Formalizing Giles Gardam's Disproof of Kaplansky's Unit Conjecture

Siddhartha Gadgil*
Indian Institute of Science
India
gadgil@iisc.ac.in

Anand Rao Tadipatri
Indian Institute of Science Education and Research
India
art71@cam.ac.uk

Abstract

We describe a formalization in Lean 4 of Giles Gardam's disproof of Kaplansky's Unit Conjecture. This makes use of a combination of deductive proving and formally verified computation, using the nature of Lean 4 as a programming language which is also a proof assistant.

Our goal in this work, besides formalization of the specific result, is to show what is possible with the current state of the art and illustrate how it can be achieved. Specifically we illustrate *real time* formalization of an important mathematical result and the *seamless integration* of proofs and computations in Lean 4.

CCS Concepts: • **Theory of computation → Logic and verification**.

Keywords: Automated Theorem Proving; Interactive Theorem Proving; Dependent Type Theory; Lean theorem prover; Kaplansky's conjectures; group rings

ACM Reference Format:
Siddhartha Gadgil and Anand Rao Tadipatri. 2024. Formalizing Giles Gardam's Disproof of Kaplansky's Unit Conjecture. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '24), January 15–16, 2024, London, UK*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3636501.3636947

1 Introduction

A little over two years ago, Giles Gardam disproved [10] a long-standing conjecture, often called the *Kaplansky Unit Conjecture* [24]. Besides the simplicity of the statement and its history, this conjecture was important as it was one of a cluster of related statements with important relations to many areas (including the Whitehead conjecture in topology).

We describe here the formalization of Gardam's disproof in the *Lean 4* theorem prover[1] [33], which was completed roughly a year after the result was announced. We used Lean 4 as a proof assistant, but also took advantage of its being a full-fledged programming language. Indeed, Gardam's result can be viewed as having two components – that a specific group is *torsion-free* (this was classically known) and that certain explicit associated elements in its group ring are (non-trivial) *units* (Gardam discovered these and showed they were units). The former is a mathematical proof, and the latter is a computation, specifically symbolic algebra.

Thus, our work involved making definitions in such a way that results such as torsion-freeness can be proved, while at the same time being effective, i.e., usable in computations.

1.1 Significance of our formalization

We describe some significant features that facilitated of our formalization. We feel this shows what is possible with the current state of the art and illustrates a way in which it can be achieved. While some of these may be novel, others will have been used in other formalizations with similar contexts and needs.

1.1.1 Mathematical choices. Many mathematical choices were crucial in facilitating our formalization. Firstly, Gardam gave several definitions of the group P for which he disproved the Kaplansky Unit conjecture: in terms of matrices, presentations, geometry etc. We believe working with most of these would make formalization significantly harder than the one we chose: a description as a metabelian group.

We formalized metabelian groups in general. These are special instances of *group extensions*. While we used the general theory of group extensions (in terms of *cocycles*), we made use of the special nature of metabelian groups (as a group extension with the kernel and quotient abelian) to simplify the formalization. For the data for metabelian groups we used concrete descriptions of Gardam (and some simple calculations based on these). We believe that using either more abstraction and generality or more concrete

*Both authors contributed equally to this research.

CPP '24, January 15–16, 2024, London, UK

ACM ISBN 979-8-4007-0488-8/24/01
https://doi.org/10.1145/3636501.3636947

[1] Source code in supplementary material

descriptions with specific calculations would have made the formalization significantly harder.

1.1.2 Integration of proofs and computations.

Our proof used the seamless integration of proofs and programs (and even meta-programs) of Lean 4, which facilitates (one style of) formalizations involving computations, which we find attractive and has some advantages (as discussed in Section 2). Specifically, we implemented (straightforward) algorithms for decision problems and used these in proving equality. Further, we used *typeclass inference* to *lift* these algorithms to more complex decision problems, including enumeration over finite sets as well as deciding equality of linear transformations by checking equality on basis elements.

To be able to both decide equality and prove results about free modules (with basis infinite but having decidable equality), we essentially gave two definitions of free modules as quotients of formal sums. One of these, in terms of equality of coefficients, was amenable to be used for decision problems while the other, in terms of elementary moves, was better suited to proving universal properties. We proved these definitions equivalent.

There have of course been many important instances of formalizations of computational proofs (as discussed in 1.2), many which predate this work (and Lean 4) by decades. We do not claim that our approach, or approaches using special features of Lean 4, are superior to the many other ways this can be done.

1.1.3 Real time formalization.

Our result is an instance of *real time formalization* of an important recent mathematical result. There have been a few such formalizations recently (as mentioned in 1.2), but such formalizations are not (yet) commonplace. Real time formalizations of significant mathematical results promise to have an impact on research mathematics by ensuring that a proof and all intermediate lemmas are correct as stated. This not only allows trusting of results, but also ensures that the friction caused to a reader by having slightly wrong statements for intermediate results (something far too common in practice in mathematics) is eliminated. Moreover, the real-time formalization of important mathematical results makes it easier to formalize subsequent work that builds on top of them; with the accumulation of a formal corpus of modern mathematics, the goal of formally verifying *every* new significant result before publication may become more realistic.

1.2 Related work

Formalization of mathematics began with Automath [9] in 1968. Since then many systems for formalization of mathematics have been developed, and significant amounts of mathematics have been formalized in Mizar [35], Coq [1], HOL Light [20], Isabelle [36], Metamath [30], Agda [37], Lean [32], etc. Landmark formalizations include those of the four-colour problem [11], the Jordan curve theorem [18], the prime number theorem [19], the Kepler conjecture [16], the odd order theorem [12], the central theorem of condensed mathematics [44], sphere eversion [28].

