Formalisation of the Finite Simple Conway Groups in Lean

Mathematics Bachelor's Thesis

Erik van der Plas (6784852)

Supervised by dr. Johan Commelin

June 17, 2024

Utrecht University

CONTENTS

1 INTRODUCTION

In this thesis we study the Conway groups, a family of sporadic simple groups discovered by John Horton Conway in 1968. These groups are closely related to the Leech lattice, a remarkable lattice in 24-dimensional Euclidean space, discovered by John Leech, just the year before, in 1967.[1]We will demonstrate a construction of the Leech lattice, show how it can be used to construct the Conway groups and provide a proof of the simplicity of one of them, Co_1. Furthermore, we formalise part of this proof in the Lean theorem prover, a computer programming language for writing and verifying mathematical proofs. This is a tiny contribution to the much greater project of formalising the classification of finite simple groups in Lean.

The thesis is structured as follows. In Chapter 2, we introduce some fundamental group theory concepts, revising definitions and results taught during the Mathematics Bachelor's program at Utrecht University. In Chapter 3, we dive into the theory of permutation groups, which will provide a useful framework for studying the simplicity of various sporadic simple groups through Iwasawa's lemma, including the Conway groups. Chapter 4 provides an overview of the classification of finite simple groups (CFSG), which is a monumental result in group theory. In Chapter 5 we introduce the Golay code, in order to construct the Leech lattice in Chapter 6, and in turn the Conway groups in Chapter 7. Finally, in Chapter 8, we present our formalisation of the Conway groups in Lean, and discuss the challenges we encountered during this process. Lastly, Chapter 9 summarizes our findings and suggests directions for future research.

1 The German mathematician Ernst Witt claims to have discovered the Leech lattice in 1940, but he did not publish his results. To this day, it remains unclear to what extent he indeed knew of this special lattice in advance of Leech.

2 PREREQUISITES

We assume the reader to be familiar with the basic concepts of group theory, as taught at the University of Utrecht in the course Groepen & Ringen (WISB124), or equivalently, Part I of Dummit and Foote, 2003. The most important definitions and results for this thesis will also be reviewed in this chapter.

2.1 COSETS

Definition 1 (Left/right cosets)**.** For a subgroup H of a group G and $g \in G$, we define the left coset $gH = \{gh \mid h \in H\}$ and the right coset $Hg = \{hg \mid h \in H\}$. Furthermore, with $G : H$ we denote the set of all left (or right) cosets of H in G, and we call $|G : H|$ the index of H in G.

A simple example of cosets is given by the integers modulo some $n \in \mathbb{N}$. For the group of integers under addition, denoted by $\mathbb{Z}$, $n\mathbb{Z}$ is a subgroup of $\mathbb{Z}$ of all multiples of n. The (left) cosets of $n\mathbb{Z}$ in $\mathbb{Z}$ are given by $a + n\mathbb{Z}$ for $a = 0, 1, \dots, n-1$, corresponding with all $x \in \mathbb{Z}$ such that $x \equiv a \pmod{n}$. Note that in this example, the left and right cosets coincide, as the additive group is abelian.

Lemma 1. For a subgroup H of G, for all $g_1, g_2 \in G$ the left cosets g_1H and g_2H (and analogously, the right cosets Hg_1 and Hg_2) are either disjoint or identical.

Proof. Let H be a subgroup of G and $g_1, g_2 \in G$. Suppose that g_1H and g_2H are not disjoint. Consequently, there exist $h_1, h_2 \in H$ such that $g_1h_1 = g_2h_2$. We can rewrite as $g_2^{-1}g_1 = h_2h_1^{-1}$ and as H is a subgroup, and therefore closed under multiplication, we find that $g_2^{-1}g_1 \in H$. Let $c \in g_1H$ and $h' := g_1^{-1}c$ such that $c = g_1h' \in g_1H$. Clearly, $g_2^{-1}g_1h' \in H$ and therefore $c = g_2g_2^{-1}g_1h' \in g_2H$. As our choice of $c \in g_1H$ was arbitrary, we conclude $g_1H \subseteq g_2H$ and by symmetric argument also $g_2H \subseteq g_1H$, hence $g_1H = g_2H$. So indeed, cosets are either disjoint or identical. □

2.2 MAXIMAL SUBGROUPS

Definition 2 (Maximal subgroup)**.** We consider a subgroup H of G maximal if there is no subgroup K such that $H < K < G$.

For example, in the dihedral group $D_3 = \langle r, f \mid r^3 = f^2 = \mathbf{1}, rf = fr^{-1} \rangle$ of order 6, the subgroup $H := \langle r \rangle = \{\mathbf{1}, r, r^2\}$ is maximal, as there is no subgroup K such that $H < K < D_3$. In this case, this follows immediately from Lagrange's theorem, as the order of such subgroup K would have to divide the order of D_3, and the only such divisor strictly larger than $|H| = 3$ is 6 itself.

2.3 NORMAL SUBGROUPS

Definition 3 (Normal subgroup)**.** For a group G, a subgroup N is called normal if for all $g \in G$ we have that the coset $gN = Ng$. Equivalently, N is invariant under conjugation by the elements of G, i.e. $gng^{-1} \in N$ for all $n \in N$ and $g \in G$. We denote this by $N \trianglelefteq G$.

The same example from the previous section can be used to illustrate normal subgroups, as the subgroup $\langle r \rangle$ of the dihedral group D_3 is normal. We can easily check this by considering all conjugates, but it turns out there are many other ways to assess subgroup normality, which can be more tractable for larger groups.

2.4 SIMPLE GROUPS

Definition 4 (Simple group)**.** A group G is called simple if it has no nontrivial normal subgroups, i.e. G has no normal subgroups other than $\{\mathbf{1}\}$ and G itself.

A clear example of a family of simple groups, are the cyclic groups of prime order, denoted by $C_p = \langle g \mid g^p = \mathbf{1} \rangle$ for some prime p. Their simplicity follows from Lagrange's theorem, as the order of any subgroup of C_p must divide p, and the only such divisors are 1 and p. Consequently, C_p has no nontrivial (normal) subgroups.

Simple groups are the building blocks of group theory, as any finite group can be decomposed into simple groups, in a so-called composition series.

Definition 5 (Composition series)**.** A composition series of a group G is a sequence of subgroups

$$\{\mathbf{1}\} = G_0 \trianglelefteq G_1 \trianglelefteq \ldots \trianglelefteq G_{n-1} \trianglelefteq G_n = G,$$

such that each G_i is a normal subgroup of G_{i+1}, and G_{i+1}/G_i is simple for all i.

It is precisely for this reason that simple groups are of such importance in group theory, as we can study the structure of any finite group by studying the structure of its simple composition factors. The classification of such finite simple groups is one of the most important results in group theory, and is the subject of Chapter 4. First, we will continue with some more fundamental group theory concepts.

2.5 PERFECT GROUPS

Sometimes, it can be rather difficult to assess whether a group is simple or not. For the groups of interest in this thesis, the Conway groups, we will make use of Iwasawa's lemma to show that they are simple. For this lemma, we need the notion of commutator subgroups and perfect groups.

Definition 6 (Commutator)**.** For $g, h \in G$, we define the commutator by $[g, h] := g^{-1}h^{-1}gh$.

Definition 7 (Commutator subgroup)**.** The commutator subgroup of G, denoted by G', is the subgroup generated by all the commutators.

To illustrate the commutator subgroup, consider that for all abelian groups, the commutator subgroup is trivial ($\{\mathbf{1}\}$). This also clarifies the terminology, as for any group G and $g, h \in G$ the commutator $[g, h] = \mathbf{1}$ if and only if g and h *commute*. By construction, the commutator subgroup is also invariant under conjugation, thus a normal subgroup:

Lemma 2. For any group G, the commutator subgroup G' is a normal subgroup.

Proof. Let $g, h, c \in G$. Then, $[g, h] \in G'$ and

$$\begin{aligned} c^{-1}[g,h]c &= c^{-1}g^{-1}h^{-1}ghc \\ &= c^{-1}g^{-1}cc^{-1}h^{-1}cc^{-1}gcc^{-1}hc \\ &= (c^{-1}g^{-1}c)(c^{-1}h^{-1}c)(c^{-1}gc)(c^{-1}hc) \\ &= (c^{-1}gc)^{-1}(c^{-1}hc)^{-1}(c^{-1}gc)(c^{-1}hc) \\ &= [c^{-1}gc, c^{-1}hc] \in G'. \end{aligned}$$

As the choice for g, h and c was arbitrary, G' is invariant under conjugation by the elements of G, hence G' is normal in G. $\square$

In some sense, the commutator subgroup gives us a measure of how far a group is from being abelian. More formally:

Lemma 3. For any group G and normal subgroup $N \trianglelefteq G$, the quotient group G/N is abelian if and only if $G' \subseteq N$.

Proof. Let N be a normal subgroup of some group G.

Suppose that G/N is abelian. Then, for all $g, h \in G$ we have $(gN)(hN) = (hN)(gN)$. Consequently, $(gN)^{-1}(hN)^{-1}(gN)(hN) = N$, or equivalently $[g, h]N = N$, and thus $[g, h] \in N$.

Conversely, suppose that $[g, h] \in N$ for all $g, h \in G$. Then, $[g, h]N = N$, hence $(gN)^{-1}(hN)^{-1}(gN)(hN) = N$ and $(gN)(hN) = (hN)(gN)$. So, G/N is abelian. $\square$

Definition 8 (Perfect group)**.** The group G is called perfect if the commutator subgroup $G' = G$.

Lemma 4. Any non-abelian simple group is perfect.

Proof. Let G be a non-abelian simple group, such that it has no non-trivial normal subgroups. As the commutator subgroup G' is a normal subgroup by Lemma 2, we find that $G' = \{\mathbf{1}\}$ or $G' = G$. The former case implies that all elements of G commute, which contradicts the assumption that G is non-abelian. Hence, $G' = G$ and G is perfect. $\square$

Although it is rather trivial to show that non-abelian simple groups are perfect, the converse is not true in general. However, Iwasawa's lemma provides a partial converse of this statement, as the introduction of some extra constraints on a perfect group do imply simplicity.

PERMUTATION GROUPS 3

It follows from Cayley's theorem that all groups are isomorphic to a permutation group (i.e. a subgroup of a symmetric group). Hence, from now on, we can consider any group G as permuting the elements of some set Ω. Here we adopt the notation from Wilson, 2009: for all permutations $\pi, \rho \in G$ acting on some $a \in \Omega$ we denote $a^\pi = \pi(a)$ and $a^{\pi\rho} = \rho(\pi(a))$. Note that this notation inverts the order of multiplication as compared to function composition.

In this chapter, we will define some properties of permutation groups. We will use these properties in our statement of Iwasawa's lemma, which we will prove at the end of this chapter.

3.1 FAITHFULNESS

Definition 9 (Faithfulness)**.** The group action of G on Ω is called faithful if $a^\pi = a$ for all $a \in \Omega$ implies that $\pi = \mathbf{1}$.

Faithfulness follows naturally for permutation groups acting on the associated permuted set, such as $G := \mathrm{Sym}(\Omega)$ acting on Ω. But we can certainly consider group actions that are not faithful. For example $D_3 := \langle r, f \mid r^3 = f^2 = \mathbf{1}, rf = fr^{-1} \rangle$ acting on just the orientation of a triangle. Here, the orientation that some $r^i f^j \in D_3$ with $i, j \in \mathbb{Z}$ entails is precisely determined by the parity of j. In particular, not only the identity element fixes the orientation, but also r and r^2, and thus the action is not faithful.

3.2 STABILIZERS

Definition 10 (Stabilizer)**.** For a group G acting on Ω and some $a \in \Omega$ we call

$$G_a = \{\pi \in G \mid a^\pi = a\}$$

the stabilizer of a in G.

For example, we can view D_3 as a permutation group acting on the triangle's vertices. The group action corresponding with reflections over one of the vertices and the identity action both fix that vertex, and are collectively its stabilizer.

Lemma 5. For a group G acting on Ω and some $a \in \Omega$, the stabilizer G_a is a subgroup of G (that is, $G_a \leq G$).

Proof. Clearly, the identity element of G always fixes a, so G_a is non-empty. Furthermore, for all $\pi, \rho \in G_a$, we find that $\pi\rho^{-1} \in G_a$ as $a^{\pi\rho^{-1}} = a^{\rho^{-1}} = a$, where the latter equivalence holds as $a^\rho = a$. We can conclude by the subgroup criterion that indeed $G_a \leq G$. □

Consequently, we can also unambiguously call a stabilizer a *stabilizer subgroup*.

