for Eq. (4.1) generated by X.*

Proof. We need to prove that $\mathcal{M}_{X,677}$ obeys Definition 4.2.10. We already know that $\diamond$ obeys Eq. (4.1) on $\tilde{M}$, since from Corollary 4.2.9 we know that $\diamond$ obeys 677 on M_X, and $\tilde{M} \subseteq M_X$. We can also define the map $i_{X,677} : X \to \tilde{M}$ as the inclusion map, since $X \subseteq \tilde{M}$. Now we just need to show that every function $f : X \to M$ into a 677 magma M can be extended to a unique homomorphism $\varphi_f : \tilde{M} \to M$ (since $\varphi_f \circ i_{X,677} = \varphi_f|_X$). Uniqueness is clear since $\tilde{M}$ is generated by X through $\diamond$, and φ_f is a homomorphism.

For existence, we define φ_f by first extending f to the unique homomorphism from M_X (with the pairing map) to M, and then restricting to $\tilde{M}$.

To verify the homomorphism property $\varphi_f(x \diamond y) = \varphi_f(x) \diamond \varphi_f(y)$, we are already done when $x \diamond y = (x, y)$, since φ_f is already an homomorphism with respect to the pairing map. The only remaining case is when $x \diamond y = y_L$ and $x < y = (y_L, (x \diamond y_L) \diamond x)$. We proceed by induction. The base case is given when y is a letter of X, in which case the claim is trivial, since it is not possible that $x < y$. Suppose by induction that for all $y' < y$, we have $\forall x' \in \tilde{M} : \varphi_f(x') \diamond \varphi_f(y') = \varphi_f(x' \diamond y')$. Therefore, knowing that $y = y_L \diamond y_R$, we obtain that:

$$\varphi_f(y) = \varphi_f(y_L) \diamond \varphi_f(y_R) = \varphi_f(y_L) \diamond (\varphi_f(x \diamond y_L) \diamond \varphi_f(x)) =$$
$$\varphi_f(y_L) \diamond ((\varphi_f(x) \diamond \varphi_f(y_L)) \diamond \varphi_f(x)),$$

where in the second equality we use the inductive hypotesis to prove that $\varphi_f(y_R) = \varphi_f(x \diamond y_L) \diamond \varphi_f(x)$, since $x < y$. The claim follows since M obeys 677. □

4.3 Equation 1516 does not imply Equation 255

In this final section, we will analyze the refutation $1516 \not\vdash 255$. In the first part, we will prove this non-implication by using various methods, specifically the greedy construction, to build a magma that obeys 1516 but not 255. In the second part, we will analyze its formalization in Lean, describing the most interesting parts of the code. The first part of the proof was formalized by Bernhard Reinke by adapting some code from the proof of another non-implication ($E1516 \not\vdash E1489$), while I contributed to formalizing the rest of the proof together with Lorenzo Luccioli and Pietro Monticone.

Initially, we tried to complete the formalization following the blueprint proof, but it was only thanks to the implementation in Lean that we noticed some issues in the proof that had initially escaped our attention. For this reason, we first modified the "on paper" proof, addressing the various problems we had encountered, and then completed the formalization.

An informal description of the proof can be found in the blueprint[3], while the code for the formalization, consisting of approximately 2000 lines and developed over about a month of continuous work, is available on GitHub[4].

4.3.1 Blueprint proof

Let us begin by presenting Equation 1516:

$$x = (y \diamond y) \diamond (x \diamond (x \diamond y)), \tag{4.9}$$

and Equation 255:

$$x = ((x \diamond x) \diamond x) \diamond x. \tag{4.10}$$

Using the notation $S_x := x \diamond x,\ L_y x := y \diamond x,\ R_y x := y \diamond x$, as done previously, we can rewrite Eq. (4.9) as:

$$x = L_{Sy} L_x L_x y. \tag{4.11}$$

Since we not deal with the formalization of the first part of the proof, we will avoid going into the details, but we will outiline all the most important steps.

Using a greedy construction, we begin by building a translation-invariant model with carrier $\mathbb{Z}$ that obeys 1516. We briefly introduce the definition of translation-invariant magmas.

Definition 4.3.1 (Translation-invariant magma)**.** A *translation-invariant magma* is a magma whose carrier G is an abelian group $G = (G, +)$, and whose magma operation takes the form

$$y \diamond x = y + f(x - y),$$

for some function $f : G \to G$.

[3]see `https://teorth.github.io/equational_theories/blueprint/1516-chapter.html`.

[4]see `https://github.com/teorth/equational_theories/blob/main/equational_theories/ManuallyProved/Equation1516.lean`.