1.2.1 Real time formalization.

Most significant formalizations were achieved decades after the original proof was found. However, there are a few cases where an important mathematical result was formalized in real-time (months or a couple of years after the proof was found). These include the formalization of the Cap set conjecture by Dahmen-Hölzl-Lewis [8], and that of the Erdös-Graham density theorem (proved by Thomas Bloom) [2] by Bloom and Bhavik Mehta. More recently, the result of Campos, Griffiths, Morris and Sahasrabudhe announcing an exponential improvement to the upper bound on the diagonal Ramsey numbers [6] was formalized a few months later by Bhavik Mehta [2]. In the Liquid Tensor Experiment [44], mathematics still under development was formalized (but this was a large community effort, so not easy to replicate). The formalization of the proof of the Polynomial Freiman-Ruzsa conjecture [14] [3] was another successful community effort that led to the result being formalized only a few weeks after being announced.

1.2.2 Computations in proofs.

A mix of computation and proofs in formalization has been used in many cases, and many systems have been developed for this. Indeed, in the case of the Kepler conjecture and the four-colour problem, a motivation for formalization was ensuring computations are trustworthy. Some of the work done to combine computations and proofs is the following:

- Facilitating verification of solutions obtained by SAT solvers [21], primality tests [47] etc.
- A method that has been used extensively is to reduce a problem to an equation between computable terms (e.g. that a boolean term is equal to true), and use kernel reduction. This is used extensively in the Mathematical Components library [29] and the SSReflect proof language [13].
- Reducing a problem to an equality between computable terms, extract code, and run it (either with a trusted evaluator or something less trustworthy). This is used in Coq's `vm_compute` or `native_compute`.
- Using special purpose automation to construct proofs of equations (like Coq's `ring` tactic, and the extensive automation in systems like Isabelle and HOL Light).
- Using rigorous numerics to establish results, as in Immler's formalization of the proof of the Smale conjecture [23].

[2] https://xenaproject.wordpress.com/2023/11/04/formalising-modern-research-mathematics-in-real-time/
[3] https://teorth.github.io/pfr/

We use most of the above approaches (adapted to Lean) in our formalization, though not at the scale used in the pioneering works mentioned above. We use Lean's `Decidable` typeclass (see Section 2.2) to bridge between proofs and computations, both reducing to an equation when we use `decide` and running compiled code where we use `native_decide`. We also use Lean's simplifier both directly with the `simp` tactic and through the `aesop` tactic. While Lean has a `ring` tactic, we did not use it in part because this work was done when the ring tactic was not fully ported to Lean 4.

Our use of the `Decidable` typeclass is similar to the approach in small scale reflection, where a decidable predicate can be represented by a program giving a boolean value. Indeed, an instance of the `Decidable` typeclass is in a sense a lift of such a program to one returning a proof of the proposition or its negation. Logically, this is equivalent to a boolean valued program with a proof of correctness. Examples of the use of an approach based on boolean-valued functions are the definition of presentations of finite groups in the Mathematical Components library[4] and the work by Peterfalvi on the Odd Order Theorem [41].

1.3 Outline of the paper

In Section 2 we briefly describe some aspects of Lean 4 and our formalization.

In Section 3 we sketch the mathematical background of Kaplansky's conjecture and Gardam's theorem. We begin by defining and motivating group rings in Section 3.3. We state Kaplansky's conjectures in Section 3.4, and Gardam's theorem in Section 3.5. In Section 3.6 we describe Metabelian groups. The group P used in Gardam's counterexample and the proof that it is torsion-free are briefly described in Section 3.7.

In Section 4 we turn to describing our formalization of Gardam's theorem. We begin with describing the constructions of free modules in Section 4.1 and of group rings in Section 4.2. In Section 4.3 we describe a technique we use: lifting decision problems using enumeration over finite sets and bases, and its implementation. We describe the construction of Metabelian groups in Section 4.4 and the specific group P in Section 4.5. The proof of the result that P is torsion-free is described in Section 4.6. The disproof of the Unit Conjecture, using these ingredients, is sketched in Section 4.7.

Essentially the only source of possible errors in a formalization is a wrong definition used in the statement of the result. To guard against this, we prove some extra results as discussed in Section 4.8. We conclude with some remarks.

2 Lean 4: Foundations, Computations

The Lean Theorem Prover [32] is an interactive theorem prover created by Leonardo de Moura. It has an extensive mathematical library, `mathlib` [29], built by a large community effort, which includes most undergraduate mathematics and some advanced topics.

2.1 Lean Theorem Prover 4

Lean is based on the Calculus of Inductive Constructions (CIC) [39], which is an extension of Martin-Löf Type Theory (MLTT) [27]. The Coq system [1] is also based on CIC and Agda [37] is based on MLTT. Due to the Curry-Howard isomorphism [46], these foundation systems include proofs and programs at the same level, allowing an integration of algorithms and proofs. Lean 4 exploits this fully, being a self-hosting programming language, i.e., a programming language with the primary compiler/interpreter written largely in the language itself (hence with the necessary expressivity and performance) and being a popular proof assistant with a large mathematical library. To the best of our knowledge Idris [3] is the only other major language of this nature, i.e., self-hosting and with proofs as part of the language

Lean, like many other interactive theorem provers, supports various forms of automation. These include *tactics*, *unification*, *simplification*, *term rewriting*, and *typeclass inference*. As described in Section 2.2 we make use of typeclass inference as a bridge to proofs. We also make use of the other forms of automation in our proofs. In particular, we use the `aesop` [26] tactic which combines simplification, unification and user-specified proof rules in a configurable best-first proof search.

We use the seamless integration of proofs, programs and meta-programs in Lean in this work. The use of a common language at all levels has advantages (as with client and server in software): allowing code to be shared, avoiding context switches, and eliminating (often error-prone and tedious) cross-language interfaces. Integration of computation and proofs predates our work, and indeed Lean, by decades. There are many approaches to this, as discussed in Section 1.2. We illustrate one approach, which we find attractive, in this work.