3.3 TRANSITIVITY

Definition 11 (Transitivity)**.** The group action of G on Ω is called transitive if for all $a, b \in \Omega$ there exists some $\pi \in G$ such that $a^\pi = b$.

An obvious example of a transitive group action is $\mathrm{Sym}(\Omega)$ acting on Ω, as it contains all possible permutations of Ω. In contrast, stabilizer subgroups act non-transitively by definition. We can derive an interesting consequence from transitivity regarding stabilizers:

Lemma 6. If G acts transitively on Ω, then for all $a \in \Omega$ there exists a natural bijection from Ω to the right cosets of G_a in G, given by $a^\pi \leftrightarrow G_a\pi$.

Proof. Assume G acts transitively on Ω and let $a \in \Omega$. Let $\phi : \Omega \to G : G_a$ be given by $a^\pi \mapsto G_a\pi$.

To prove that this map is well-defined, let $x \in \Omega$. By transitivity, we find there always exists some $\pi \in G$ such that $a^\pi = x$. However, this π is not necessarily unique. Suppose that $a^{\pi_1} = a^{\pi_2} := x$ for distinct $\pi_1, \pi_2 \in G$. Clearly, $\pi_1\pi_2^{-1} \in G_a$, as $a^{\pi_1\pi_2^{-1}} = x^{\pi_2^{-1}} = a$, so $\pi_1\pi_2^{-1}\pi_2 \in G_a\pi_2$ and clearly also $\pi_1\pi_2^{-1}\pi_2 = \pi_1\mathbf{1} = \mathbf{1}\pi_1 \in G_a\pi_1$. Cosets are either disjoint or identical by Lemma 1, so $G_a\pi_1 = G_a\pi_2$. Hence, ϕ is well-defined.

To prove bijectivity, we need to show that ϕ is both injective and surjective.

Suppose $a^{\pi_1} \neq a^{\pi_2}$ for some $\pi_1, \pi_2 \in G$. It follows that $\pi_1 \notin G_a\pi_2$, as otherwise there would be an element $\rho \in G_a$ such that $\pi_1 = \rho\pi_2$, which would imply $a^{\pi_1} = a^{\rho\pi_2} = a^{\pi_2}$. Clearly however, $\pi_1 =$

$\mathbf{1}\pi_1 \in G_a\pi_1$. Consequently, as cosets are either disjoint or identical by Lemma 1, we find that $G_a\pi_1 \neq G_a\pi_2$. This contrapositively proves injectivity of ϕ.

Now, let $G_a\pi$ be any right coset of G_a in G, for some $\pi \in G$. Clearly, $a^\pi \in \Omega$, so surjectivity follows naturally.

We can conclude that ϕ is indeed a well-defined bijection $\Omega \leftrightarrow G : G_a$. □

3.4 PRIMITIVITY

Definition 12 (Partition)**.** A partition of a set Ω is a set of mutually disjoint nonempty subsets of Ω whose union is Ω. In other words, it is a grouping of the elements of Ω into nonempty subsets, such that every element is contained in precisely one of these subsets.

Two examples of partitions on any set Ω are the trivial partitions: either into a single set, i.e. $\{\Omega\}$, or into $|\Omega|$ singleton sets, i.e. $\{\{x\} \mid x \in \Omega\}$. We can also consider how a permutation group G acting on the set Ω preserves partitions of Ω. In this context, we often call a partition a 'block system' and its elements 'blocks'. We say a partition or block system is preserved, if for all $a, b \in \Omega$ in the same block of the partition, a^π and b^π are also in the same block for all $\pi \in G$.

Definition 13 (Primitivity)**.** The group action of G on Ω is called primitive if it is transitive, and the only partitions it preserves are the trivial partitions.

Equivalently, if a group action is not primitive, there exists a nontrivial partition of Ω that is preserved by the group, which is often called a 'system of imprimitivity'. We can demonstrate maximality of stabilizer subgroups for primitive groups:

Lemma 7. If G acts primitively on Ω, the stabilizer G_a is a maximal subgroup of G for all $a \in \Omega$.

Proof. Suppose that G acts primitively on Ω and let $a \in \Omega$. Assume for the sake of contradiction that G_a is not maximal, so that we can find a subgroup H such that $G_a < H < G$. As G is primitive, it is transitive, hence by Lemma 6 we find the natural bijection $\phi : \Omega \leftrightarrow G : G_a$. We construct $\psi : G : G_a \to G : H$ by $G_a\pi \mapsto H\pi$ for all $\pi \in G$, and the composite function $\psi \circ \phi$ mapping the elements of Ω to the right cosets of H in G. As by assumption these cosets are strictly larger than the right cosets of G_a in G, we find that $\psi \circ \phi$ groups the elements of Ω in a nontrivial partition, which is naturally preserved by G. This contradicts the primitivity of G, so we can conclude that G_a is indeed a maximal subgroup for all $a \in \Omega$. □

3.5 IWASAWA'S LEMMA

Lemma 8 (Iwasawa)**.** Let G be a finite perfect group, acting faithfully and primitively on Ω. Suppose that some stabilizer G_a contains an abelian normal subgroup A (that is, $A \lhd G_a$) whose conjugates in G generate all of G. Then G is simple.

Proof. Suppose that G, Ω, G_a and A are as premised, and assume for the sake of contradiction that N is a nontrivial normal subgroup of G.

First we show there exists an $x \in \Omega$ such that $N \nleq G_x$. Assume on the contrary that N is contained in every stabilizer. So, for all $x \in \Omega$ and $\pi \in N$ we have $\pi \in G_x$, hence $x^\pi = x$. As G acts faithfully on Ω it is implied that $\pi = \mathbf{1}$. But that contradicts that N is nontrivial.

Consequently, we can choose a point stabilizer G_x such that $N \nleq G_x$, and therefore $G = NG_x$, since G_x is a maximal subgroup of G by Lemma 7. By transitivity, we can write $x = a^\pi$ for some $\pi \in G$, hence

$$G_x = G_{a^\pi} = \{\pi^{-1}\rho\pi \mid a^\rho = a\} = \pi^{-1}G_a\pi.$$

So, G_x and G_a are conjugate and therefore isomorphic. Hence, there also exists an abelian normal subgroup $X \lhd G_x$ where $X \cong A$.

Let $\pi \in G$ and write $\pi = \rho\sigma$ where $\rho \in G_x$ and $\sigma \in N$. We then find that

$$\pi^{-1}X\pi = \sigma^{-1}\rho^{-1}X\rho\sigma = \sigma^{-1}X\sigma$$

by normality of X in G_x. As by assumption the conjugates of X in G generate G, we find that $G = NX$. By the second isomorphism theorem we then find that $G/N \cong X/(X \cap N)$, which is abelian.

Consequently, $G' \subseteq N$ by Lemma 3 and $N < G$ by assumption, which contradicts that G is perfect, so G must be simple.

□

4 CLASSIFICATION OF FINITE SIMPLE GROUPS

Finite simple group are of great interest for group theorists, as they are the building blocks of group theory. As explained in Section 2.4, any finite group can be decomposed into simple groups, similar to how any integer can be decomposed into prime factors. A lot of effort has been put into classifying all finite simple groups, and the result is the Classification of Finite Simple Groups (CFSG), cited here from Wilson, 2009[1]:

Theorem 1 (CFSG)**.** Every finite simple group is isomorphic to one of the following groups:

1. a cyclic group C_p of prime order p,
2. an alternating group A_n for $n \geq 5$,
3. a classical group [...],
4. an exceptional group of Lie type [...], or the [derived] Tits group,
5. one of 26 sporadic simple groups:
   - the five Mathieu groups M_{11}, M_{12}, M_{22}, M_{23}, M_{24},
   - the seven Leech lattice groups Co_1, Co_2, Co_3, McL, HS, Suz, J_2,
   - the three Fischer groups Fi_{22}, Fi_{23}, Fi_{24}',
   - the five Monstrous groups $\mathbb{M}$, $\mathbb{B}$, Th, HN, He,
   - the six pariahs J_1, J_3, J_4, O'N, Ly, Ru.

Conversely, every group in this list is simple. There are some repetitions in this list [...].

The full proof of this theorem goes well beyond the scope of this thesis. It was originally scattered over hundreds of journal articles written by about 100 authors, and quite recently completed in 2004. Starting with Gorenstein, Lyons, and Solomon, 1994, the Second Generation

1 Some of the details of the CFSG are omitted where an ellipsis ('[...]') was used, as they are not relevant for the rest of this thesis.

Proof project, or by the authors' initials, the 'GLS' project, was initiated to provide a more accessible proof of the CFSG. In this yet unfinished book series, the mathematics behind the original proof is rewritten, condensed to 'just' 12 volumes of about 500 pages each. Currently, the first 10 volumes have been published, the latest being Capdeboscq et al., 2023.

It is precisely due to the vastness of the theorem's proof, that it is very suitable for formalisation. Currently, only mathematicians with an extremely deep understanding of the subject can understand the proof, and verify parts of it. By formalising the proof in a proof assistant, such as Lean, the complete proof can be verified at once. Consequently, the results can be made accessible to a much larger audience, allowing for further developments in the field of group theory.

5 GOLAY CODE

The Golay code is a type of linear code, which is a concept from coding theory. First, we will introduce the basic definitions and results from coding theory that are necessary to understand the Golay code. Then, we will provide a construction of the Golay code using the vertex adjancency of the icosahedron.

5.1 LINEAR CODES

Definition 14 (Linear code)**.** A q-ary linear code C of length n and dimension d is a linear subspace with dimension d of the vector space $\mathbb{F}_q^n$ where $\mathbb{F}_q$ is the finite field with q elements. The elements of C are called codewords.

Linear codes can be used for error detection and correction in noisy data transmission. For example, binary messages can be divided in chunks of d bits, which are encoded as codewords in a binary linear code (i.e. $q = 2$). As in practice $n > d$, these codewords contain information that is in a sense redundant. The best error-correcting codes are designed such that the information can be recovered even if some bits are corrupted, making use of this redundancy. Even when the received codeword is corrupted by noise, the receiver can still decode the message by finding the closest codeword in the code. Although in this thesis we are not so interested in this application, the mathematical properties of linear codes will also be of interest in the context of the Conway groups.

We can define a notion of distance between codewords in a linear code, which is also indicative of the error-correcting capabilities of the code. Namely, as the distance between two codewords increases, the number of errors that can be corrected also increases, making the code more robust against noise. Related to this notion of distance is the weight of a codeword.

Definition 15 (Hamming distance)**.** The Hamming distance between two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^n$ (cf. codewords in a linear code) is the number of coordinates in which they differ. This Hamming distance is denoted by $d_H(\mathbf{u}, \mathbf{v})$, with $d_H : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{N}$.

Definition 16 (Codeword weight)**.** The weight of a vector $\mathbf{u} \in \mathbb{F}_q^n$ is the number of nonzero coordinates in $\mathbf{u}$. This weight is denoted by $w(\mathbf{u})$, with $w : \mathbb{F}_q^n \to \mathbb{N}$ and clearly, $w(\mathbf{u}) = d_H(\mathbf{u}, \mathbf{0})$.

Note that for binary codes, we find that $w(\mathbf{u}) = \mathbf{u} \cdot \mathbf{u}$ where $\cdot$ denotes the standard vector dot product. The relation between the Hamming distance and the weight of codewords is given by the following lemma.

Lemma 9. For a linear code C, the minimum nonzero Hamming distance

$$\min_{\mathbf{u},\mathbf{v} \in C, \mathbf{u} \neq \mathbf{v}} d_H(\mathbf{u}, \mathbf{v})$$

is equal to the minimum nonzero weight

$$\min_{\mathbf{0} \neq \mathbf{u} \in C} w(\mathbf{u}).$$

Proof. Let $\mathbf{u}, \mathbf{v} \in C$ be two distinct codewords such that their Hamming distance is minimal, and let $\mathbf{0} \neq \mathbf{x} \in C$ be a codeword such that $w(\mathbf{x})$ is minimal. Then, $d_H(\mathbf{u}, \mathbf{v}) = w(\mathbf{v} - \mathbf{u})$, and $w(\mathbf{v} - \mathbf{u}) \geq w(\mathbf{x})$ as $\mathbf{v} - \mathbf{u} \neq \mathbf{0}$ and $w(\mathbf{x})$ is minimal. Conversely, $w(\mathbf{x}) = d_H(\mathbf{x}, \mathbf{0}) \leq d_H(\mathbf{u}, \mathbf{v})$ as $\mathbf{x} \neq \mathbf{0}$ and $d_H(\mathbf{u}, \mathbf{v})$ is minimal. We conclude that $d_H(\mathbf{u}, \mathbf{v}) = w(\mathbf{x})$, and consequently, the minimum nonzero Hamming distance is equal to the minimum nonzero weight. □

For binary codes in particular, we can derive an interesting relation between the weight of two codewords and the weight of their sum.