Thus, taking $\mathbb{Z}$ as the carrier, we define:

$$x \diamond y := x + f(y - x), \tag{4.12}$$

for some function $f : \mathbb{Z} \to \mathbb{Z}$, with $f(0) = 0$. This ensures that $Sx = x$. Assuming we take $y = x + h$, we then have $L_x y = x + f(h)$, $L_y^2 x = x + f^2(h)$, and $L_y L_x^2 y = y + f(f^2(h) - h)$. Therefore, in this case, Eq. (4.11) becomes:

$$f(f^2(h) - h) = -h. \tag{4.13}$$

Let $E = \{(h, f(h))\}$ be the graph of f. We have that if $(a, b), (b, c) \in E$, then $(c - a, -a) \in E$, since:

$$f(c - a) = f(f(b) - a) = f(f^2(a) - a) \overset{4.13}{=} -a.$$

This helps to explain the next definition.

Definition 4.3.2 (1516 seed)**.** A *1516 seed* is a finite collection E of pairs (a, b), with $(a, b) \in \mathbb{Z}$, obeying the following statements:

1. E is finite;
2. $(0, 0) \in E$;
3. if $(a, b) \in E$ and $a \neq 0$, then $b \neq 0, -a$;
4. if $(a, b), (a, b') \in E$, then $b = b'$;
5. if $(a, b), (b, c) \in E$, then $(c - a, -a) \in E$;
6. If $(b, a), (b', a), (-b, d), (-b', d') \in E$ with $b \neq b'$, then $b + d \neq d', b' + d'$.

An extension of a 1516 seed E is a 1516 seed E' that contains E.

Now we provide, but do not prove, two results that will be helpful later.

Lemma 4.3.3 (1516 extension)**.** *Let E be a* 1516 *seed, and let $a_0 \in \mathbb{Z}$. Then there exists an extension E' of E that contains (a_0, c_0), for some $c_0 \in \mathbb{Z}$.*

Lemma 4.3.4 (1516 extension variant)**.** *Let E be a* 1516 *seed, and let $h \in \mathbb{Z}$ be non-zero. Then there exists an extension E' of E that contains $(a_i, a_i + h)$, for $i = 1, 2, 3, 4$, for some distinct a_1, a_2, a_3, a_4.*

Now, let us see how to construct a 1516 magma, using the greedy construction starting from a 1516 seed.

Theorem 4.3.5 (Base Magma)**.** *There exists a* 1516 *magma $\mathcal{M}$ with carrier $\mathbb{Z}$ with the properties that:*

i) $Sa = a$, $\forall a \in \mathbb{Z}$;

ii) For any distinct $a, b \in \mathbb{Z}$, there exist at least four solutions c for the equation $R_a c = b$;

iii) For each $a \in \mathbb{Z}$, there exist at least two $b \neq a$ such that $L_a R_a b = b$.

Proof. We start with $E_0 = \{(0,0), (-1,2), (3,1), (-10,20), (30,10)\}$, that is trivially a 1516 seed. Thanks to Lemma 4.3.3 and Lemma 4.3.4, we can iteratively extend E_0 using the greedy algorithm until we obtain a graph $\{(a, f(a)) : a \in \mathbb{Z}\}$ of a function $f : \mathbb{Z} \to \mathbb{Z}$ such that:

- $f(0) = 0,\ f(-1) = 2,\ f(3) = 1,\ f(-10) = 20,\ f(30) = 10$;
- if $f(a) = b$ and $f(b) = c$, then $f(c - a) = -a$;
- for every non-zero h, there are distinct $a_{h,1}, a_{h,2}, a_{h,3}, a_{h,4}$ with $f(a_{h,i}) = a_{h,i} + h$, for $i = 1, 2, 3, 4$;
- Eq. (4.13) holds;
- If we define the magma operation $\diamond$ by Eq. (4.12), we obtain Eq. (4.9). Therefore $\mathcal{M}$ is a 1516 magma.

Since $f(0) = 0$, we have i). Given distinct $a, b \in \mathbb{Z}$, we can rewrite $b = a + h$, for some non-zero h. There exist at least four values d_h such that $f(d_h) = d_h + h$. So we see that:

$$R_a(a - d_h) = a - d_h + f(d_h) = a - d_h + d_h + h = a + h = b,$$

from which we conclude ii).

Finally, fixed $a \in \mathbb{Z}$, we have $R_a(a+1) = a+1+f(a-a-1) = a+3$, so $L_aR_a(a+1) = a+f(a+3-a) = a+1$, and in the same way $L_aR_a(a+10) = a+10$, giving iii). □

Let us denote by A the free group over $\mathbb{N}$. Note that a magma with the same properties as in Theorem 4.3.5 but having carrier A can be constructed similarly. The same thing is true for the following results. This specification is due to the fact that in the formalization in Lean we used A instead of $\mathbb{Z}$, in order to make use of the pre-existing code.