2.2 Typeclasses and Automation

The principal bridge between computations and proofs we use is the `Decidable` typeclass. For a proposition `P`, an instance of `Decidable P` signifies that `P` is a *decidable proposition*, meaning that it can be algorithmically settled in the positive or the negative. Concretely, the `Decidable` typeclass associates to a proposition `P` a type `Decidable P`, terms of which correspond to proofs that `P` is true or proofs that `P` is false. Thus, if for each `a : A`, `P a` is an associated proposition (for example, for each `a : Nat` we can take `P a` to be `a` being an exact square), then a (dependent) function `(a : A) → (Decidable P a)` gives, for each `a : A`, either a proof that `P a` is true or a proof that `P a` is false.

Further, as `Decidable` is a so called *typeclass*, *instances* (terms) of `Decidable` can be inferred using functions with

[4] https://github.com/math-comp/math-comp/blob/master/mathcomp/fingroup/presentation.v

codomain of the form `Decidable P` by *typeclass inference*. In particular, we construct instances using enumeration over finite sets as well as bases of free Abelian groups.

We remark that alternatively *tactics* could be written that bridge between proofs and computations.

3 Mathematical Background

Kaplansky's Unit Conjecture says that the only *units* in the *Group Ring* $k[G]$ of a *torsion-free* group G over a field k are elements of the form $a \cdot g$, where a is a unit in k and g is an element of G.

We first recall some basic definitions and motivate the question.

3.1 Zero-divisors and Units in Rings

If the product of two integers n and m is zero, then (at least) one of them must be zero. On the other hand, if we consider *pairs* of integers with addition and multiplication defined pointwise (i.e., $(a_1, b_1) + (a_2, b_2) = (a_1 + a_2, b_1 + b_2)$ and $(a_1, b_1) \cdot (a_2, b_2) = (a_1 a_2, b_1 b_2)$), then we have $(1, 0) \cdot (0, 1) = (0, 0)$ but $(1, 0) \neq (0, 0)$ and $(0, 1) \neq (0, 0)$. We say $(0, 1)$ and $(1, 0)$ are *zero-divisors* in the *ring* (we recall the definition below) $\mathbb{Z}^2$ of pairs of integers, as their product is the *zero element* $(0, 0)$ in $\mathbb{Z}^2$ but neither of them is equal to the zero element.

A ring is a set with two operations, addition and multiplication, satisfying some standard axioms: both operations are associative; addition is commutative; there is an additive identity; every element has an additive inverse; multiplication distributes over addition. We say the ring R has zero divisors if there are non-zero elements $a, b \in R$ such that $a \cdot b = 0$, where 0 refers to the additive identity (zero element) of the ring. We saw above that the ring of integers has no zero divisors, but the ring of pairs of integers does.

Note that multiplication in a ring need not be commutative. Important examples of rings, where multiplication is not commutative, are *rings of matrices*. Namely, for any n the set of $n \times n$ real-valued matrices is a ring, with addition and multiplication defined in the usual way. In this ring zero divisors are (non-zero) matrices which are not invertible, as follows from basic Linear Algebra.

An element a in a ring is said to be a *unit* if it has an inverse, i.e., there is an element b such that $a \cdot b = 1$ and $b \cdot a = 1$. In the ring of integers the units are 1 and -1. In the ring of $n \times n$ matrices the units are the invertible matrices.

A ring is a *field* if every non-zero element is a unit, i.e., has a multiplicative inverse. For example, real numbers form a field, but the ring of integers does not.

3.2 Polynomials and Laurent Polynomials

The set of polynomials $P(x) = \sum_{i=0}^{n} a_i x^i$ over the real numbers (i.e., with a_i real) form a ring. We can see that this ring has no zero divisors and the units in this ring are the non-zero constant polynomials, i.e., the polynomials of the form a with $a \neq 0$. To see this, observe that if $P(x) = \sum_{i=0}^{n} a_i x^i$ and $Q(x) = \sum_{j=0}^{m} b_j x^j$ are polynomials, with $a_n \neq 0$ and $b_m \neq 0$, then the coefficient of x^{n+m} is $a_n b_m \neq 0$. Thus, if $P(x)Q(x) = 0$, then $P(x) = 0$ or $Q(x) = 0$. Similarly, if $P(x)Q(x) = 1$, then $P(x)$ and $Q(x)$ are both constant polynomials.

Note that all the above is true for the ring of polynomials $P(x) = \sum_{i=0}^{n} a_i x^i$ over any field k (i.e., with $a_i \in k$), not just the real numbers.

Laurent polynomials are like polynomials except allowing negative powers of the variable x. Thus, a Laurent polynomial is of the form $P(x) = \sum_{i=n'}^{n} a_i x^i$ with $n, n' \in \mathbb{Z}$ and $n' \leq n$. We again take a_i to be elements of a field k.

We see that the ring of Laurent polynomials over a field also has no zero divisors. However, units are not just constant polynomials. Namely, for $a \neq 0$ and $n \in \mathbb{Z}$, the monomial ax^n is a unit as $(ax^n) \cdot (a^{-1}x^{-n}) = 1$. We can see that these are the only units, and that there are no zero divisors. Namely, consider polynomials $P(x) = \sum_{i=n'}^{n} a_i x^i$ and $Q(x) = \sum_{i=m'}^{m} b_i x^i$ with $n, n', m, m' \in \mathbb{Z}$, $n' \leq n$, $m' \leq m$, and a_n, $a_{n'}$, b_m and $b_{m'}$ all not zero. Then the coefficients of $n + m$ and $n' + m'$ are $a_n b_m \neq 0$ and $a'_n b'_m \neq 0$. Thus, $P(x)Q(x) \neq 0$. Further, if $P(x)Q(x) = 1$, then we must have $n' + m' = n + m$, so $n = n'$ and $m = m'$. It follows that $P(x)$ is of the form ax^n with $a \neq 0$.