Lemma 10. For any two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_2^n$, the weight of their sum is related to their own weights by

$$w(\mathbf{u} + \mathbf{v}) = w(\mathbf{u}) + w(\mathbf{v}) - 2w(\mathbf{u} \circ \mathbf{v}),$$

where $\circ$ denotes the pointwise product of the vectors.

Proof. Let $\mathbf{u} = (u_1, u_2, \dots, u_n)$ and $\mathbf{v} = (v_1, v_2, \dots, v_n)$. Then, the sum $\mathbf{u} + \mathbf{v}$ is given by

$$(u_1 + v_1, u_2 + v_2, \dots, u_n + v_n).$$

Then, for all $0 \leq i \leq n$, we have that $u_i + v_i \equiv 1 \pmod 2$ if and only if $u_i \neq v_i$, and $u_i + v_i \equiv 0 \pmod 2$ otherwise. Also, $u_i \circ v_i = 1$ if and only if $u_i = v_i = 1$, and $u_i \circ v_i = 0$ otherwise. We find that $w(\mathbf{u}+\mathbf{v}) = w(\mathbf{u}) + w(\mathbf{v}) - 2w(\mathbf{u} \circ \mathbf{v})$, as claimed: only in coordinates where both **u** and **v** are 1, the sum's weight is corrected (-2) by the pointwise product of the vectors. As our choice of **u** and **v** was arbitrary, we conclude that the lemma holds for all vectors in $\mathbb{F}_2^n$. □

We can define another interesting property on linear codes, namely self-orthogonality.

Definition 17 (Self-orthogonality)**.** A linear code C over the field F is self-orthogonal if for any two codewords $\mathbf{u}, \mathbf{v} \in C$, we have $\mathbf{u} \cdot \mathbf{v} = 0$.

Lemma 11. *If a linear code C of dimension d has a self-orthogonal basis $\mathcal{B}$, then C is self-orthogonal.*

Proof. Let $\mathbf{u}, \mathbf{v} \in C$. Given the basis $\mathcal{B}$, we can write **u** and **v** as

$$\mathbf{u} = \sum_{i=1}^{d} \alpha_i \mathbf{b}_i,$$
$$\mathbf{v} = \sum_{i=1}^{d} \beta_i \mathbf{b}_i,$$

where $\mathbf{b}_i \in \mathcal{B}$ and $\alpha_i, \beta_i \in F$. We find by linearity of the dot product that

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{d} \sum_{j=1}^{d} \alpha_i \beta_j (\mathbf{b}_i \cdot \mathbf{b}_j) = 0,$$

as the basis $\mathcal{B}$ is self-orthogonal. As our choice of **u** and **v** was arbitrary, we conclude that C is self-orthogonal. □

Binary self-orthogonal codes have an interesting property regarding the weight of their codewords, which is demonstrated in the following lemma.

Lemma 12. *If a binary linear code C is self-orthogonal, then all its codewords have even weight.*

Proof. Let $\mathbf{u} \in C$ be a codeword. Then, $\mathbf{u} \cdot \mathbf{u} = 0$ by self-orthogonality. As $w(\mathbf{u}) \equiv \mathbf{u} \cdot \mathbf{u} \equiv 0 \pmod 2$, we find that $w(\mathbf{u})$ is even. As our choice of **u** was arbitrary, we conclude that all codewords in C have even weight. □

Another interesting property of binary codes is that of being doubly even.

Definition 18 (Doubly even code)**.** A binary linear code is called doubly even if all its codewords have a weight divisible by 4.

Lemma 13. If a binary linear code C of dimension d is self-orthogonal, and has a basis $\mathcal{B}$ with codewords of weight divisible by 4, then C is a doubly even code.

Proof. Let $\mathbf{u} \in C$. Then, $\mathbf{u}$ can be written as a linear combination of the basis $\mathcal{B}$:

$$\mathbf{u} = \sum_{i \in U} \mathbf{b}_i,$$

where $\mathbf{b}_i \in \mathcal{B}$ and $U \subseteq \{1, \ldots, d\}$. We can now prove that C is doubly even by induction on the size of U. Suppose $|U| = 1$. Then, $\mathbf{u} = \mathbf{b}_i$ for some $\mathbf{b}_i \in \mathcal{B}$, and by assumption $w(\mathbf{b}_i)$ is divisible by 4. Now, suppose that $|U| = k$ and assume the induction hypothesis holds, that for all $V \subseteq \{1, \ldots, d\}$ such that $|V| = k - 1$, the codeword $\sum_{i \in V} \mathbf{b}_i$ has weight divisible by 4. Then, we can choose some $i \in U$. Let $V := U \setminus \{i\}$ and $\mathbf{v} := \sum_{j \in V} \mathbf{b}_j$. By Lemma 10, we find:

$$w(\mathbf{u}) = w(\mathbf{b}_i) + w(\mathbf{v}) - 2w(\mathbf{b}_i \circ \mathbf{v}).$$

As $w(\mathbf{b}_i)$ is divisible by 4 by assumption, $w(\mathbf{v})$ is divisible by 4 by the induction hypothesis, and $2w(\mathbf{b}_i \circ \mathbf{v})$ is divisible by 4 by Lemma 12, we find that $w(\mathbf{u})$ is divisible by 4. By induction, we can conclude that all codewords in C have weight divisible by 4 and hence C is a doubly even code. $\square$

Finally, to construct the Golay code, we will make use of the concept of a generator matrix:

Definition 19 (Generator matrix)**.** A generator matrix of a q-ary linear code C of length n and dimension d is a $d \times n$ matrix over $\mathbb{F}_q$ whose rows form a basis for C.

5.2 CONSTRUCTION USING THE ICOSAHEDRON

We now have all the tools from coding theory to construct a mathematical object called the Golay code, which in turn will be used in the construction of the Leech lattice. Although irrelevant for this thesis, the Golay code is not only of mathematical interest, but also found practical applications in the Voyager space program of NASA, where it was used to successfully transmit color images from outer space.

Definition 20 (Golay code)**.** The (extended binary)[1]Golay code is the binary linear code of length 24 and dimension 12 such that the Hamming distance between any two distinct codewords is at least 8.

It is not immediately clear that such code exists, nor that it is unique up to the reordering of vector coordinates. To show this, we will first provide a construction of the Golay code.

5.2.1 *Generator matrix*

Lemma 14. A generator matrix of the Golay code is given by $(\mathbf{I}_{12} \mid \mathbf{A})$, where $\mathbf{I}_{12}$ is the 12×12 identity matrix, and $\mathbf{A}$ is the complement of the adjacency matrix of the icosahedron.

Proof. The adjacency matrix of the icosahedron is given by the following 12×12 matrix:

$$\mathbf{A} = \begin{pmatrix} 1&0&0&1&1&1&1&1&0&0&0&1 \\ 0&1&0&0&1&1&1&1&1&0&1&0 \\ 0&0&1&0&0&1&1&1&1&1&0&1 \\ 1&0&0&1&0&0&1&1&1&1&1&0 \\ 1&1&0&0&1&0&0&1&1&1&0&1 \\ 1&1&1&0&0&1&0&0&1&1&1&0 \\ 1&1&1&1&0&0&1&0&0&1&0&1 \\ 1&1&1&1&1&0&0&1&0&0&1&0 \\ 0&1&1&1&1&1&0&0&1&0&0&1 \\ 0&0&1&1&1&1&1&0&0&1&1&0 \\ 0&1&0&1&0&1&0&1&0&1&1&1 \\ 1&0&1&0&1&0&1&0&1&0&1&1 \end{pmatrix},$$

where the entry at (i, j) is 0 if vertices i and j are adjacent, and 1 otherwise, using the labelling of the vertices of the icosahedron as in Figure 5.2.1[2]. Clearly, the rows of $(\mathbf{I}_{12} \mid \mathbf{A})$ form the basis, which we will denote with $\mathcal{B}$, for a 12-dimensional linear subspace of $\mathbb{F}_2^{24}$, which we will denote by C. We will show that the linear code C is indeed the Golay code, by showing that the minimum nonzero Hamming distance between two distinct codewords is 8. By inspecting the rows of $(\mathbf{I}_{12} \mid \mathbf{A})$ we find that $\mathcal{B}$ is self-orthogonal, hence C is self-orthogonal by Lemma 11. Also, the codewords in the basis $\mathcal{B}$, or the *basis codewords*, have weight $1 + (12 - 5) = 8 \mid 4$, hence by Lemma 13, we find that C is a doubly-even code. Next, suppose there exists a codeword $\mathbf{0} \neq \mathbf{u} \in C$ with weight $w(\mathbf{u}) < 8$. Then, as C is a doubly even code, we find that $w(\mathbf{u}) = 4$. Clearly, $\mathbf{u}$ is a sum of at most 4 basis codewords, but we can check that no such sum has weight 4 (either by iterating over the $12 + (12 \cdot 11)/2! + (12 \cdot 11 \cdot 10)/3! + (12 \cdot 11 \cdot 10 \cdot 9)/4! = 793$ possible sums, or using the symmetry of the icosahedron), thus we arrive at a

1 We will refer to the extended binary Golay code as the Golay code. There are other Golay codes, but we will not consider them in this text.

contradiction. So the minimum nonzero weight of a codeword in C is 8, and consequently, the minimum nonzero Hamming distance between two distinct codewords is 8 by Lemma 9. In conclusion, C is the Golay code as defined in Definition 20. $\square$

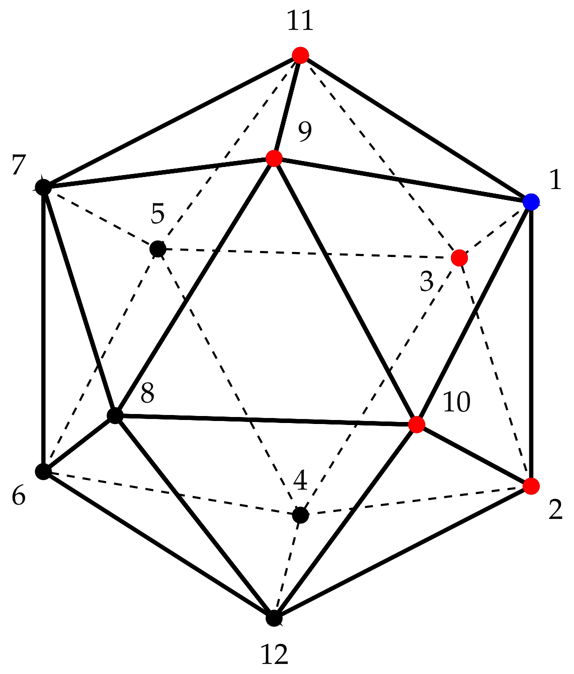

Figure 5.2.1: Icosahedron with labelled vertices: vertices adjacent to vertex 1 (highlighted in blue) are colored red.

5.2.2 *Cardinality of codewords by weight*

As the Golay code is a linear code, there are $2^{12} = 4096$ codewords in total. The weight distribution of these codewords is as follows:

- There is 1 codeword with all coordinates zeroes ('zero codeword').
- There is 1 codeword with all coordinates ones.
- There are 759 codewords with weight 8. As the Hamming distance between two distinct weight 8 codewords is at least 8, we find that no 5 coordinates have shared ones. In other words, the $\binom{24}{5}$ sets of 5 coordinates are in precisely one weight 8 codeword each, which in turn contain $\binom{8}{5}$ of such sets. Indeed, $\binom{24}{5}/\binom{8}{5} = 759$.
- There are 759 codewords with weight 16. They correspond with the complements of the weight 8 codewords.

- There are 2576 codewords with weight 12. This is the remainder of codewords: $4096 - 1 - 1 - 759 - 759 = 2576$.

5.2.3 *Automorphisms of the Golay code*

The aforementioned construction is just one of the ways to construct a linear code that satisfies the properties of the Golay code, as defined in Definition 20. In the chapter on M_{24} of The Finite Simple Groups Wilson, 2009, it is shown that this version of the Golay code is unique up to the reordering of vector coordinates, and its automorphism group is isomorphic to the Mathieu group M_{24}. This proof is beyond the scope of this thesis, but we will use this fact in proving the simplicity of the Conway group Co_1 in Chapter 7.

6 LEECH LATTICE

We will now lift the Golay code to a geometric structure called a lattice. First, we will define the general concept of a lattice. Then, we will construct the Leech lattice, which is closely related to the Golay code.