Now we need to extend the magma to build a more complex 1516 magma, with a new carrier $G = \mathbb{Z} \uplus G'$, where:

$$G' := \{(a, c, n) \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{N} : a \neq c\}. \tag{4.14}$$

The elements of $\mathbb{Z}$ will be the squares in the final magma (from before, in fact, we have that $Sa = a,\ \forall a \in \mathbb{Z}$), while the elements in G' will be the non-squares. The most important coordinate of an element $(a, c, n) \in G'$ is a, since we will extend the squaring map S by defining $S(a, c, n) := a$; the other two components are technical; in particular, n will be used to ensure a certain infinite surjective property.

The 1516 magma constructed in Theorem 4.3.5 will be the restriction of G to $\mathbb{Z}$. Thus, for $a, b \in \mathbb{Z}$, $a \diamond b$ is already defined in G, but the rest of the multiplication table is currently undefined.

First, we need to provide some auxiliary results on the magma structure over $\mathbb{Z}$ that will be useful in the next steps.

Lemma 4.3.6. *Let $a, c \in \mathbb{Z}$, then $L_ca = a$ if and only if $c = a$.*

Proof. If $c = a$, we have that $L_aa = Sa = a$. On the other hand, if $L_ca = a$, from Eq. (4.11):

$$c = L_{Sa}L_cL_ca = L_{Sa}L_ca = L_{Sa}a = Sa = a. \qquad \square$$

Lemma 4.3.7. *For every $a \in \mathbb{Z}$, the function $L_a : \mathbb{Z} \to \mathbb{Z}$ is surjective.*

Proof. Since Eq. (4.11) holds, i.e., $L_{Sa}L_bL_ba = b$, we have that $L_{Sa} = L_a$ is surjective. □

As already mentioned, we extend the squaring map S to the entire set G by defining $S(a, c, n) := a$, so now S is a map from G to $\mathbb{Z}$. Now, for all $b \in \mathbb{Z}$, we need to extend the map $L_b : \mathbb{Z} \to \mathbb{Z}$ to a map from G to G, and also introduce additional maps $L_x : G \to G$ for $x \in G'$, obeying the following axioms:

Axiom A: for any $x \in G'$, $L_x x = Sx$;

Axiom B: for any $x \in G'$ and $b \in \mathbb{Z}$, $L_{Sx}L_bL_bx = b$;

Axiom C: for any $x \in G$ and $y \in G'$, $L_{Sx}L_yL_yx = y$.

We begin by proving the following lemma.

Lemma 4.3.8 (Useful elements)**.** *There exist a map $c : G' \times \mathbb{Z} \to \mathbb{Z}$ such that, for all $y \in G'$, we have:*

- $\forall b \in \mathbb{Z}$, $c_{y,b} \neq a, b, c$ *and* $L_aL_{c_{y,b}}b = c_{y,b}$;
- *the map $b \mapsto c_{y,b}$ is injective.*

Proof. Let us fix $y = (a, c, n) \in G'$ and $b \in \mathbb{Z}$. If $b = a$, from Item (iii), there exist at least two $c' \neq a$ such that $c' = L_aR_ac' = L_aL_{c'}a$. Thus, at least one is different from c, and we can take $c_{y,a} = c'$. Let us now consider the case where $b \neq a$. By Item (ii), there exist at least four c' such that $R_ac' = b$: we choose the value $\neq a, b, c$ and set $c_{y,b} = c'$. Hence, we have that:

$$L_aL_{c_{y,b}}b = L_aL_{c_{y,b}}R_ac_{y,b} = L_aL_{c_{y,b}}L_{c_{y,b}}a = c_{y,b},$$

where the last equality is given by Eq. (4.11).

Now we want to prove that the map $b \mapsto c_{y,b}$ is injective. Let $b \neq b'$ be two elements of $\mathbb{Z}$ such that $c_{y,b} = c_{y,b'}$. If $b = a$ or $b' = a$, then $c_{y,b} \neq c_{y,b'}$ by construction. Otherwise, $b = R_ac_{y,b} = R_ac_{y,b'} = b'$. □

Now we can proceed to extend all the operation to the carrier $\mathbb{Z} \times G$, with a greedy construction. Defining the theory Γ for the algorithm is equivalent to defining properties for a partial definition of $L_{c'} : G \to G$.

We define a *partial solution* as a partial assignment of $L_{c'}y \in G$ for $c' \in \mathbb{Z}$ and $y \in G$.