We can more generally consider (Laurent) polynomials with coefficients in any ring R. However, the above statements no longer hold in general.

3.3 Group Rings

Group Rings are analogues of Laurent polynomials with powers x^n replaced by elements in a group G. For this, it is convenient to view Laurent polynomials a little differently. Observe that instead of using the order on powers of x, we can define Laurent polynomials as (formal) sums $\sum_{i=1}^{k} a_i x^{n_i}$ with $n_i \in \mathbb{Z}$ for all i and with all the powers n_i distinct. The multiplication is defined as the bilinear extension of that of $x^n \cdot x^m = x^{n+m}$, i.e.,

$$(\sum_{i=1}^{k} a_i x^{n_i}) \cdot (\sum_{j=1}^{l} b_j x^{m_j}) = \sum_{i=1}^{k} \sum_{j=1}^{l} (a_i b_j) x^{n_i + m_j},$$

with the right-hand side assumed to be simplified by grouping together terms corresponding to the same powers of x.

Fix a group G and a ring R. The group ring $R[G]$ is the set of formal sums $\sum_{i=1}^{n} a_i g_i$ with $a_i \in R$ and $g_i \in G$. We assume that the g_i are distinct (otherwise we can combine coefficients) and that all the coefficients are non-zero. Further, two sums related by reordering the terms are considered equal.

Formally, the group ring is the free R-module with basis G (a *free module* is the analogue of a vector space with a given basis, allowing coefficients to be in a ring instead of a field). The multiplication in $R[G]$ is the bilinear extension of that of G, i.e.,

$$(\sum_{i=1}^{k} a_i g_i) \cdot (\sum_{j=1}^{l} b_j h_j) = \sum_{i=1}^{k} \sum_{j=1}^{l} (a_i b_j)(g_i h_j).$$

The right-hand side is assumed to be simplified grouping together terms corresponding to the same group elements (just as we simplify polynomials in the variable x by grouping together monomials of the same degree).

Observe that the group ring $R[\mathbb{Z}]$ is precisely the ring of Laurent polynomials with coefficients in R, with $\mathbb{Z}$ viewed as the infinite cyclic multiplicative group $\langle x \rangle$, i.e., with $n \in \mathbb{Z}$ identified with the element $x^n \in \langle x \rangle$.

Group rings occur naturally and are important in many areas of mathematics. For instance, if a vector space (or more generally an R-module) admits a linear action by a group, then it has the structure of a module over the group ring $R[G]$. In particular chains and co-chains (over a ring R) on covers of simplicial complexes (more generally cell complexes) form modules over the group ring $R[G]$ for the group G of deck transformations. Many important topological invariants, such as the Alexander polynomials and Reidemeister torsion, are constructed from such $R[G]$-modules.

3.4 Kaplansky's conjectures

Consider group rings over a field k. Suppose g is a torsion element in a group G, i.e., we have $g^n = e$ for some $n > 1$ but $g \neq e$. In contrast to the case of Laurent polynomials (i.e. $G = \mathbb{Z}$), we do have zero-divisors in $k[G]$ as

$$(1 - g)(1 + g + \cdots + g^{n-1}) = 1 - g^n = 0.$$

Similarly, for all but finitely many groups with torsion, $k[G]$ has non-trivial units, i.e., units not of the form $a \cdot g$ [22].

The Kaplansky conjectures assert that we do not have non-trivial units or zero divisors if G is **torsion-free** (i.e., the only element with finite order is the identity) and k is a field.

Thus, suppose k is a field, and G is a torsion-free group. We have the following conjectures, the first of which was formulated earlier by Higman [22].

Conjecture 3.1 (Kaplansky's Unit Conjecture)**.** *Given elements α, β in $k[G]$, if $\alpha\beta = 1$ then there exists $a \in k$, $g \in G$ such that $\alpha = a \cdot g$.*

Conjecture 3.2 (Kaplansky's Zero Divisors Conjecture)**.** *Given elements α, β in $k[G]$, if $\alpha\beta = 0$ then $\alpha = 0$ or $\beta = 0$.*

Beyond the simplicity and generality of the statements, these are attractive because of their similarity to many important mathematical questions. For example, in understanding a basic question in topology – the relation between homotopy type and homeomorphism type – Whitehead introduced an intermediate relation, *simple homotopy type*, and showed that simple homotopy types are classified by elements of the *Whitehead Group*. Again, groups with torsion can have non-trivial Whitehead groups, while the **Whitehead Conjecture** postulates that torsion-free groups have trivial Whitehead groups. Indeed, there are direct relations too between the Whitehead group and Kaplansky's conjectures.

3.5 Gardam's theorem

In [10], Gardam proved the following theorem, showing that Kaplansky's Unit conjecture is false.

Theorem 3.3 (Gardam)**.** *Let $\mathbb{F}_2$ be the field with 2 elements. There exists a torsion-free group P and elements α, α' in the group ring $\mathbb{F}_2[P]$ such that $\alpha \cdot \alpha' = 1$ and α and α' are not of the form $a \cdot g$ with $a \neq 0 \in \mathbb{F}_2$, $g \in P$.*

Specifically, Gardam considered a certain (well studied) group P, called the *Promislow* [42] or *Hantzsche–Wendt* group. This was known to be torsion-free. Gardam showed that there are elements $\alpha, \alpha' \in \mathbb{F}_2[P]$ with $\alpha \cdot \alpha' = 1$ and with α not of the form $a \cdot g$. The elements were discovered by a computer search using SAT solvers (i.e., programs that solve the *Boolean satisfiability problem*, i.e., equation with Boolean variables and constraints built from these using logical operations such as $\wedge$, $\vee$, and $\neg$) and were given explicitly by Gardam.