6.1 LATTICES

Definition 21 (Lattice)**.** An n-dimensional lattice Λ is an infinite additive group of vectors in $\mathbb{R}^n$, generated by a set of n linearly independent vectors. As such, any lattice can be written as

$$\Lambda = \left\{ \sum_{i=1}^{n} s_i \mathbf{b}_i \mid s_i \in \mathbb{Z} \right\},$$

where $\mathbf{b}_1, \ldots, \mathbf{b}_n$ form a basis of $\mathbb{R}^n$.

An example of a 2D lattice is the hexogonal lattice, as shown in Figure **??**. It is clear that all its points can be generated by integer multiples of the colored basis vectors.

We can define a norm over the lattice vectors, as follows:

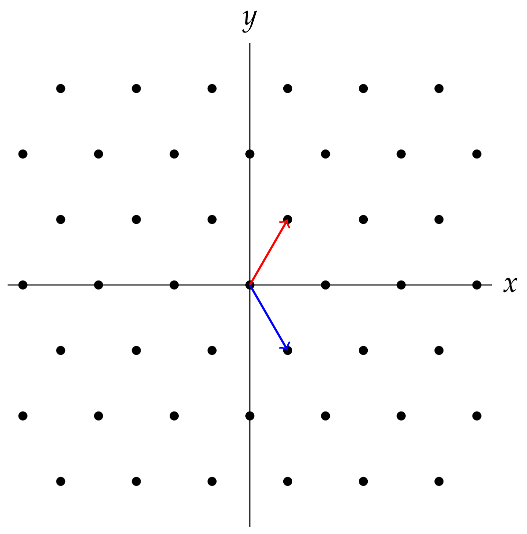

Figure 6.1.1: The hexagonal lattice in $\mathbb{R}^2$, with a basis indicated by the red and blue vectors.

Definition 22 (Lattice point norm)**.** For an n-dimensional lattice Λ, the norm of a vector $\mathbf{v} \in \Lambda$ is defined as

$$\|\mathbf{v}\| = \mathbf{v} \cdot \mathbf{v},$$

where $\cdot$ denotes the standard inner dot product over $\mathbb{R}^n$, i.e. the squared Euclidean length.

Furthermore, we can define the following properties of lattice, which we will use in our definition of the Leech lattice:

Definition 23 (Integrality)**.** A lattice is called integral if the standard inner dot product of any two vectors in Λ is an integer.

Definition 24 (Integral lattice parity)**.** A lattice, if it is integral, is called even if the norms of all vectors in Λ are even, and odd otherwise.

Definition 25 (Lattice determinant)**.** The determinant of a lattice Λ is the determinant of the $n \times n$ matrix with entries $\mathbf{b}_i \cdot \mathbf{b}_j$ for $1 \leq i, j \leq n$ where $\mathbf{b}_1, \ldots, \mathbf{b}_n$ form a basis of Λ.

Note that the determinant of a lattice is independent of the choice of basis up to a sign change, which follows from linear algebra on the properties of the determinant of a matrix.

Definition 26 (Unimodularity)**.** A lattice is called unimodular if its determinant is ± 1.

If we take the hexagonal lattice in Figure **??** as an example, with basis vectors $\mathbf{b}_1 = (1, \sqrt{3})$ and $\mathbf{b}_2 = (1, -\sqrt{3})$, we find that the lattice is integral and even, but not unimodular. The lattices $\mathbb{Z}^n$ with $n \in \mathbb{Z}^+$, however, are integral and unimodular (consider for example the basis $\mathbf{b}_1 = (1, 0, \ldots, 0), \mathbf{b}_2 = (0, 1, \ldots, 0), \ldots, \mathbf{b}_n = (0, 0, \ldots, 1)$).

6.2 CONSTRUCTION USING THE GOLAY CODE

Now we have defined what a lattice is, we can define the Leech lattice using the properties as defined in the previous section.

Definition 27 (Leech lattice)**.** The Leech lattice is the 24-dimensional even unimodular lattice such that the norm of every nonzero vector is at least 4.

It is not immediately clear that such lattice exists, so we will provide a construction of a lattice that satisfies this definition using the Golay code. This construction will allow us to exhibit certain properties of the Leech lattice in a more explicit fashion, and will be used to construct the automorphism group Co_0 in the next chapter.

Lemma 15. The Leech lattice can be constructed as the set of all vectors $\frac{1}{\sqrt{8}}(x_1, \ldots, x_{24})$ with $x_i \in \mathbb{Z}$ for all $1 \leq i \leq 24$ such that

$$\sum_{i=1}^{24} x_i \equiv 4x_1 \equiv 4x_2 \equiv \ldots \equiv 4x_{24} \pmod 8 \text{, and} \tag{6.1}$$

$$([x_1 \equiv k], [x_2 \equiv k], \ldots, [x_{24} \equiv k]) \in \mathcal{G} \tag{6.2}$$

for all $k \in \mathbb{Z}/4\mathbb{Z}$, where $[\ldots]$ denotes the Iverson bracket[1], and $\mathcal{G}$ is the Golay code.

Proof. Let Λ denote the lattice as constructed above, which is supposedly the Leech lattice. Clearly, Λ is a lattice as an infinite group under addition of dimension 24 (consider the vector with all coordinates zero, except for one 8). The scalar $\frac{1}{\sqrt{8}}$ is chosen such that unimodularity holds, and correspondingly the minimal norm condition is satisfied. Namely, if we construct nonzero vectors using the relations 6.1 and 6.2 with as small a norm as possible, particularly with every coordinate (when scaled by $\sqrt{8}$) having the same parity and 0 or at least 8 coordinates in each congruence class modulo 4, we find:

- A vector with 8 coordinates equal to 2, the rest zeroes. This vector has norm $\left(\frac{1}{\sqrt{8}}\right)^2 \cdot 8 \cdot 2^2 = 32/8 = 4$.
- A vector with 2 coordinates equal to 4, the rest zeroes. This vector has norm $\left(\frac{1}{\sqrt{8}}\right)^2 \cdot 2 \cdot 4^2 = 32/8 = 4$.
- A vector with one coordinate equal to 3, the rest -1. This vector has norm $\left(\frac{1}{\sqrt{8}}\right)^2 \cdot (3^2 + 23 \cdot (-1)^2) = 32/8 = 4$.

So, indeed every nonzero vector in Λ has norm at least 4.

Furthermore, for any vector $\frac{1}{\sqrt{8}}(x_1, \ldots, x_{24}) \in \Lambda$, the sum of squares $\sum_{i=1}^{24} x_i^2$ is divisible by 16, hence the lattice is even. □

1 The Iverson bracket is defined as:

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true,} \\ 0 & \text{otherwise,} \end{cases}$$

for any logical statement P.

6.3 STRUCTURE OF THE LEECH LATTICE

From here on, we will refer to the Leech lattice as Λ. We will now take a closer look at the structure of the Leech lattice, as this will be important to understand its automorphism group, from which we will derive the simple Conway groups in the next chapter.

6.3.1 *Minimal vectors*

We will first consider all minimal vectors of the Leech lattice, which are the vectors of smallest nonzero norm in the lattice. By Definition 27, we know that the norm of any nonzero vector in the Leech lattice is at least 4. Let $\mathbf{x} \in \Lambda$ such that $\|\mathbf{x}\| = 4$ and let us write $\mathbf{x} = \frac{1}{\sqrt{8}}(x_1, \ldots, x_{24})$. Then, $\sum_{i=1}^{24} x_i^2 = 32$. Using our construction from Lemma 15, we find that $\mathbf{x}$ must be of shapes:

- $\frac{1}{\sqrt{8}}((\pm 4)^2, 0^{22})$ (i.e. ± 4 in two coordinates and zeroes elsewhere), which corresponds with the zero codeword of the Golay code, such that there are $\binom{24}{2}$ positions to place the ± 4, and 2^2 ways to choose signs, hence there are $\binom{24}{2} \cdot 2^2 = 1104$ such vectors.
- $\frac{1}{\sqrt{8}}((\pm 2)^8, 0^{16})$ (i.e. ± 2 in eight coordinates and zeroes elsewhere), which corresponds with a codeword of weight 8 in the Golay code, of which there are 759, as shown in Subsection 5.2.2. There are 2^7 ways to choose the signs such that the congruency relation 6.1 holds (namely, we must have an even number of -2's), hence there are $759 \cdot 2^7 = 97152$ such vectors.
- $\frac{1}{\sqrt{8}}(\pm 3, (\mp 1)^{23})$ (i.e. ± 3 in one coordinate, ∓ 1 elsewhere), which corresponds with the zero codeword of the Golay code, such that there are 24 positions to place the ± 3. We can let all codewords of the Golay code act on such vectors by flipping the sign of the coordinates where the codeword has a 1, to have the Golay code condition 6.2 hold. Consequently, we have $24 \cdot 2^{12} = 98304$ such vectors.

In total, we find that there are $1104 + 97152 + 98304 = 196560$ minimal vectors in the Leech lattice. Note that these vectors correspond precisely with the orbits of the given shape vectors under the action of the semidirect product $2^{12} : M_{24}$: as discussed in Subsection 5.2.3, M_{24} gives the coordinate permutations fixing the Golay code, and 2^{12} acts as sign flips by the Golay code (cf. the reasoning for the last case).

6.3.2 *Vectors of other norms*

Similarly, we can consider the vectors of norm 6 in the Leech lattice by studying its orbits under the action of $2^{12} : M_{24}$. We will be more concise in our reasoning here, but we use the same methods as for the minimal vectors. We find the following shapes and orbit sizes of vectors of norm 6:

- $\frac{1}{\sqrt{8}}((\pm 2)^{12}, 0^{12})$, with $2576 \cdot 2^{11} = 5275648$ such vectors.
- $\frac{1}{\sqrt{8}}((\pm 2)^{7}, \pm 4, \mp 2, 0^{15})$, with $759 \cdot 16 \cdot 2^{8} = 3108864$ such vectors.
- $\frac{1}{\sqrt{8}}(\pm 5, (\pm 1)^{23})$, with $24 \cdot 2^{12} = 98304$ such vectors.
- $\frac{1}{\sqrt{8}}((\pm 3)^{3}, (\mp 1)^{21})$, with $\binom{24}{3} \cdot 2^{12} = 8290304$ such vectors.

In total, we find that there are $5275648 + 3108864 + 98304 + 8290304 = 16773120$ vectors of norm 6 in the Leech lattice.

For vectors of norm 8, we find the shapes and orbit sizes:

- $\frac{1}{\sqrt{8}}(\pm 8, 0^{23})$, with $24 \cdot 2 = 48$ such vectors.
- $\frac{1}{\sqrt{8}}(\pm 6, (\pm 2)^{7}, 0^{16})$, with $759 \cdot 8 \cdot 2^{7} = 777216$ such vectors.
- $\frac{1}{\sqrt{8}}((\pm 4)^{4}, 0^{20})$, with $\binom{24}{4} \cdot 2^{4} = 170016$ such vectors.
- $\frac{1}{\sqrt{8}}((\pm 4)^{2}, (\pm 2)^{8}, 0^{14})$, with $759 \cdot \binom{16}{2} \cdot 2^{2} \cdot 2^{7} = 46632960$ such vectors.
- $\frac{1}{\sqrt{8}}(\pm 4, (\pm 2)^{12}, 0^{11})$, with $2576 \cdot 12 \cdot 2 \cdot 2^{11} = 126615552$ such vectors.
- $\frac{1}{\sqrt{8}}((\pm 2)^{16}, 0^{8})$, with $759 \cdot 2^{15} = 24870912$ such vectors.
- $\frac{1}{\sqrt{8}}(\pm 5, (\mp 3)^{2}, (\pm 1)^{21})$, with $24 \cdot \binom{23}{2} \cdot 2^{12} = 24870912$ such vectors.
- $\frac{1}{\sqrt{8}}((\pm 3)^{5}, (\mp 1)^{19})$, with $\binom{24}{5} \cdot 2^{12} = 174096384$ such vectors.

In total, we find that there are $48 + 777216 + 170016 + 46632960 + 126615552 + 24870912 + 24870912 + 174096384 = 398034000$ vectors of norm 8 in the Leech lattice.