Definition 4.3.9 (Partial solution)**.** We say that a partial assignment for the functions $L_{c'} : G \to G$ for some $c' \in \mathbb{Z}$ is a **partial solution** if it satisfies the following properties, for any $c' \in \mathbb{Z}$ and $y \in G$:

a. if $y \in \mathbb{Z}$, then $L_{c'}y$ agrees with the operation in Theorem 4.3.5;

b. if $y = (a, c, 0) \in G'$, then $L_a y = a$;

c. if $y = (a, c, n) \in G'$ and $n \neq 0$, then $L_a y = (a, c, 0)$;

d. If $y = (a, c, n) \in G'$ and $b \in \mathbb{Z}$, then $L_{c_{y,b}} y = b$;

e. For any a, c, $L_c(a, c, n)$ is only defined for finitely many n;

f. If $y = (a, c, n) \in G'$ and $c' \in \mathbb{Z}$ are such that $L_{c'}y$ is defined, then $L_{c'}L_{c'}y$ and $L_a L_{c'} L_{c'} y$ are defined, with $L_a L_{c'} L_{c'} y$ equal to c';

g. If $y = (a, c, n) \in G'$ and $c' \in \mathbb{Z}$ are such that $c' \neq a$ and $L_{c'}y$ is defined, then $L_{c'}y \neq c'$.

This rule set was designed gradually to solve different problems that arose during the construction. In particular, the first version of the rule set that was proposed in the informal proof did not include Item b., Item c., and Item d., and only had a weaker version of Item f..

Moreover, the version of the informal proof that was in the blueprint was not organized in this modular fashion, and contained several typos as well as some obscure passages that turned out to be incorrect. The feedback from the initial formalization attempt was crucial to identify and correct these issues, prompting the original author, Terence Tao, to revise the informal

proof and provide a more detailed and formalization-friendly explanation of the construction.

This is an example of how the formalization process can help to improve the quality of the informal proof, and how the collaboration between the original designer of the proof and the people working on the Lean code can be beneficial for both parties. In fact, while some mistakes could have been spotted even without the formalization, a few subtle issues may have gone unnoticed, and became apparent only when trying to fill in some details of the code.

Let us now prove the existence of a partial solution.

Lemma 4.3.10 (Existence of a partial solution)**.** *A partial solution exists.*

Proof. We define $L_{c'}y$ for $y = (a, c, n) \in G'$ as follows:

i) if $c' = a$ and $n = 0$, then $L_{c'}y := a$;

ii) if $c' = a$ and $n \neq 0$, then $L_{c'}y := (a, c, 0)$;

iii) if $c' = c_{y,b}$ for some b, then $L_{c'}y := b$;

iv) in all other cases, $L_{c'}y$ is undefined.

We also defined $L_{c'}y$ for $y \in \mathbb{Z}$ using Theorem 4.3.5. From Lemma 4.3.8 we see that this is a well-defined operation, with $L_c y$ undefined for any $y = (a, c, n) \in G'$. So properties from Item a. to Item e. and Item g. are clear from construction. In each of the three cases Item (i), Item (ii), Item (iii), it is clear from construction that Item e. is satisfied. □

Now we want to extend the partial solution so that it is globally defined, and that (Axiom A), (Axiom B), and (Axiom C) are satisfied. To do this, we need to divide the extension into two steps, as we require the functions L_a to verify an infinite surjectivity property. The first extension is natural from the greedy construction, by defining the operation on a new pair where it was not previously defined.

Lemma 4.3.11 (First extension)**.** *Let $c' \in \mathbb{Z}$ and $y = (a, c, n) \in G'$. Suppose we have a partial solution for which $L_{c'}y$ is currently undefined, then one can extend this partial solution in such a manner that $L_{c'}y$ is now defined.*

Proof. From Lemma 4.3.7, we know that $L_a : \mathbb{Z} \to \mathbb{Z}$ and $L_{c'} : \mathbb{Z} \to \mathbb{Z}$ are surjective, so we can find $b \in \mathbb{Z}$ such that:

$$L_a L_{c'} b = c'. \tag{4.15}$$

If $b = c'$, then $L_{c'}c' = c'$. Therefore, Eq. (4.15) becomes $L_a c' = c'$, and from Eq. (4.9), we have $a = c'$. But this would lead to a contradiction, since $L_a y$ is already defined in Item (i). Therefore, $b \neq c'$, and we can define $L_{c'}y := b$, ensuring that Item g. is satisfied. □

Now we need to introduce a second extension step. At each step of the greedy construction, we take a pair $(c', y) \in \mathbb{Z} \times G'$ and a natural number m, and we require not only that $L_{c'}y$ is defined, but also that the equation $L_{c'}z = y$ has at least m distinct solutions $z \in G'$. This will be useful to show that the final $L_{c'}$ will have infinitely many solutions z to this equation. We will now show how, given a fixed $y \in G'$, if $L_{c'}z$ is still undefined, it is possible to set it equal to y.