Gardam gives many descriptions of the group P. For instance, for those familiar with Geometric topology, a topological description is as the fundamental group of a 3-manifold M obtained by gluing two copies of the orientable twisted I-bundle over the Klein bottle along their boundaries.

Given our motivation of simultaneously computing with as well as proving theorems about the group, the description of P that was most convenient to us was as a *Metabelian* group. We next recall Metabelian groups.

3.6 Metabelian Groups

A *Metabelian Group* is a group G with an *abelian normal subgroup* K so that the quotient $Q = G/K$ is also abelian. The simplest such groups are the products $K \times Q$. However, for fixed K and Q, there are in general other corresponding metabelian groups G, as we see with following examples.

- The permutation group $G = S_3$ is a metabelian group, with abelian normal subgroup $K \cong \mathbb{Z}/3$ generated by the cycle σ and quotient $Q \cong \mathbb{Z}/2$.
- Let $G = \mathbb{Z}$ and $K = 2\mathbb{Z}$ be the subgroup of even integers. Then $Q = G/K \cong \mathbb{Z}/2$. This gives $\mathbb{Z}$ a description as a metabelian group with $K \cong \mathbb{Z}$ and $Q \cong \mathbb{Z}/2$.

The two examples illustrate the two ways in which G can fail to be a product. In a precise sense, we see that these are the only two ways. Thus, we can determine all metabelian groups G corresponding to fixed K and Q in terms of some explicit data. We remark that this is a special case of the so called *group extension* problem.

Firstly, observe that as a *set* we can always identify G with $K \times Q$ by choosing a *section* $s : Q \to G$, i.e., a function $s : Q \to G$ such that $q(s(x)) = x$ for all x in Q, where $q : G \to Q$ is the quotient homomorphism. Every element $g \in G$ is then uniquely of the form $g = k \cdot s(q)$ with $k \in K$ and $q \in Q$.

In the case of the permutation group we can choose s to be a homomorphism by mapping the generator 1_2 of $\mathbb{Z}/2$ to a transposition τ. Thus Q can be identified with a subgroup of G and s regarded as the inclusion (and omitted from the notation). However, an element of the group $K \subset G$ does not in general commute with an element of $Q \subset G$, so G is not a product of K and Q. Instead, we can write $kqk'q' = kk'^q qq'$, where $k'^q = qk'q^{-1}$ is the result of conjugation. Conjugation gives a *group action* of $Q \subset G$ on K. Conversely, given an arbitrary group action $Q \times K \to K$, $(q, k) \mapsto q \cdot k$, we can define a metabelian group with underlying set $K \times Q$ and multiplication given by the formula

$$(k, q) \cdot (k', q') = (k + q \cdot k', q + q').$$

This is a special case of the familiar *semi-direct product* construction.

Indeed, as K is abelian, for all metabelian groups we have an action of Q on K, given by $q \cdot k = s(q) \cdot k \cdot s(q)^{-1}$, which does not depend on the choice of section s. This action is part of the data determining a metabelian group. Suppose henceforth we have fixed K, Q and a group action of Q on K.

The second example illustrates that the group action is not enough to determine G; indeed, as G is abelian the action is trivial, but G is not the product of K and Q. The reason for this is we cannot choose the section $s : Q \to G$ to be a homomorphism; the image $s(1_2)$ must be an odd number $2k + 1$, and while $1_2 + 1_2 = 0$, $s(1_2) + s(1_2) = 4k + 2 \neq 0$ (but $s(0) = 0$ for a homomorphism).

We can measure the extent to which s is not a homomorphism by considering, for elements $q, q' \in Q$, the quantity $c(q, q') := s(qq') \cdot (s(q)s(q'))^{-1}$. As s is a section, we can deduce that in fact $c(q, q') \in K \subset G$, so c gives a function $c : Q \times Q \to K$.

The associativity of G implies a certain condition on the function c, called the *cocycle condition*. Functions $c : Q \times Q \to K$ satisfying this condition are called *cocycles*. Conversely, given a (group action and) a cocycle, we can define a group structure on $G = K \times Q$ by

$$(k, q) \cdot (k', q') = (k + q \cdot k' + c(q, q'), q + q').$$

Indeed all metabelian groups are of this form. Thus, an arbitrary metabelian group is determined by K, Q, an action of Q on K and a cocyle.

We remark that even if the group G is a (semi-)direct product, an arbitrary section $s : Q \to G$ need not be a homomorphism (indeed most sections are not homomorphisms). Thus, the cocycle c need not be zero for a typical section. However, note that if s is a section and $\varphi : Q \to K$ is a function, then $s' : q \mapsto \varphi(q)s(q)$ is also a section, and indeed all sections are of this form. It is easy to compute the cocycle c' corresponding to s' in terms of c and φ, namely c' differs from c by a so called *coboundary*. Thus, metabelian groups are determined by K, Q, an action of Q on K, and a cocyle up to a coboundary. A cocycle up to a coboundary is an element of a group called $\mathrm{Ext}(Q, K)$ (for extension), which is also the so called group cohomology $H^1(Q, K)$.

3.7 The Group P and its Torsion Freeness

The group P is a Metabelian group with kernel $K = \mathbb{Z}^3$ and quotient $Q = \mathbb{Z}/2 \times \mathbb{Z}/2$. Gardam gives an explicit description of the action and the cocycle which determine P. This allows us to explicitly (without much effort) work with elements in P. In particular, we can prove the (well known) result that P is torsion free.

Proposition 3.4. *The group P is torsion-free.*

Sketch of proof. Let $g \in P$ and $n > 1$ be such that $g^n = e$. It follows that $(g^2)^n = (g^n)^2 = e$.