6.3.3 *Crosses*

If we consider the quotient $\Lambda/2\Lambda \cong \mathbb{F}_2^{24}$ and classify the resulting 2^{24} cosets, we can make some interesting observations. Let $\mathbf{x}, \mathbf{y} \in \Lambda$ and suppose that $\mathbf{x} \equiv \mathbf{y} \pmod{2\Lambda}$. Then $\mathbf{x} \pm \mathbf{y} \in 2\Lambda$, hence $\|\mathbf{x} \pm \mathbf{y}\| = 0$ or $\|\mathbf{x} \pm \mathbf{y}\| \geq 2^2 \cdot 4 = 16$. The former case is only possible if $\mathbf{x} = \pm\mathbf{y}$, in the latter case we find:

$$\begin{aligned} \|\mathbf{x}+\mathbf{y}\| &= \|\mathbf{x}\| + 2(\mathbf{x}\cdot\mathbf{y}) + \|\mathbf{y}\| \geq 16, \\ \|\mathbf{x}-\mathbf{y}\| &= \|\mathbf{x}\| - 2(\mathbf{x}\cdot\mathbf{y}) + \|\mathbf{y}\| \geq 16, \qquad \text{thus,} \\ \|\mathbf{x}\| + \|\mathbf{y}\| &\geq 16. \end{aligned} \tag{6.3}$$

Consequently, the zero vector is in its own congruency class, the vectors of norm 4 are in $196560/2 = 98280$ congruency classes with their negatives and the vectors of norm 6 are in $16773120/2 = 8386560$ congruency classes with their negatives. The vectors of norm 8 are also in congruence classes with their negatives, but also with perpendicular vectors of norm 8, satisfying full equality in 6.3. As sets of perpendicular vectors are linearly independent, these congruence classes, including negatives, contain $24 \cdot 2 = 48$ vectors. We have now accounted for all $1 + 98280 + 8386560 + 398034000/48 = 2^{24}$ congruence classes.

Definition 28 (Leech lattice cross)**.** A cross $\mathcal{C}$ is defined as a subset of the Leech lattice consisting of vectors of norm 8 such that for each pair $\mathbf{x}, \mathbf{y} \in \mathcal{C}$ one of the following three conditions holds:

- $\mathbf{x}$ and $\mathbf{y}$ are perpendicular, i.e. $\mathbf{x} \cdot \mathbf{y} = 0$,
- $\mathbf{x} = \mathbf{y}$, or,
- $\mathbf{x} = -\mathbf{y}$.

We can now associate the cosets of the Leech lattice modulo 2Λ for vectors of norm 8 with these crosses, each containing 48 vectors. In particular, we find that there are $398034000/48 = 8292375$ crosses in the Leech lattice. We will use these crosses to define the automorphism group of the Leech lattice in the next chapter.

7 CONWAY GROUPS

In this chapter, we will finally introduce the Conway groups. These groups were discovered by John Conway in 1968 by investigation of the Leech lattice (Conway, 1968; Conway, 1969). In his work, Conway designed a similar construction of the Leech lattice as we did in the previous chapter, and described the automorphism group for this highly symmetric lattice. He also discovered three associated, previously unknown sporadic simple groups. This caused a major surge in the discovery of the remaining sporadic simple groups, and marked Conway as one of the most influential mathematicians of the 20th century.

The largest of the Conway groups is the Conway group Co_0, which is the full automorphism group of the Leech lattice, and is not simple. The largest simple Conway group is the Conway group Co_1, which is the quotient of Co_0 by its center. The other Conway groups, Co_2 and Co_3, are subgroups of Co_0, both constrained to fix certain vectors of the Leech lattice. In this chapter, we will describe these groups in more detail and prove the simplicity of Co_1.

7.1 Co_0

We start with the largest of the Conway groups, Co_0. This is the authomorphism group of the Leech lattice, considering the isometries preserving the lattice, fixing the origin.

To describe Co_0 in more detail, we will consider how it acts on the crosses. We call the canonical cross $\mathcal{C}_0$ the cross consisting of the vectors $\frac{1}{\sqrt{8}}(c_1, \ldots, c_{24})$ where $c_i = \pm 8$ for one $1 \leq i \leq 24$ and $c_j = 0$ for all $j \neq i$. As is clear from our construction in Lemma 15, the canonical cross is fixed precisely by $2^{12} : M_{24}$ from the previous chapter. For the other crosses, we can find their stabilizers as follows. As determined before, each cross consists of 48 vectors, which are pairwise orthogonal, negatives or equal. Therefore, we can think of a cross as a set of 24 orthogonal pairs, where each vector forms such pair with its negative. We can consider the action of M_{24} acting on a cross by permuting the 24 orthogonal pairs, and the action of 2^{12} by swapping the order of the pairs, and we conclude that the other crosses are stabilized by conjugates of $2^{12} : M_{24}$ in Co_0. As by our counting argument in $\Lambda/2\Lambda$ in Subsection 6.3.3 we know that all norm 8 vectors fall into one of 8292375 crosses, we can conclude that Co_0 acts transitively on the set of crosses. Now, using Lemma 6, we find the order of Co_0 to be $|\,\mathrm{Co}_0\,| = 8292375 \cdot 2^{12} \cdot |M_{24}| = 8\,315\,553\,613\,086\,720\,000$. Also, $2^{12} : M_{24}$ is a maximal subgroup of Co_0. As its conjugates in Co_0 stabilize all crosses, we find that Co_0 acts primitively on the set of crosses. Furthermore, Co_0 is a perfect group, as Conway has shown in his original construction (Conway, 1968).

We can also show that Co_0 is not simple. Namely, the nontrivial subgroup generated by the scalar matrix $(-\mathbf{1}) := -1 \cdot I$, with I the identity matrix, is normal in Co_0: scalar matrices always commute with other matrices. In particular, all crosses are fixed by $\langle -\mathbf{1} \rangle$.

7.2 Co_1

The next Conway group we will consider is Co_1, which is defined as the quotient of Co_0 by its normal subgroup generated by the scalar matrix $-\mathbf{1}$. It has order $|\,\mathrm{Co}_1\,| = |\,\mathrm{Co}_0\,|/2 = 4\,157\,776\,806\,543\,360\,000$.

In this section, we will prove that Co_1 is simple. To this end, we will collect our conditions to apply Iwasawa's lemma. Primitivity of the action of Co_1 on the set of crosses follows naturally from the action of Co_0 on the set of crosses. Similary, Co_1 is perfect, as it is a quotient of the perfect group Co_0. Whereas Co_0 did not act faithfully on the set of crosses, we factored out precisely the normal subgroup fixing all crosses to obtain Co_1, hence Co_1 acts faithfully on the set of crosses.

Now, we only need to exhibit a particular structure of the stabilizers of the crosses in order to apply Iwasawa's lemma. As all stabilizers are conjugates of the stabilizer of the canonical cross, we can focus on its stabilizer, $2^{12} : M_{24}$. We will now demonstrate that its normal abelian subgroup 2^{12} generates all of $2^{12} : M_{24}$ by conjugation. To this end, we'll construct an alternate cross, labeled $\mathcal{C}'$, as follows. For clarity, let us write the coordinates of any $\mathbf{c} \in \mathcal{C}'$ in 6×4 blocks, as follows:

$$\mathbf{c} = \frac{1}{\sqrt{8}} \begin{pmatrix} c_1 & c_2 & c_3 & c_4 \\ c_5 & c_6 & c_7 & c_8 \\ c_9 & c_{10} & c_{11} & c_{12} \\ c_{13} & c_{14} & c_{15} & c_{16} \\ c_{17} & c_{18} & c_{19} & c_{20} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{pmatrix}, \tag{7.1}$$

where precisely one of the rows (blocks) is one of

$$\begin{aligned} &\pm(+4, +4, +4, +4), \\ &\pm(-4, -4, +4, +4), \\ &\pm(-4, +4, -4, +4), \\ &\pm(+4, -4, -4, +4), \end{aligned} \tag{7.2}$$

and the others are zeroes. One can easily verify that this is indeed a cross: we can construct $6 \cdot 2 \cdot 4 = 48$ such vectors, which are are in the Leech lattice as constructed in Lemma 15, have norm 8, and are pairwise perpendicular, negatives or equal. We now exhibit the nontrivial coordinate permutation $\pi \in M_{24}$ acting on the vectors $\mathbf{c} \in \mathcal{C}'$, written in matrix form as in 7.1, as follows:

$$\mathbf{c}^{\pi} = \frac{1}{\sqrt{8}} \begin{pmatrix} c_1 & c_2 & c_3 & c_4 \\ c_6 & c_5 & c_8 & c_7 \\ c_9 & c_{10} & c_{11} & c_{12} \\ c_{14} & c_{13} & c_{16} & c_{15} \\ c_{19} & c_{20} & c_{17} & c_{18} \\ c_{24} & c_{23} & c_{22} & c_{21} \end{pmatrix} \left| \begin{array}{l} \text{Type A: identity} \\ \text{Type B: swap columns 1-2, and 3-4} \\ \text{Type A} \\ \text{Type B} \\ \text{Type C: swap columns 1-3, and 2-4} \\ \text{Type D: swap columns 1-4, and 2-3,} \end{array} \right.$$

where the permutation of each row is of a different type, as labeled above. If we consider the action of these types on the rows in 7.2, we find the following permutations for each row:

- $\pm(+4, +4, +4, +4)$: fixed by all types,
- $\pm(-4, -4, +4, +4)$: fixed by A and B, but C and D act as a sign flip, mapping to $\pm(+4, +4, -4, -4)$,
- $\pm(-4, +4, -4, +4)$: fixed by A and C, but B and D act as a sign flip, mapping to $\pm(+4, -4, +4, -4)$,
- $\pm(+4, -4, -4, +4)$: fixed by A and D, but B and C act as a sign flip, mapping to $\pm(-4, +4, +4, -4)$.

Note that the action of π flips the signs of a total of 8 pairs in the cross, so it is indeed contained in M_{24}. Furthermore, the cross $\mathcal{C}'$ as a whole is fixed by π. Therefore, the action of π corresponds with a sign flip on this alternate cross, or, in other words, is conjugate to some $\rho \in 2^{12}$ acting on the canonical cross. As M_{24} is simple, no nontrivial subgroup of M_{24} is invariant under conjugation, so we conclude that 2^{12} generates all of $2^{12} : M_{24}$ by conjugation. Consequently, as all cross stabilizers are conjugate, 2^{12} generates all of Co_1 by conjugation.

Lemma 16. The group Co_1 is simple.

Proof. We have shown that Co_1 is perfect, and acts primitively and faithfully on the set of crosses of the Leech lattice. Moreover, we have demonstrated that the normal abelian subgroup 2^{12} of the canonical cross stabilizer $2^{12} : M_{24}$, generates all of Co_1 by conjugation. Therefore, we can apply Iwasawa's lemma to conclude that Co_1 is simple. $\square$

This is the main result of this thesis, and the aim of the next chapter is to show how part of this proof was formalised in the Lean theorem prover. For completeness, we will now also construct the other Conway groups Co_2 and Co_3, but we will not go into as much detail as before, and not prove their simplicity, nor formalise them.

7.3 Co_2 and Co_3

In this section, we will describe the Conway groups Co_2 and Co_3. These groups are subgroups of Co_1, and in particular they are constrained to fix a lattice vector of norm 4 and 6, respectively. It turns out that both Co_2 and Co_3 are simple. Furthermore, one can show that the action of Co_0 is transitive on both the vectors of norm 4 and 6, so, using the results from Section 6.3, we find the orders of their respective stabilizer subgroups are:

- $|\,\mathrm{Co}_2\,| = |\,\mathrm{Co}_0\,|/196560 = 42\,305\,421\,312\,000$,
- $|\,\mathrm{Co}_3\,| = |\,\mathrm{Co}_0\,|/16773120 = 495\,766\,656\,000$.

This process of construcing subgroups fixing part of the lattice is first described in great detail by Conway himself (Conway, 1969) and can be extended to vectors of other norms, and even to other simplices. Other groups that can be constructed in this way were already discovered, thus were not new (simple) groups, at the time. However, their new construction through the Leech lattice offers an elegant and uniform way to construct many groups that are of great interest in the broader study of finite groups.

8 FORMALISATION

In this chapter, we will discuss the formalisation of the finite simple Conway groups in Lean. We will start by introducing the Lean theorem prover and the mathlib library, then we will discuss the implementation of part of the proofs outlined earlier in this thesis. Part of the code is listed in the appendices and throughout this chapter. The full source code is listed on GitHub at `https://github.com/erikvdplas/mathlib4`.

8.1 LEAN THEOREM PROVER

Lean is an interactive theorem prover originally developed by Leonardo de Moura at Microsoft Research (Moura et al., 2015). The latest version of Lean, Lean 4, was released in 2023 and is the version used in this project. The aim of Lean is to provide a powerful and efficient tool for formalising mathematics and verifying proofs. The core idea is to encode mathematical definitions and theorems in a formal language, and use a sophisticated type checker to verify the correctness of proofs. Loosely speaking, if a proof written in Lean matches the type of a given proposition, and the code compiles, then the proof is verifiably correct. Alternatively, one can also skip certain proof steps using the 'sorry' keyword instead of a proof, but Lean will justly mark all dependent proofs as incomplete. The type checker itself is based on a small set of axioms and rules, which are used to derive all other results. This means that the only part of the system that needs to be trusted is this foundational *kernel*, which is the implementation of the type checker.