Lemma 4.3.12 (Second extension)**.** *Suppose we have a partial solution, and let $c' \in \mathbb{Z}$ and $y = (a, c, n) \in G$, then, unless $c' = a$ and $n = 0$, exists $z \in G$ such that $L_{c'}z$ is currently undefined, but it is possible to extend the partial solution so that $L_{c'}z = y$.*

Remark 4.3.13. We exclude the case $c' = a$ and $n = 0$, because as we have defined the partial solution, we know that the set $\{x \in G' : L_a x = y\}$ is infinite, as it contains $(a, c, n), \forall n \in \mathbb{N}$.

Proof. We divide the proof into several cases:

(Case 1): $L_{c'}y = w$ for some $w \in G'$. From Lemma 4.3.8 we have that $c_{w,c'} \neq c'$, and $L_{c_{w,c'}}w = c'$. From Item e. we know that, given

$z = (c_{w,c'}, c', n)$, $L_{c'}z$ is only defined for finitely many n, therefore we can find n' such that $L_{c'}(c_{w,c'}, c', n')$ is undefined. Thus, we set $z = (c_{w,c'}, c', n')$ and extend the partial solution by defining $L_{c'}z := y$. So that we have $L_{c_{w,c'}} L_{c'} L_{c'} = c'$.

(Case 2): $c' \neq a$ and $L_{c'}y = b$ for some $b \in \mathbb{Z}$. From Item g. we have $c' \neq b$. By Theorem 4.3.5 we can find a' such that $L_{a'}b = R_b a' = c'$, and $a' \neq c'$. By Item e., we can find $z = (a', c', n')$ such that $L_{c'}z$ is currently undefined. Therefore we define $L_{c'}z := y$, and then we have $L_{a'}L_{c'}L_{c'}z = c'$.

(Case 3): $c' \neq a$ and $L_{c'}y$ is currently undefined. Using the first extension, we set $L_{c'}y$ equal to some $b \in \mathbb{Z}$, then we apply the second case.

(Case 4): $c' = a$. Since we are excluding the case $n = 0$, by Item c., we fall into the first case. □

We exclude the case $c' = a$ and $n = 0$, because as we have defined the partial solution, we know that the set $\{x \in G' : L_a x = y\}$ is infinite, as it contains (a, c, n), for any $n \in \mathbb{N}$. Instead, in the other cases, we add a finite number m of solutions z at each step of the greedy construction.

This distinction was not present in the first version of the informal proof, which remained a bit ambiguous on this point. During the first attempt to formalize the proof, we faced this issue and came up with a solution that tried to avoid modifying the structure of the code, and in particular we wanted to avoid the addition of an external parameter to the greedy extension step. The crucial point is that initially we had required the finiteness of the partial solution, and therefore we needed to control the number of new solutions that we were adding at each step and for how many pairs (c', y) we were adding them, but at the same time we needed a value that we could increase arbitrarily to ensure the infinite surjectivity of the final operation. We found that the cardinality of the set $\{a \mid L_a x = y \text{ for some } x \text{ and } y\}$ was a good candidate for this purpose, while at the same time being a quantity that only depended on data that was already present in the partial solution.

We implemented almost the whole proof with this approach, but near the end another more profound issue arose that forced us to modify the informal proof and prompted the original author to rewrite the blueprint in the current form. Since we needed to heavily modify the code to adapt it to the new proof, we decided to try again the previously discarded idea of adding an external parameter, and we found that this was a much cleaner and more straightforward solution; in fact, the idea of using the cardinality of the set above was a bit convoluted and added a lot of unnecessary complexity to the code that made it longer to write and harder to understand.

At this point, after explaining the two steps, we show how by running the greedy algorithm we manage to obtain an extension $L_b : G \to G$ that satisfies Axiom B.

Proposition 4.3.14 (Obtaining Axiom B)**.** *There exists a way to extend $L_b : \mathbb{Z} \to \mathbb{Z}$ to $L_b : G \to G$ in such a way that Axiom B holds, furthermore for each $b \in \mathbb{Z}$ and $x \in G'$, the set $\{y \in G' : L_b y = x\}$ is infinite. Also, we can ensure that $L_b x \neq x$, for any $b \in \mathbb{Z}$ and $x \in G'$.*