Using the explicit description of the group P, we see that $g^2 \in K \cong \mathbb{Z}^3$. As $\mathbb{Z}^3$ is torsion-free and $(g^2)^n$ is trivial (and $n > 1$), it follows that g^2 is trivial.

Finally, using the explicit description of the group P again, we deduce that $g^2 = e \implies g = e$. Thus $g^n = e \implies g = e$, i.e., P is torsion-free. □

The self-contained and explicit nature of this proof makes it straightforward to formalize. We state precisely the formulae for g^2 in terms of g in the various cases (depending on the Q-component of g). This lets us deduce both that $g^2 \in K$ and that $g^2 = e \implies g = e$.

4 Formalization: proofs, enumeration, computation

The formalization of Gardam's theorem involved two largely independent components, the construction of group rings with decidable equality and the construction of the group P and the proof that it is torsion-free. In this section we first sketch the construction of group rings. We then sketch some typeclass-based methods used, in particular for enumeration, to construct and prove the properties of the group P. We then sketch the construction and torsion-freeness of P. Finally we see that the ingredients can be put together to disprove the Unit Conjecture. Figure 1 shows the modules and their direct imports.

The documentation, with links to the source code, is available in the supplementary material, and we give links relative to the `build/doc` folder. The best way to browse the documentation is to run a static web server, for example to run `python3 -m http.server` from the 'build/docs' directory. An overview is at ./UnitConjecture.html. The package

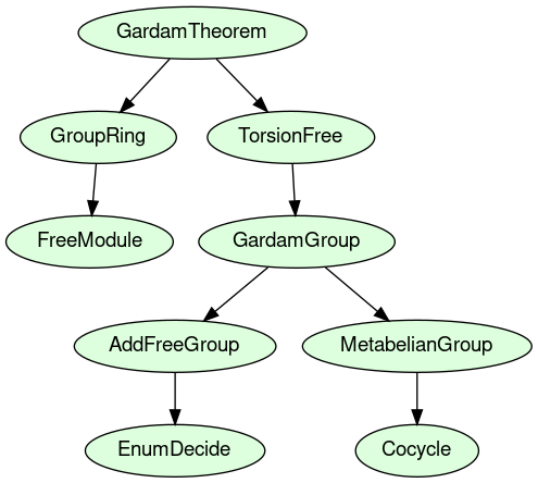

Figure 1. (direct) module dependencies

`UnitConjecture` contains all the modules[5], which can also be navigated by following imports.

Henceforth, **show** or **prove** mean **formally prove in Lean 4.**

4.1 Free Modules

Our construction of Free Modules[6] must facilitate two goals. We need to construct the group ring $\mathbb{F}_2[P]$ and also be able to decide (with proof) when two elements of $\mathbb{F}_2[P]$ are equal. As mentioned in 2.2, the latter is encoded in the `Decidable` typeclass; allowing typeclass inference which we use extensively.

A group ring $R[G]$ is a free R-module with an additional structure, the product. Further the product is a bilinear extension of a function given on the basis G of the free R-module. Thus, our goal is to construct free R-modules with basis X, including when X is infinite, so that both of the following are possible (assuming decidable equality in R and in X).

- We can decide when two elements of the free R-module are equal.
- We can define R-module homomorphisms given values on the basis X.

The first is a computational goal and the second a conceptual one. To achieve both, we make two constructions, and essentially prove that they are equivalent (we actually formally prove only those parts of the equivalence that we need). Both definitions are as quotients of *formal sums*. We first sketch the first construction, which is the definition used, and decidable equality using this. We then sketch the second construction and its equivalence to the first.

[5]We link to the relevant files in footnotes in the first paragraph of the section.

[6]./UnitConjecture/FreeModule.html

4.1.1 Formal sums, Coordinates and Free modules.

Fix a ring R and a type X, with both R and X having decidable equality. A *formal sum* is an expression of the form $\sum_{i=1}^{n} a_i x_i$ where $a_i \in R$ and $x_i \in X$ for $1 \leq i \leq n$. We encode these as equivalence classes (under a quotient we define) of lists of pairs `(a_i, x_i)`, i.e., terms with type `List (R × X)`.

Given a formal sum $s = \sum_{i=1}^{n} a_i x_i$, we can associate to it coordinates as a function $\chi_s : X \to R$, with the coordinate of an element x_0 of X being the sum of the coefficients a_i corresponding to indices i with $x_i = x_0$. Observe that only finitely many coordinates are non-zero (but we do not formally state or prove this). Note that to define the coordinate function χ_s as a (computable) function, we need decidability of equality in X.

We define two formal sums s_1 and s_2 to be equivalent if $\chi_{s_1} = \chi_{s_2}$. It is easy to show that this is an equivalence relation. The free module $R[X]$ is defined as the quotient of the formal sums of X by this equivalence relation.

4.1.2 Supports and Decidable equality.

Observe that if X is infinite (as it is in our case), we cannot check equality of coordinate functions by checking at all values. Thus, we need some other way to check equality of coordinates. We do this by relating coordinates to supports.

We define the support $\mathrm{supp}(s)$ of the formal sum $s = \sum_{i=1}^{n} a_i x_i$ to be the list consisting of the elements x_i (this is a coarse notion of support, ignoring that coefficients may be zero and/or may cancel). We show that for $x_0 \in X$, if $\chi_s(x_0) \neq 0$ then $x_0 \in \mathrm{supp}(s)$ (stated in Listing 1).

```
theorem nonzero_coord_in_support
  (s : FormalSum R X) : ∀ x : X, 0 ≠ s.coords x →
  x ∈ s.support := ...
```

Listing 1. Non-zero coordinates in support

It follows immediately that for formal sums s_1 and s_2, $\chi_{s_1} = \chi_{s_2}$ if and only if $\chi_{s_1}(x) = \chi_{s_2}(x)$ for all $x \in \mathrm{supp}(s_1)$ and also for all $x \in \mathrm{supp}(s_2)$.