Furthermore, Lean is also designed to automate as much of the proving process as possible, by supplying a set of *tactics* which automate common proof steps. In these thesis we are mainly interested in pragmatic use of Lean, and will not go into further details about the underlying computer science theory. Instead, we will give many examples of Lean code in this chapter to illustrate how it can be used to formalise mathematics.

8.2 MATHLIB

Mathlib is an open-source project on Github, serving as a collection of definitions and proofs from all branches of mathematics (The mathlib community, 2020). It has a large community of contributors, primarily from the mathematical academic community. Guided by a select group of maintainers, anyone can contribute through pull requests. Communication regarding the development of mathlib takes place on the Zulip chat platform. Besides the more general results in (linear) algebra, analysis, and topology, mathlib also contains more specialized, larger projects: the classification of finite simple groups would also fit well within this library. In fact, many of the more fundamental group theoretic results are already present in mathlib, which we also gratefully used for this thesis.

8.3 IWASAWA'S LEMMA

At the time of writing, there is an open pull request in the mathlib repository formalising Iwasawa's Lemma (Chambert-Loir, 2024). In our fork of mathlib, we have already merged this pull request, and modified it in order to compile (see Appendix B).

Iwasawa's lemma applies to groups acting on sets with a certain structure. Reiterating Lemma 8, it is concerned with any perfect group G acting faithfully and primitively on a set Ω. Furthermore, the stabilizer G_a of some point $a \in \Omega$, has a normal abelian subgroup A generating the whole group G by conjugation. In the proof of the lemma we have demonstrated that all stabilizer subgroups are conjugate, so we can actually consider any stabilizer subgroup without loss of generality.

In 8.1, precisely this subgroup structure is implied, albeit in a different way. This 'Iwasawa structure' can now be used to build and prove the lemma. To exhibit the structure, one must show that for some specific group and associated group action on a set, we can construct the Iwasawa structure, i.e. provide an instance of such structure.

```
variable (M : Type*) [Group M] (α : Type*) [MulAction M α]

/-- The structure underlying the Iwasawa criterion -/
```

```
structure IwasawaStructure where
/-- The subgroups of the Iwasawa structure -/
  T : α → Subgroup M
/-- The commutativity property of the subgroups -/
  is_comm : ∀ x : α, (T x).IsCommutative
/-- The conjugacy property of the subgroups -/
  is_conj : ∀ g : M, ∀ x : α, T (g • x) = MulAut.conj g • T x
/-- The subgroups generate the group -/
  is_generator : iSup T = ⊤
```

Source Code 8.1: Iwasawa subgroup structure
(© Antoine Chambert-Loir, 2024)

Here, our objects of interest are denoted by the variables M and α, where M imposes a multiplicative action on α. Next, we can prove the lemma using this structure. In 8.2, we further premise that M is nontrivial (the trivial group is not considered simple) and perfect, and that the action is quasiprimitive and faithful. Note that quasipreprimitivity is also a less restrictive condition than primitivity, and is sufficient for the lemma to hold. However, we will not go into the detail about the difference here, as it is not relevant for the rest of the proof.

In proving the lemma, we first unpack what it means for a group to be simple, namely that any of its normal subgroups is either the trivial group or the whole group. Then, we introduce a normal subgroup N of M and show that it is either the trivial group or the whole group. In the next step, a generalized version of Iwasawa's lemma is used to show that any normal subgroup of M that does not fix all points contains the commutator subgroup of M. This corresponds with our proof of the complete lemma, without the final step, where we use that the commutator subgroup is the whole group, i.e. M is perfect. In the alternate case that N does fix all points, we show that N must be the trivial group by faithfulness of the action.

```
theorem IwasawaStructure.isSimpleGroup
        (is_nontrivial : Nontrivial M) (is_perfect : commutator M = ⊤)
        (is_qprim : IsQuasipreprimitive M α) (is_faithful : FaithfulSMul M
        ↪  α)
        (IwaS : IwasawaStructure M α) : IsSimpleGroup M := by
  apply IsSimpleGroup.mk
  intro N nN
  cases or_iff_not_imp_left.mpr (IwaS.commutator_le is_qprim nN) with
  | inl h =>
          refine' Or.inl (N.eq_bot_iff_forall.mpr fun n hn => _)
          apply is_faithful.eq_of_smul_eq_smul
          intro x
          rw [one_smul]
          exact Set.eq_univ_iff_forall.mp h x ⟨n, hn⟩
  | inr h => exact Or.inr (top_le_iff.mp (le_trans (ge_of_eq is_perfect)
  ↪  h))
```

Source Code 8.2: Iwasawa's lemma
(© Antoine Chambert-Loir, 2024)

In the proof of the generalized lemma concerning any normal subgroup N of M, that does not fix all points, we use the Iwasawa structure to show that N contains the commutator subgroup. This proof is a bit more involved, and differs slightly from our written proof as the quasipreprimitivity of the action is used and the subgroup structure is expressed differently. In 8.3, an important step that overlaps with our written proof is the (indirect) use of the second ismorphism theorem, using 'commutator_le_of_self_sup_commutative_eq_top' to show that the commutator subgroup is contained in N, as NA is the whole group, for some abelian subgroup A of M (cf. Lemma 3). This is a good example of how you can use nested lemmas in Lean to build up to a more complex result.

```
theorem IwasawaStructure.commutator_le
        (is_qprim : IsQuasipreprimitive M α) (IwaS : IwasawaStructure M α)
        {N : Subgroup M} (nN : N.Normal) (hNX : MulAction.fixedPoints N α
        ↪  ≠ ⊤) :
        commutator M ≤ N := by
  have is_transN := is_qprim.pretransitive_of_normal nN hNX
  have ntα : Nontrivial α := isnontrivial_of_nontrivial_action hNX
  obtain a : α := Nontrivial.to_nonempty.some
  apply nN.commutator_le_of_self_sup_commutative_eq_top ?_ (IwaS.is_comm
  ↪  a)
  -- We have to prove that N ⊔ IwaS.T x = ⊤
  rw [eq_top_iff, ← IwaS.is_generator, iSup_le_iff]
  intro x
  obtain ⟨g, rfl⟩ := MulAction.exists_smul_eq N a x
  rw [Subgroup.smul_def, IwaS.is_conj g a]
  rintro _ ⟨k, hk, rfl⟩
  have hg' : ↑g ∈ N ⊔ IwaS.T a := Subgroup.mem_sup_left (Subtype.mem g)
  have hk' : k ∈ N ⊔ IwaS.T a := Subgroup.mem_sup_right hk
  exact (N ⊔ IwaS.T a).mul_mem ((N ⊔ IwaS.T a).mul_mem hg' hk') ((N ⊔
  ↪  IwaS.T a).inv_mem hg')
```

Source Code 8.3: Generalized Iwasawa's lemma
(© Antoine Chambert-Loir, 2024)

In the next section, our aim is to construct the Conway groups Co_0 and Co_1, and discuss how we can use this formalised lemma to show that Co_1 is simple.

8.4 CONWAY GROUPS

In this section, we will describe the code written to formalise the Conway groups in Lean. Furthermore, we will discuss the formalisation of the simplicity of Co_1, and what is still missing to complete proof. As in our written construction, the Conway groups are constructed using the Leech lattice, so we will start by introducing the formalisation of lattices and the isometries preserving them, fixing the origin. The code for this section is listed in Appendix A.

8.4.1 *Lattices and automorphisms*

In 8.4, we define the class of integral lattices. This class extends the 'AddCommGroup' class, describing an abelian group with an addition operation, and the 'Inner' class, endowing it with an inner product operation mapping to the integers. We require a lattice to be 'Free' and 'Finite', such that it is generated by integer multiples of a finite basis of vectors. Furthermore, we require the inner product to be additive in the first argument, symmetric, and when applied to a vector with itself, non-negative and zero only for the zero vector. From these properties we can later show that the inner product is also additive in the second argument ('inner_add'), and that the inner product with the zero vector is zero ('inner_zero' and 'zero_inner'). We also define properties of lattices, namely its parity ('IsEven') and whether it is unimodular ('IsUnimodular').

```
class IntegralLattice (Λ : Type*) extends Inner ℤ Λ, AddCommGroup Λ where
  [free : Free ℤ Λ]
  [finite : Finite ℤ Λ]
  (add_inner : ∀ x y z: Λ, ⟪(x + y), z⟫_ℤ = ⟪x, z⟫_ℤ + ⟪y, z⟫_ℤ)
  (inner_sym : ∀ x y: Λ, ⟪x, y⟫_ℤ = ⟪y, x⟫_ℤ)
  (inner_self : ∀ x: Λ, ⟪x, x⟫_ℤ ≥ 0)
  (inner_self_eq_zero : ∀ x: Λ, ⟪x, x⟫_ℤ = 0 → x = 0)
```

Source Code 8.4: Integral lattice class

Next, we define equivalences, i.e. isometries fixing the origin, between lattices, as shown in 8.5. We can build upon the existing 'AddEquiv' structure, which describe equivalences preserving the addition operation. Hence, we only need to add the preservation of the inner product, which we do in the 'preserves_inner' field. Furthermore, we define an infix notation for lattice equivalence, for brevity. Additionally, we define a bit of boilerplate to make use of results from more generic equivalence structures, and help Lean automatically make the correct inferences.

```
structure IntegralLatticeEquiv (Λ₁ Λ₂ : Type*)
  [IntegralLattice Λ₁] [IntegralLattice Λ₂]
  extends Λ₁ ≃+ Λ₂ where
  (preserves_inner' : ∀ x y: Λ₁, ⟨toFun x, toFun y⟩_ℤ = ⟨x, y⟩_ℤ)

infixl:25 " ≃ₗ " => IntegralLatticeEquiv
```

Source Code 8.5: Lattice equivalence

Lastly, we define the group of automorphisms of a lattice, as shown in 8.6. An automorphism is defined as an equivalence between the lattice and itself, and the group structure is defined by composition of equivalences.

```
variable (Λ : Type*) [IntegralLattice Λ]

@[reducible]
def IntegralLatticeAut := IntegralLatticeEquiv Λ Λ

namespace IntegralLatticeAut

instance : Group (IntegralLatticeAut Λ) where
  mul f g := IntegralLatticeEquiv.trans g f
  one := IntegralLatticeEquiv.refl Λ
  inv := IntegralLatticeEquiv.symm
  mul_assoc := by intros; rfl
  one_mul := by intros; rfl
  mul_one := by intros; rfl
  mul_left_inv := IntegralLatticeEquiv.self_trans_symm
```

Source Code 8.6: Lattice automorphism group

8.4.2 *Leech lattice and crosses*

We can now define the Leech lattice, by extending our 'IntegralLattice' class with the requirements uniquely defining the Leech lattice, as shown in 8.7 (cf. Definition 27). Note in particular that we inherit the inner product from the 'IntegralLattice' class, and can use 'IsEven' and 'IsUnimodular' as properties.

Demonstrating the uniqueness of the Leech lattice in Lean is still an open problem, and constructing an instance of the Leech lattice built from the Golay code also turned out to be too challenging in the given time. The latter would also allow us to concretely demonstrate certain properties of the Leech lattice, similar to our written proofs in Chapter 6.

```
class LeechLattice (Λ : Type*) extends IntegralLattice Λ where
  (even : IsEven Λ)
  (unimodular : IsUnimodular Λ)
  (rank_eq_24 : finrank ℤ Λ = 24)
  (min_norm : ∀ (x : Λ), x ≠ 0 → ⟪x, x⟫_ℤ ≥ 4)
```

Source Code 8.7: Leech lattice

Next, we define the notion of Leech lattice crosses, and the action of Leech lattice automorphisms on them, as shown in 8.8.

```
variable (Λ : Type*) [LeechLattice Λ]

structure Cross where
  (carrier : Set Λ)
  (norm_8 : ∀ x ∈ carrier, ⟪x, x⟫_ℤ = 8)
  (perpendicular : ∀ x ∈ carrier, ∀ y ∈ carrier, x ≠ y → ⟪x, y⟫_ℤ = 0 ∨ x
  ↪  = -y)

instance : SMul (IntegralLatticeAut Λ) (Cross Λ) where
  smul f C := {
        carrier := f '' C.carrier,
        norm_8 := by
          intro x h
          simp at h
          rcases h with ⟨y, hy, rfl⟩
          rw [f.preserves_inner]
          apply C.norm_8 y hy
        perpendicular := by
          intro x hx y hy hxy
          simp at hx hy
          rcases hx with ⟨x, hx, rfl⟩
          rcases hy with ⟨y, hy, rfl⟩
          rw [f.preserves_inner]
          have := C.perpendicular x hx y hy
          rw [← map_neg f]
          simp_all [- map_neg]
  }
```

Source Code 8.8: Leech lattice crosses, and automorphism action

8.4.3 *Conway groups and the simplicity of* Co_1

Now, we finally define the Conway groups, as shown in 8.9. We define Co_0 as the group of integral lattice automorphisms of the Leech lattice, and Co_1 as the quotient of Co_0 by its center.

```
variable (Λ : Type*) [LeechLattice Λ]

abbrev Conway₀ := IntegralLatticeAut Λ
abbrev Conway₁ := Conway₀ Λ / (Subgroup.center (Conway₀ Λ))
```

Source Code 8.9: Conway groups Co_0 and Co_1

There are many open problems left to solve in order to prove simplicity of Co_1. If we choose to go along the lines of the written proof, we need to describe the action of Co_1 on the set of crosses, and show that it acts faithfully on this set. Furthermore, we need to show that Co_0 (and consequently, Co_1) is perfect, and acts primitively on the set of crosses, or alternatively, quasipreprimitively. Lastly, we need to exhibit the Iwasawa structure with subgroups of Co_1, namely the maximal abelian subgroups of the cross stabilizers. Then, we can finally apply Iwasawa's lemma from the previous section to show that Co_1 is simple. This is left as future work.