Proof. By iterating Lemma 4.3.11 and Lemma 4.3.12 in alternation, we can find an extension of the partial solution such that, for any $y = (a, c, n) \in G'$, $c' \in \mathbb{Z}$ and $k \in \mathbb{N}$, $L_{c'} y$ is defined, and such that $L_{c'} z = y$ has at least k distinct solutions $z \in G'$: it is enough to extend the partial solution several times using Lemma 4.3.12, in order to obtain the necessary number of solutions to the equation. So we can find an increasing chain of partial solutions with these property, and taking the limit of this chain we can find a fully defined operation $L_{c'} y$ for $c' \in \mathbb{Z}$ and $y \in G$. This operation obeys all the properties of Definition 4.3.9, but not Item e.. Thus, considering the increasing chain of extensions, the partial solutions will have an increasing number k of solutions to $L_{c'} z = y$, so since the operation $L_{c'}$ is defined as the limit of this chain, it will have more solutions than any $k \in \mathbb{N}$, and therefore, it will be infinite. For this reason, for each $b \in \mathbb{Z}$ and $x \in G'$, the set $\{y \in G' : L_b y = x\}$ is infinite. □

At this point, the operation $\diamond$ is well defined when we have an element from $\mathbb{Z}$ on the left and one from G on the right. The only thing left to define is $L_x : G \to G$ for $x \in G'$. In this case, the greedy construction is simpler to implement, and it is not necessary to describe the extensions outside the proof as we did previously.

Proposition 4.3.15 (Obtaining Axiom A, C)**.** *There exists maps $L_x : G \to G$ for each "non-square" $x \in G'$, such that Axiom A, Axiom C hold.*

Proof. We fix $x \in G'$. Our task is to find a function L_x such that:

$$L_x x = Sx, \tag{4.16}$$

and

$$L_{Sy} L_x L_x y = x, \tag{4.17}$$

for all $y \in G$. We construct a seed from which to start the greedy construction, in order to define a partial function L_x such that:

a. it is defined on a finite number of values;

b. it is injective;

c. it satisfies Eq. (4.16);

d. it satisfies Eq. (4.17), in the case where $L_x L_x y$ is defined.

If there exists a $y \in G$ such that $L_x y$ is not yet defined, then, because L_x is injective, there will be at most one z such that $L_x z = y$. If such a z exists, we define $L_x y := w'$, where $w' \in \{w : L_{S_z} w = x\}$ is different from y and is not in the domain or range of L_x. We can do this because, from the infinite surjectivity property in Proposition 4.3.14, we know that the set $\{w : L_{S_z} w = x\}$ is infinite. We already know that with this new extension, conditions (a), (b), and (c) continue to be satisfied, and now we need to check (d). But in the case of y, we have $L_{Sy} L_x L_x y = L_{Sy} L_x w'$, but $L_x w'$ is not defined. On the other hand, in the case of z, we have:

$$L_{Sz} L_x \underbrace{L_x z}_{y} = L_{Sz} \underbrace{L_x y}_{w'} = L_{Sz} w' = x.$$

If, otherwise, no such z exists, we set $L_x y$ to be an arbitrary element of G, not already in the domain or range of L_x, and we can see that the seed properties continue to be satisfied.

Therefore, starting from a seed where only $L_x x := Sx$ is defined, and iteratively extending it following the criteria just outlined, we obtain the thesis. □

After these different greedy constructions, we have finally build a new magma $\mathcal{M} = (G, \diamond)$ that obeys Equation 1516. Now the only thing left to do is to prove that violates 255, also proving that $1516 \not\vdash 255$.

Theorem 4.3.16. *Let $\mathcal{M} = (G, \diamond)$ be the magma just build, then $\mathcal{M}$ does not obey Equation* 255.

Proof. We fix $x = (a, c, 0) \in G'$. From Axiom C we know that $L_x x = Sx = a$, and by Item b. we have $L_a x = a$. Putting everything in Eq. (4.10):

$$(\underbrace{(x \diamond x)}_{a} \diamond x) \diamond x = \underbrace{(a \diamond x)}_{a} \diamond x = a \diamond x = a \neq x.$$

Therefore $\mathcal{M}$ does not obey Equation 255. □

4.3.2 Formalization of the proof

Now that we have finished describing the informal proof provided in the blueprint, explaining how it was progressively modified to resolve issues that we encountered during the formalization, we will conclude by providing some details about its formalization.