We are thus reduced to deciding when two functions with values in R are equal on members of a list in X. This is easily done using induction (using the assumption that we can decide equality in R). Thus, we can decide when formal sums are equivalent.

With a little work, we can deduce decidability of equality in the quotient $R[X]$ from decidability of equivalence of formal sums, using basic theorems about quotients that are part of Lean.

4.1.3 Elementary moves and Universal properties.

To construct multiplication in the Group Ring, we need to extend functions from X to R-modules to $R[X]$-module homomorphisms. We do not formally define R-modules or prove such an extension result, but we implicitly make the definitions and prove the results involved and use them in the group ring construction.

Given a function $f : X \to N$, with N an R-module, f clearly extends to formal sums by $f(\sum_{i=1}^{n} a_i x_i) = \sum_{i=1}^{n} a_i f(x_i)$. What needs to be shown is that if formal sums are equivalent then they map to the same element. This is not so easy when X is infinite.

We instead give a second description of free modules. Namely, we define a relation on formal sums as given by elementary moves:

- if the first term has zero coefficient, it is deleted.
- if the first two terms are of the form `(a, x)` and `(b, x)` for some `x : X`, they are replaced by a single term `(a + b, x)`.
- the first two terms are exchanged.
- a term is prepended to two formal sums related by an elementary move.

Note that this is not an equivalence relation. Nevertheless, in Lean, the quotient is defined, and is the quotient by the equivalence relation generated by the elementary moves.

It is straightforward to show that coordinates are unchanged by elementary moves. The key result, which takes some work, is to show that two formal sums with the same coordinate function are related by a sequence of elementary moves, i.e., have equal images in the quotient by elementary moves.

This in turn depends on the technical result that if a formal sum s has $a_0 := \chi_s(x_0) \neq 0$ for some $x_0 \in X$, then s is related by a sequence of elementary moves to a formal sum of the form $a_0 x_0 + t$, with the number of terms in t less than the number in s, as is stated in Listing 2. This is proved by well-founded recursion, and the main result is also deduced from this by well-founded recursion.

```
theorem nonzero_coeff_has_complement  (x₀ : X)(s :
  FormalSum R X) :
  0 ≠ s.coords x₀ →
  (∃ ys : FormalSum R X,
    (((s.coords x₀, x₀) :: ys) ≃ s) ∧ (List.length
  ys < s.length)) := ...
```

Listing 2. Complements in formal sums

The relations between moves and coordinates show that the equivalence relation generated by moves is the same as that given by having the same coordinate functions. In practice, we use the consequence that a function on formal sums is constant on equivalence classes (corresponding to equal coordinates), and hence well-defined on the Free Module, if and only if it is invariant under elementary moves. This can be conveniently used by using the induction tactic for an elementary move.

4.2 Group Rings

Using the construction and properties of Free Modules, we construct the Group Ring $R[G]$[7]. First, we define multiplication on formal sums recursively in two steps, first recursively defining multiplication of a formal sum on the right by a *monomial* $a \cdot g$, and then recursively defining multiplication of formal sums.

Various properties are proved by (iterated) induction. These allow us to define the product on $R[G]$ by showing invariance under elementary moves. This is also done in two steps, where in the first the first term is a formal sum.

Further, we show that there is a ring structure on the group ring (statement in Listing 3).

```
instance : Ring (FreeModule R G) := ...
```

Listing 3. Group Ring as a Ring

4.3 Lifting Decisions using Enumeration, Bases

As the group $Q = \mathbb{Z}/2 \times \mathbb{Z}/2$ is finite, deciding whether the cocycle condition holds for a function $c : Q \to Q \to K$ can be done by enumeration and checking for each triple of elements of Q. We automate such enumerations using a typeclass `DecideForall`[8]. This captures the property of Q that if $p : Q \to Prop$ is a property of elements of Q such that each $p(q)$ is decidable for each $q : Q$, so is the proposition $\forall x : Q, p(x)$.

By recursion we prove that for the type `Fin n` (consisting of the natural numbers below `n` with the operations corresponding to $\mathbb{Z}/n\mathbb{Z}$) we have an *instance* of `DecideForAll`. This means we can decide the proposition `∀ x : Fin n, p(x)` if we can decide `p(x)` for each `x`. This is stated along with the definition of the typeclass in Listing 4.

```
class DecideForall (α : Type _) where
  decideForall (p : α → Prop) [DecidablePred p]:
      Decidable (∀ x : α, p x)

instance {k: Nat} : DecideForall (Fin k) := ...
```

Listing 4. DecideForall

This can be used iteratively automatically. An illustration (and test) is in Listing 5.

```
theorem Zmod3.assoc :
  ∀ x y z : Fin 3, (x + y) + z = x + (y + z) := by
    decide
```

Listing 5. Associativity by enumeration

Further, we construct instances of `DecideForAll` of products and sums given those for components. This lets us infer an instance for $Q = \mathbb{Z}/2 \times \mathbb{Z}/2$.

Further, in the case of *finitely-generated free abelian groups*, we can define homomorphisms by specifying their values

[7] ./UnitConjecture/GroupRing.html
[8] ./UnitConjecture/EnumDecide.html

on the basis, and we can decide equality of homomorphisms by enumeration of basis elements. Once more we define a typeclass, `AddFreeGroup` – corresponding to an abelian group A being free with basis a type X. We construct instances for $\mathbb{Z}$ with basis a singleton (represented by `Unit` in Lean), and for direct products of free abelian groups, with basis the direct sum of the bases of the factors. Note that we do not assume that the basis X is contained in A (it is usually not), instead we have an inclusion (as a field of the typeclass).

We show that if A is a free abelian goup whose basis X has an instance of `DecideForall`, then equality of homomorphisms on A is decidable (provided the codomain has decidable equality). We also show that functions from X to abelian groups extend uniquely to homomorphisms on A.