9 CONCLUSION & FUTURE WORK

In this thesis, we have introduced the Leech lattice and the Conway groups. We have shown how the Leech lattice can be constructed from the Golay code, and in turn how the Conway groups can be constructed from the Leech lattice. We have calculated the order of all Conway groups, and proven that Co_1 is simple. Part of this last proof was formalised in the Lean theorem prover.

The work is far from complete, however. Some of the written proof in Chapter 7 is still a bit informal or hand-wavy, and a lot of details need to be filled in before it can be formalised. Therefore, in the Lean code a lot of declerations still use the 'sorry' keyword to skip proof steps. Due to the steep learning curve of Lean and the complexity of the proof, it was not feasible to complete the proof in the time frame of this thesis. However, the existing structures could provide a good starting point for future work, possibly by other students or researchers.

Furthermore, the formalisation of the other Conway groups, Co_2 and Co_3, is still open. These groups are subgroups of Co_1 that fix certain vectors of the Leech lattice, as succinctly described in Section 7.3. In a broader context, the formalisation of the Classification of Finite Simple Groups, as discussed in Chapter 4, is a very interesting but ambitious project. Precisely because of the complexity and sheer size of the proof, it is very suitable for formalisation. This thesis has demonstrated that the Lean theorem prover is well-suited for research concerning finite groups; nevertheless, our code is a neglible contribution to this project, and much more work is needed to achieve this goal.

Acknowledgements. I would like to thank my supervisor, Johan Commelin, for his guidance and support during this project. His first-hand knowledge of Lean and his literature recommendations proved invaluable. Besides his technical expertise, he also provided me with the positive encouragement, flexibility and understanding I needed at times, and for that I am very grateful.

A LEAN CODE

Mathlib ⟩ *GroupTheory* ⟩ *IntegralLattice* ⟩ *Basic.lean* (New file)

```
/-
Copyright (c) 2024 Erik van der Plas. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Erik van der Plas
-/
import Mathlib.Analysis.InnerProductSpace.Basic
import Mathlib.LinearAlgebra.FreeModule.Basic
import Mathlib.LinearAlgebra.Matrix.Determinant

open Module
open FiniteDimensional

/-- An *integral lattice* is a finitely generated free abelian group
of finite rank `n` with a bilinear form that takes integer values. -/
class IntegralLattice (Λ : Type*) extends Inner ℤ Λ, AddCommGroup Λ where
  [free : Free ℤ Λ]
  [finite : Finite ℤ Λ]
  (add_inner : ∀ x y z: Λ, ⟪(x + y), z⟫_ℤ = ⟪x, z⟫_ℤ + ⟪y, z⟫_ℤ)
  (inner_sym : ∀ x y: Λ, ⟪x, y⟫_ℤ = ⟪y, x⟫_ℤ)
  (inner_self : ∀ x: Λ, ⟪x, x⟫_ℤ ≥ 0)
  (inner_self_eq_zero : ∀ x: Λ, ⟪x, x⟫_ℤ = 0 → x = 0)

attribute [instance] IntegralLattice.free IntegralLattice.finite

namespace IntegralLattice

variable {Λ : Type*} [IntegralLattice Λ]

/-- Inner product is also additive in the second argument by symmetry. -/
lemma inner_add (x y z : Λ) : ⟪x, (y + z)⟫_ℤ = ⟪x, y⟫_ℤ + ⟪x, z⟫_ℤ := by
  rw [inner_sym, add_inner, inner_sym y x, inner_sym z x]

/-- Bilinear form of the inner product on an integral lattice as additive
↪  monoid homomorphism.  -/
def InnerBilin : Λ →+ Λ →+ ℤ :=
  AddMonoidHom.mk' (fun x ↦ AddMonoidHom.mk' (fun y ↦ ⟪x, y⟫_ℤ) (inner_add
  ↪  x)) <| by
    intro x y
    ext z
    dsimp
    apply add_inner

/-- Inner product with zero vector is always zero. -/
@[simp]
lemma inner_zero (x : Λ) : ⟪x, 0⟫_ℤ = 0 := by
  apply (InnerBilin x).map_zero

@[simp]
lemma zero_inner (y : Λ) : ⟪0, y⟫_ℤ = 0 := by
  rw [inner_sym, inner_zero]

variable (Λ)

/-- A lattice is even if all norms are even. -/
def IsEven : Prop := ∀ x: Λ, Even ⟪x, x⟫_ℤ

def gramMatrix {Λ ι : Type*} [IntegralLattice Λ] (v : ι → Λ) :=
  Matrix.of (fun i j ↦ ⟨v i, v j⟩_ℤ)

/-- The determinant of a lattice is defined as the determinant
of the Gram matrix of its basis. -/
noncomputable
def determinant : ℤ :=
  (gramMatrix (Free.chooseBasis ℤ Λ)).det

/-- A lattice is unimodular if its determinant is equal to 1. -/
def IsUnimodular : Prop := determinant Λ = 1

end IntegralLattice
```

Mathlib ⟩ GroupTheory ⟩ IntegralLattice ⟩ Equiv.lean (New file)

```
/-
Copyright (c) 2024 Erik van der Plas. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Erik van der Plas
-/
import Mathlib.GroupTheory.IntegralLattice.Basic

open IntegralLattice

/-- A lattice equivalence corresponds to structure preserving isometries
↪  fixing the origin,
  i.e. equivalence preserving vector additions and inner products. -/
structure IntegralLatticeEquiv (Λ₁ Λ₂ : Type*)
  [IntegralLattice Λ₁] [IntegralLattice Λ₂]
  extends Λ₁ ≃+ Λ₂ where
  (preserves_inner' : ∀ x y: Λ₁, ⟨toFun x, toFun y⟩_ℤ = ⟨x, y⟩_ℤ)

/-- Notation for an `IntegralLatticeEquiv`. -/
infixl:25 " ≃ₗ " => IntegralLatticeEquiv

namespace IntegralLatticeEquiv

variable {Λ₁ Λ₂ Λ₃ : Type*} [IntegralLattice Λ₁] [IntegralLattice Λ₂]
↪  [IntegralLattice Λ₃]

/-- Lattice equivalence relation is reflexive. -/
def refl (Λ : Type*) [IntegralLattice Λ] : Λ ≃ₗ Λ where
  __ := AddEquiv.refl Λ
  preserves_inner' _ _ := rfl

/-- Lattice equivalence relation is symmetric. -/
def symm (f : Λ₁ ≃ₗ Λ₂) : Λ₂ ≃ₗ Λ₁ where
  __ := f.toAddEquiv.symm
  preserves_inner' x y := by
    rw [← f.preserves_inner']
    simp

/-- Lattice equivalence relation is transitive. -/
def trans (f : Λ₁ ≃ₗ Λ₂) (g : Λ₂ ≃ₗ Λ₃) : Λ₁ ≃ₗ Λ₃ where
  __ := f.toAddEquiv.trans g.toAddEquiv
  preserves_inner' x y := by
    rw [← f.preserves_inner', ← g.preserves_inner']
    simp

instance : EquivLike (Λ₁ ≃ₗ Λ₂) Λ₁ Λ₂ where
  coe := fun x => x.toFun
  inv := fun x ↦ x.invFun
  left_inv := fun x ↦ x.left_inv
  right_inv := fun x ↦ x.right_inv
  coe_injective' := by
    intros f g h
    cases f
    simp_all

instance : AddEquivClass (Λ₁ ≃ₗ Λ₂) Λ₁ Λ₂ where
  map_add f a b := f.map_add a b

lemma preserves_inner (f : Λ₁ ≃ₗ Λ₂) (x y : Λ₁) : ⟪f x, f y⟫_ℤ = ⟪x, y⟫_ℤ
↪  :=
  f.preserves_inner' x y

@[ext]
theorem ext {f g : Λ₁ ≃ₗ Λ₂} (h : ∀ x, f x = g x) : f = g :=
  DFunLike.ext f g h

@[simp]
theorem self_trans_symm (f : Λ₁ ≃ₗ Λ₂) : f.trans f.symm = refl Λ₁ :=
  ext f.left_inv

end IntegralLatticeEquiv
```

Mathlib ⟩ *GroupTheory* ⟩ *IntegralLattice* ⟩ *Aut.lean* (New file)

```
/-
Copyright (c) 2024 Erik van der Plas. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Erik van der Plas
-/
import Mathlib.GroupTheory.IntegralLattice.Equiv
import Mathlib.LinearAlgebra.Determinant

variable (Λ : Type*) [IntegralLattice Λ]

/-- The group of integral lattice automorphisms -/
@[reducible]
def IntegralLatticeAut := IntegralLatticeEquiv Λ Λ

namespace IntegralLatticeAut

/-- Integral lattice automorphisms form a group under composition. -/
instance : Group (IntegralLatticeAut Λ) where
  mul f g := IntegralLatticeEquiv.trans g f
  one := IntegralLatticeEquiv.refl Λ
  inv := IntegralLatticeEquiv.symm
  mul_assoc := by intros; rfl
  one_mul := by intros; rfl
  mul_one := by intros; rfl
  mul_left_inv := IntegralLatticeEquiv.self_trans_symm

variable {Λ}

@[simp]
lemma one_apply (x : Λ) : (1 : IntegralLatticeAut Λ) x = x :=
  rfl

@[simp]
lemma mul_apply (f g : IntegralLatticeAut Λ) (x : Λ) : (f * g) x = f (g x)
↪  :=
  rfl

/-- The determinant of an integral lattice automorphism, as a group
↪  homomorphism. -/
noncomputable
def det : IntegralLatticeAut Λ →* ℤ where
  toFun := fun f ↦ LinearMap.det
  ↪  f.toAddEquiv.toAddMonoidHom.toIntLinearMap
  map_one' := LinearMap.det_id
  map_mul' := fun f g ↦ by
    dsimp
    rw [← map_mul]
    rfl

lemma _root_.SignType.cast_inj : Function.Injective SignType.cast := by
  intros a b h
  revert a b
  decide

def _root_.Int.SignHom : ℤ →* SignType := {
  toFun := fun n => SignType.sign n,
  map_one' := rfl
  map_mul' := fun n m => by
    dsimp
    have : Int.sign (n * m) = Int.sign n * Int.sign m := by
      simp
    simp [Int.sign_eq_sign] at this
    norm_cast at this
    rwa [SignType.cast_inj.eq_iff] at this
}

noncomputable
def SignHom : IntegralLatticeAut Λ →* SignType := Int.SignHom.comp det

end IntegralLatticeAut
```