We will now examine the formalization of Lemma 4.3.8 to highlight the challenges encountered in formalizing a proof that, on paper, seems rather straightforward.

```
1   lemma exists_useful_c (y : G') : ∃ c : A → A, c.Injective ∧
2   ∀ b, y.1.1 ◇ ((c b) ◇ b) = c b ∧ c b ≠ b ∧ c b ≠ y.1.1 ∧ c b ≠
    y.1.2.1 := by
3  rcases base2' y.1.1 y.1.2.1 with ⟨c₁, hc₁a, hc₁c, hc₁⟩
```

```
4  have c_aux {b : A} (h : y.1.1 ≠ b) : ∃ c, c ◇ y.1.1 = b ∧ c ≠ c₁ ∧
     c ≠ b ∧ c ≠ y.1.2.1 :=
5    base1' h c₁ b y.1.2.1
6  let c := fun b : A ↦ if h : y.1.1 = b then c₁ else (c_aux h).
     choose
7  refine ⟨c, fun b₁ b₂ ↦ ?_, fun b ↦ ?_⟩
8  · unfold c
9    rcases ne_or_eq y.1.1 b₁ with hx | ha <;> rcases ne_or_eq y.1.1 b₂
      with hy | ha'
10   · rw [dif_neg hx, dif_neg hy]
11     intro hind
12     have prop : (c_aux hx).choose ◇ y.1.1 = (c_aux hy).choose ◇ y
   .1.1 := by rw [hind]
13     have h_aux : (c_aux hx).choose ◇ y.1.1 = b₁ := (c_aux hx).
   choose_spec.1
14     have h_aux2 : (c_aux hy).choose ◇ y.1.1 = b₂ := (c_aux hy).
   choose_spec.1
15     rw [h_aux, h_aux2] at prop
16     exact prop
17   · rw [dif_neg hx, dif_pos ha']
18     exact fun h ↦ ((c_aux hx).choose_spec.2.1 h).elim
19   · rw [dif_pos ha, dif_neg hy]
20     exact fun h ↦ ((c_aux hy).choose_spec.2.1 h.symm).elim
21   · exact fun h ↦ ha ▷ ha'
22 · unfold c
23   rcases ne_or_eq y.1.1 b with h1 | h2
24   · rw [dif_neg h1]
25     refine ⟨?_, ⟨(c_aux h1).choose_spec.2.2.1, ?_, (c_aux h1).
   choose_spec.2.2.2⟩⟩
26     · nth_rw 1 [← A_idempotent y.1.1]
27       nth_rw 4 [← (c_aux h1).choose_spec.1]
28       exact (A_satisfies_Equation1516 _ _).symm
29     · by_contra h
30       have := A_idempotent _ ▷ h ▷ (c_aux h1).choose_spec.1
31       exact h1 this
32   · simp_rw [dif_pos h2, ← h2, hc₁]
33     exact ⟨trivial, hc₁a, hc₁a, hc₁c⟩
```

As can be clearly seen, the code appears to be much more complex than initially expected. Let us explain it. In the first few lines, we have the statement of the theorem: as can be observed, the first step is to fix $y \in G'$, and then the existence of a map $c : A \to A$ (which, after fixing y, would actually be the map $b \mapsto c_{y,b}$) is required, satisfying the two conditions stated in Lemma 4.3.8. After defining the map c in:

```
let c := fun b : A ↦ if h : y.1.1 = b then c₁ else (c_aux h).choose
```

by utilizing previously defined auxiliary functions, we use:

```
refine ⟨c, fun b₁ b₂ ↦ ?_, fun b ↦ ?_⟩
```

which is the command to transform the goal and split it into two parts, proving separately that the map c just defined satisfies both conditions in Lemma 4.3.8. From lines 8 to 21, we prove injectivity, and as can be seen, it is necessary to divide the proof into several cases. Specifically, we require that y is either different from or equal to b_1, and each of these cases is further divided into two subcases: $y = b_2$ or $y \neq b_2$. A similar procedure is applied starting from line 22 onward, where we proceed to prove that for each $b \in \mathbb{Z}$, we have $c_{y,b} \neq a, b, c$, and that $L_a L_{c_{y,b}} b = c_{y,b}$. This code is an example of how a proof can be formalized step by step to ensure its correctness. The proof is particularly detailed and relies on techniques of rewriting and case splitting to handle the various possibilities that arise during the construction of the map c. To make it easier to utilize the results proven in the theorem, the properties are divided into several lemmas:

```
1 noncomputable abbrev useful_c (y : G') : A → A := (exists_useful_c y)
      .choose
2
3 lemma useful_c_injective (y : G') : (useful_c y).Injective := (
      exists_useful_c y).choose_spec.1
4
5 lemma useful_c_spec (y : G') (b : A) : y.1.1 ◇ ((useful_c y b) ◇ b) =
       useful_c y b :=
6   (exists_useful_c y).choose_spec.2 b |>.1
7
```

```
lemma useful_c_ne_b (y : G') (b : A) : useful_c y b ≠ b :=
  (exists_useful_c y).choose_spec.2 b |>.2.1

lemma useful_c_ne_y₁ (y : G') (b : A) : useful_c y b ≠ y.1.1 :=
  (exists_useful_c y).choose_spec.2 b |>.2.2.1

lemma useful_c_ne_y₂ (y : G') (b : A) : useful_c y b ≠ y.1.2.1 :=
  (exists_useful_c y).choose_spec.2 b |>.2.2.2
```

We give a name to the map c constructed in the theorem, denoting it by `useful_c`, where it is necessary to always fix y, since we are working with the map $b \mapsto c_{y,b}$. When there exists a lemma of the form $\exists x : P(x)$, in Lean, to select the element x, we use the function `(name of the lemma).choose`, while the function `(name of the lemma).choose_spec.1` provides the proof that the x we have just selected satisfies $P(x)$. This is how all the lemmas in which we have divided the properties of the map c were constructed.

Chapter 5

Conclusions

In this thesis, we explored the "Equational Theories" project launched by Terence Tao, an innovative initiative that combines collaboration between professional and amateur mathematicians with the use of artificial intelligence tools and proof assistant languages, such as Lean. The project represents a new way of conducting mathematical research, where formalization and proof verification play a central role, and where collaboration is extended beyond traditional small groups to include a wider community of contributors, including those without advanced mathematical training.

One of the key aspects of the project was the use of **Lean** as a formalization tool. Lean not only ensured the correctness of the results but also played a crucial role in identifying and correcting errors in the "on paper" proofs. During the formalization process, particularly in the proof of the non-implication $1516 \not\vdash 255$, we encountered several issues in the informal proof that had initially gone unnoticed. These issues were only revealed when attempting to formalize the proof in Lean, prompting revisions to the original blueprint. This experience highlighted the importance of formalization in improving the quality and accuracy of mathematical proofs, as well as the value of collaboration between the original proof designers and those working on the formalization.

A central technique used throughout the project was the **greedy algo-**

rithm construction, which proved to be a powerful method for building algebraic structures, particularly infinite magmas, that satisfy specific equations. This iterative algorithm allowed us to extend partially defined magma operations to fully defined ones while ensuring that the resulting structures obeyed the desired equational laws. The greedy construction was instrumental in constructing counterexamples, such as the one used to prove that equation 1516 does not imply equation 255. This technique demonstrated its versatility and effectiveness in handling complex algebraic problems, especially when combined with other tools like Mace4 and Prover9.

Another notable outcome of the project was the emergence of magma cohomology, a mathematical structure that arose naturally during the exploration of certain proofs. Magma cohomology can be seen as a generalization of group cohomology, and it was discovered while working on strategies to extend magmas and resolve finite implications. The construction of magma cohomology involves extending a magma G by taking the set $G \times M$, where M is a ring or an abelian group, and defining an operation that incorporates a "cocycle" satisfying specific conditions. The relationship between magma cohomology and group cohomology highlights the deep connections between different areas of mathematics and how new structures can emerge from the study of specific problems.

In conclusion, the Equational Theories project represents a significant step forward in the way mathematical research is conducted, since it introduced a new way of collaborating in mathematics. Unlike traditional research projects, which are typically conducted by small, tightly-knit groups of experts, the Equational Theories project was designed to be open and accessible to a broader audience. This was made possible by the use of Lean, which allowed contributors with varying levels of experience to participate. The project was organized through a central GitHub repository, where contributors could claim tasks, upload formalized proofs, and track the progress of the project. Terence Tao's daily log and the Zulip discussions provided a historical perspective and facilitated communication among participants. This work has shown how

formal mathematics, combined with innovative collaboration and advanced computational tools, can lead to new insights and discoveries. The hope is that projects like this will inspire further research and collaboration, pushing the boundaries of mathematics and opening up new avenues for exploration.

Bibliography

[Bro12] Kenneth S Brown. *Cohomology of groups*. Vol. 87. Springer Science & Business Media, 2012.

[Clu+24] Joshua Clune et al. "Duper: A Proof-Producing Superposition Theorem Prover for Dependent Type Theory". In: *15th International Conference on Interactive Theorem Proving (ITP 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2024, pp. 10–1.

[Cor+22] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.

[Her78] I. N. Herstein. *Algebra*. English. Physik Verlag, Weinheim, 1978, pp. xii+390. ISBN: 3-87664-035-0.

[LF23] Jannis Limperg and Asta Halkjær From. "Aesop: White-box best-first proof search for Lean". In: *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2023, pp. 253–266.

[RG24] Marcus Rossel and Andrés Goens. "Bridging Syntax and Semantics of Lean Expressions in E-Graphs". In: *arXiv preprint arXiv:2405.10188* (2024).

[Rot95] Joseph J. Rotman. *An introduction to the theory of groups*. Fourth. Vol. 148. Graduate Texts in Mathematics. Springer-Verlag, New York, 1995, pp. xvi+513. ISBN: 0-387-94285-8. DOI: 10.1007/978-1-4612-4176-8. URL: https://doi.org/10.1007/978-1-4612-4176-8.