Mathlib 〉 *GroupTheory* 〉 *IntegralLattice* 〉 *Leech* 〉 *Basic.lean* (New file)

```
/-
Copyright (c) 2024 Erik van der Plas. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Erik van der Plas
-/
import Mathlib.GroupTheory.IntegralLattice.Equiv
import Mathlib.LinearAlgebra.Dimension.Finrank

universe u

open IntegralLattice
open FiniteDimensional

/-- The Leech lattice is the unique even, unimodular integral lattice of
↪  rank 24
  such that the norm of any nonzero vector is at least 4.
-/
class LeechLattice (Λ : Type*) extends IntegralLattice Λ where
  (even : IsEven Λ)
  (unimodular : IsUnimodular Λ)
  (rank_eq_24 : finrank ℤ Λ = 24)
  (min_norm : ∀ (x : Λ), x ≠ 0 → ⟪x, x⟫_ℤ ≥ 4)
```

```
namespace LeechLattice

variable (Λ : Type*) [LeechLattice Λ]

theorem unique (Λ₁ Λ₂ : Type*) [LeechLattice Λ₁] [LeechLattice Λ₂]:
  Nonempty (Λ₁ ≃ₗ Λ₂) := sorry

theorem exists_leech : ∃ (Λ : Type u), Nonempty (LeechLattice Λ) := sorry

instance (n : ℕ) : Finite {x: Λ | ⟪x, x⟫_ℤ = n} := sorry
instance (n : ℕ) : Finite {x: Λ // ⟪x, x⟫_ℤ = n} := sorry

-- Lemma's about cardinality of vectors of norms 2, 4, 6 and 8:
lemma card_norm_2 : Nat.card {x: Λ | ⟪x, x⟫_ℤ = 2} = 0 := by
  rw [Nat.card_eq_zero]
  left
  simp
  intro x hx
  by_cases hx0 : x = 0
  · subst x
    simp at hx
  · have := min_norm x hx0
    linarith

lemma card_norm_4 : Nat.card {x: Λ | ⟪x, x⟫_ℤ = 4} = 196560 := sorry
lemma card_norm_6 : Nat.card {x: Λ | ⟪x, x⟫_ℤ = 6} = 16773120 := sorry
lemma card_norm_8 : Nat.card {x: Λ | ⟪x, x⟫_ℤ = 8} = 398034000 := sorry

end LeechLattice
```

Mathlib ⟩ *GroupTheory* ⟩ *IntegralLattice* ⟩ *Leech* ⟩ *Cross.lean* (New file)

```
/-
Copyright (c) 2024 Erik van der Plas. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Erik van der Plas
-/
import Mathlib.GroupTheory.IntegralLattice.Leech.Basic
import Mathlib.GroupTheory.IntegralLattice.Aut
import Mathlib.Analysis.InnerProductSpace.Basic

variable (Λ : Type*) [LeechLattice Λ]

/-- A cross `C` is a subset of the Leech lattice `Λ` consisting of vectors
↪  of norm 8
  such that for each pair `x, y ∈ C` one of the following three conditions
  ↪  holds:
  (1) `x` and `y` are perpendicular, i.e. `⟪x, y⟫_ℤ = 0`,
  (2) `x = y` or
  (3) `x = -y`.
-/
@[ext]
structure Cross where
  (carrier : Set Λ)
  (norm_8 : ∀ x ∈ carrier, ⟪x, x⟫_ℤ = 8)
  (perpendicular : ∀ x ∈ carrier, ∀ y ∈ carrier, x ≠ y → ⟪x, y⟫_ℤ = 0 ∨ x
  ↪  = -y)

namespace Cross

instance : SMul (IntegralLatticeAut Λ) (Cross Λ) where
```

```
  smul f C := {
    carrier := f '' C.carrier,
    norm_8 := by
      intro x h
      simp at h
      rcases h with ⟨y, hy, rfl⟩
      rw [f.preserves_inner]
      apply C.norm_8 y hy
    perpendicular := by
      intro x hx y hy hxy
      simp at hx hy
      rcases hx with ⟨x, hx, rfl⟩
      rcases hy with ⟨y, hy, rfl⟩
      rw [f.preserves_inner]
      have := C.perpendicular x hx y hy
      rw [← map_neg f]
      simp_all [- map_neg]
  }

@[simp]
lemma smul_carrier (f : IntegralLatticeAut Λ) (C : Cross Λ) :
  (f • C).carrier = f '' C.carrier := rfl

instance : MulAction (IntegralLatticeAut Λ) (Cross Λ) where
  one_smul := by
    intro C
    ext x
    simp
  mul_smul := by
    intro f g C
    ext x
    simp

lemma card_cross : Nat.card (Cross Λ) = 8292375 := sorry

lemma card_cross_carrier (C : Cross Λ) : Nat.card C.carrier = 48 := by
  sorry

end Cross
```

Mathlib ⟩ *GroupTheory* ⟩ *SpecificGroups* ⟩ *Conway.lean* (New file)

```
/-
Copyright (c) 2024 Erik van der Plas. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Erik van der Plas
-/
import Mathlib.GroupTheory.IntegralLattice.Leech.Basic
import Mathlib.GroupTheory.IntegralLattice.Leech.Cross
import Mathlib.GroupTheory.Subgroup.Center
import Mathlib.GroupTheory.GroupAction.Primitive
import Mathlib.GroupTheory.Subgroup.Simple
import Mathlib.GroupTheory.GroupAction.Iwasawa

open IntegralLatticeAut

variable (Λ : Type*) [LeechLattice Λ]

-- The Conway-0 group is the automorphism group of the Leech lattice.
-- We use the IntegralLatticeAut structure to construct the variable.
abbrev Conway₀ := IntegralLatticeAut Λ
```

```
-- The Conway-1 group is the Conway-0 group modulo its center.
abbrev Conway₁ := Conway₀ Λ / (Subgroup.center (Conway₀ Λ))

namespace Conway₁

def iwasawa : IwasawaStructure (Conway₀ Λ) (Cross Λ) where
    T := sorry
    is_comm := sorry
    is_conj := sorry
    is_generator := sorry

lemma nontrivial : Nontrivial (Conway₁ Λ) := by
  apply Nontrivial.mk
  let id : Conway₁ Λ := 1
  -- TODO: Exhibit a non-identity element in the Conway-1 group.
  have hnt : ∃ x : Conway₁ Λ, id ≠ x := sorry
  use id

lemma quasipreprimitive : IsQuasipreprimitive (Conway₀ Λ) (Cross Λ) :=
↪  sorry

lemma perfect : commutator (Conway₀ Λ) = ⊤ := sorry

-- TODO: Define the action of the Conway-1 group on the crosses.
-- lemma faithfull : FaithfulSMul (Conway₁ Λ) (Cross Λ) := sorry

-- The Conway-1 group is a simple group.
instance simple : IsSimpleGroup (Conway₀ Λ) := by
  -- TODO: Use the Iwasawa structure by ACL and apply the isSimpleGroup
  ↪  theorem.
  sorry

end Conway₁
```

B IWASAWA PR MODIFICATIONS

Modifications to the Iwasawa pull request by Antoine Chambert-Loir (Chambert-Loir, 2024), compared to its state on 2024-05-28.

Mathlib ⟩ *GroupTheory* ⟩ *GroupAction* ⟩ *Blocks.lean* (Modified file)

```
diff --git a/Mathlib/GroupTheory/GroupAction/Blocks.lean
↪ b/Mathlib/GroupTheory/GroupAction/Blocks.lean
index 9cf09ce89..1333e2e85 100644
--- a/Mathlib/GroupTheory/GroupAction/Blocks.lean
+++ b/Mathlib/GroupTheory/GroupAction/Blocks.lean
@@ -72,7 +72,7 @@ theorem IsPartition.of_orbits :
     exact Set.Nonempty.ne_empty (MulAction.orbit_nonempty a) ha
   intro a; use orbit G a
   constructor
-  · simp only [Set.mem_range_self, mem_orbit_self,
↪ exists_unique_iff_exists, exists_true_left]
+  · simp only [Set.mem_range, exists_apply_eq_apply, mem_orbit_self,
↪ and_self]
   · simp only [Set.mem_range, exists_unique_iff_exists, exists_prop,
   ↪  and_imp, forall_exists_index,
       forall_apply_eq_imp_iff']
     rintro B b (rfl) ha
@@ -416,7 +416,7 @@ theorem IsBlock.isBlockSystem [hGX :
↪ MulAction.IsPretransitive G X]
     use g • B
     constructor
     · simp only [Set.mem_range, exists_apply_eq_apply,
     ↪  exists_unique_iff_exists, exists_true_left]
-      exact hg
+      simp [hg]
     · simp only [Set.mem_range, exists_unique_iff_exists, exists_prop,
     ↪  and_imp, forall_exists_index,
         forall_apply_eq_imp_iff']
       intro B' g' hg' ha
@@ -608,14 +608,13 @@ theorem Setoid.nat_sum {α : Type _} [Finite α] {c :
↪ Set (Set α)} (hc : Setoid
   simp only [Subtype.mk_eq_mk, Subtype.coe_mk]
   apply And.intro _ hab
   refine' ExistsUnique.unique (hc.2 b) _ _
-  simp only [exists_unique_iff_exists, exists_prop]
   exact ⟨hx, ha⟩
-  simp only [exists_unique_iff_exists, exists_prop]
   exact ⟨hy, hb⟩
   -- surjectivity
   intro a
-  obtain ⟨x, ⟨hx, ha : a ∈ x, _⟩, _⟩ := hc.2 a
-  use ⟨⟨x, hx⟩, ⟨a, ha⟩⟩
+  obtain ⟨x, ⟨hx, ha⟩, _⟩ := hc.2 a
+  simp only [Sigma.exists, Subtype.exists, exists_prop, exists_eq_right]
+  use x, hx, ha

 theorem Set.ncard_coe {α : Type*} (s : Set α) :
     s.ncard = Set.ncard (Set.univ : Set (Set.Elem s)) := by
```

Mathlib ⟩ *GroupTheory* ⟩ *GroupAction* ⟩ *Primitive.lean* (Modified file)

```
diff --git a/Mathlib/GroupTheory/GroupAction/Primitive.lean
↪  b/Mathlib/GroupTheory/GroupAction/Primitive.lean
index 75ed120aa..0dc27869b 100644
--- a/Mathlib/GroupTheory/GroupAction/Primitive.lean
+++ b/Mathlib/GroupTheory/GroupAction/Primitive.lean
@@ -472,7 +472,7 @@ theorem _root_.Setoid.IsPartition.ncard_eq_finsum
     simp only [Set.singleton_subset_iff, Set.mem_empty_iff_false,
     ↪  not_false_eq_true, and_true]
     exact ⟨hx.2, hy.2⟩
   · rintro ⟨x, hx⟩
-    obtain ⟨t, ⟨ht, hx', _⟩, _⟩ := hP.2 x
+    obtain ⟨t, ⟨ht, hx'⟩, _⟩ := hP.2 x
     use ⟨⟨t, ht⟩, ⟨x, ⟨hx, hx'⟩⟩⟩

 /-- The target of an equivariant map of large image is preprimitive if
 ↪  the source is -/
```

BIBLIOGRAPHY

Capdeboscq, Inna et al. (Dec. 2023). *The classification of the finite simple groups, number 10*. en. Providence, RI: American Mathematical Society.

Chambert-Loir, Antoine (2024). *The Iwasawa criterion for simplicity*. URL: `https://github.com/leanprover-community/mathlib4/pull/12048`.

Conway, J. H. (1968). "A PERFECT GROUP OF ORDER 8,315,553,613,086,720,000 AND THE SPORADIC SIMPLE GROUPS". In: *Proceedings of the National Academy of Sciences* 61.2, pp. 398–400. DOI: `10.1073/pnas.61.2.398`. eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.61.2.398`. URL: `https://www.pnas.org/doi/abs/10.1073/pnas.61.2.398`.

— (1969). "A Group of Order 8,315,553,613,086,720,000". In: *Bulletin of the London Mathematical Society* 1.1, pp. 79–88. DOI: `https://doi.org/10.1112/blms/1.1.79`. eprint: `https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/blms/1.1.79`. URL: `https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/blms/1.1.79`.

Dummit, D.S. and R.M. Foote (2003). *Abstract Algebra*. Wiley. ISBN: 9780471433347. URL: `https://books.google.nl/books?id=KJDBQgAACAAJ`.

Gorenstein, Daniel, Richard N. Lyons, and Ronald M. Solomon (Nov. 1994). *The classification of the finite simple groups*. Mathematical Surveys and Monographs. Providence, RI: American Mathematical Society.

Moura, Leonardo Mendonça de et al. (2015). "The Lean Theorem Prover (System Description)." In: *CADE*. Ed. by Amy P. Felty and Aart Middeldorp. Vol. 9195. Lecture Notes in Computer Science. Springer, pp. 378–388. ISBN: 978-3-319-21400-9. URL: `http://dblp.uni-trier.de/db/conf/cade/cade2015.html#MouraKADR15`.

The mathlib community (2020). "The Lean mathematical library". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pp. 367–381. DOI: `10.1145/3372885.3373824`. URL: `https://doi.org/10.1145/3372885.3373824`.

Wilson, R. (2009). *The Finite Simple Groups*. Graduate Texts in Mathematics. Springer London. ISBN: 9781848009875. URL: `https://books.google.nl/books?id=lYMAg_Sj7hUC`.