4.4 Constructing Metabelian Groups

As recalled in Section 3.6 a Metabelian group[9] is determined by

1. A pair of abelian groups Q and K, which we represent as *Additive groups*.
2. An action of Q on K by automorphisms.
3. A cocycle $c : Q \times Q \to K$.

This data is bundled together in the code as the `Cocycle` typeclass (Listing 6), with the cocyle explicitly encoded into the type.

```
class Cocycle {Q K : Type _} [AddGroup Q]
   [AddGroup K] (c : Q → Q → K) where
  /-- An action of the quotient on the kernel by
    automorphisms. -/
  α : Q → (K →+ K)
  /-- A typeclass instance for the action by
    automorphisms. -/
  autAct : AutAction α
  /-- The value of the cocycle is zero when its
    inputs are zero, as a convention. -/
  cocycle_zero : c 0 0 = (0 : K)
  /-- The *cocycle condition*. -/
  cocycle_condition : ∀ q q' q'' : Q,
      c q q' + c (q + q') q'' =
          q +ᵥ c q' q'' + c q (q' + q'')
```

Listing 6. The `Cocycle` typeclass

Given an action and the cocycle, defining the multiplication, the identity element and the inverse operation on $G := K \times Q$ are straightforward. Some work is needed to show that this is indeed a group, especially that the group action property and the cocycle condition imply associativity. By integrating with the `Aesop` automation tool for Lean 4 [26] and proving some auxilliary lemmas about actions and cocycles, we are able to produce succinct proofs of these results.

[9] ./UnitConjecture/MetabelianGroup.html

As a check, we also prove that the group resulting from this construction lies in the center of the short exact sequence with K as the kernel and Q as the quotient.

4.5 The Group P

As we saw in Section 3.7, the group P is a metabelian group[10] with the kernel group K isomorphic to $\mathbb{Z}^3$ and quotient group Q isomorphic to the Klein Four group $\mathbb{Z}/2 \times \mathbb{Z}/2$. We use the general construction of metabelian groups as in Section 4.4. For this, we need to define the action of Q on K by automorphisms, the cocycle $c : Q \times Q \to K$ (actually the curried form $c : Q \to Q \to K$), and show that these indeed form a group action and a cocycle. The group structures on $\mathbb{Z}$ and `Fin 2` (i.e., the group $\mathbb{Z}/2$) are already known to Lean, and the product group structures on the types $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ and `Fin 2 × Fin 2` are automatically inferred.

The group action is a family of homomorphisms $K \to K$, given by $k \mapsto q \cdot k$, associated to elements q of the quotient group Q. As $K = \mathbb{Z}^3$ is a free abelian group we can decide equality of homomorphisms by enumeration of basis elements. Indeed, the code in Listing 7 gives the verification of the group action property. We see that this involves typeclass derivation and decision problems, which in turn are based on enumeration on finite sets and on bases.

```
local infixr:100 " × " => AddMonoidHom.prodMap

/-- The action of Q on K by automorphisms.
The action can be given a component-wise description
    in terms of id and neg, the identity and
    negation homomorphisms. -/
@[aesop norm unfold (rule_sets [P]), reducible]
def action : Q → (K →+ K)
  | .e =>  .id ℤ  ×  .id ℤ  ×  .id ℤ
  | .a =>  .id ℤ  ×  neg ℤ  ×  neg ℤ
  | .b =>  neg ℤ  ×  .id ℤ  ×  neg ℤ
  | .c =>  neg ℤ  ×  neg ℤ  ×  .id ℤ

/-- A verification that the above action is indeed
    an action by automorphisms.
  This is done automatically with the machinery of
    decidable equality of homomorphisms on free
    groups. -/
instance : AutAction action :=
  { id_action := rfl
    compatibility := by decide }
```

Listing 7. Group action

The derivation of the cocycle condition (see Listing 8) is similar.

```
def cocycle : Q → Q → K
  | a , a  => x
  | a , c  => x
```

[10] ./UnitConjecture/GardamTheorem.html

```
  | b , b  => y
  | c , b  => -y
  | c , c  => z
  | b , c  => -x + -z
  | c , a  => -y + z
  | b , a  => -x + y + -z
  | _ , _  => 0

instance P_cocycle : Cocycle cocycle :=
  { α := action
    autAct := inferInstance
    cocycle_zero := rfl
    cocycle_condition := by decide }
```

Listing 8. Cocycle condition

We remark that the `by decide` for the cocycle condition is an enumeration over 64 cases. One could alternatively enumerate using tactics like `cases` and `split`, but these depend on the product structure of the enumeration, while our typeclass-based approach used only effective finiteness of the enumeration.

Another consequence of defining the group on the concrete underlying type $\mathbb{Z}\times\mathbb{Z}\times\mathbb{Z}\times$ `Fin 2 × Fin 2` is that it is easy to compute whether two given elements of the group are equal. This gives the construction of P a strong computational advantages over other descriptions, such as in terms of generators and relations, where elements can be represented in more than one way and equality of elements cannot be checked directly. The Lean theorem prover is able to infer that equality on P is decidable from the fact that it is the product of K and Q (both of which are in turn iterated products of types with decidable equality):

```
instance : DecidableEq P := inferInstanceAs <|
    DecidableEq (K × Q)
```

Listing 9. Decidable Equality on P

As a confirmation that our constructions and type inference have worked correctly, we check that the multiplication on P is given by the appropriate explicit formula.

```
theorem mul (k k' : K) (q q' : Q) :
    (k, q) * (k', q') =
        (k + action q k' + cocycle q q', q + q') :=
  rfl
```

Listing 10. Multiplication on P

The type $P = K\times Q$ also has an instance of a product group structure coming from `mathlib`. We mark the instances of the metabelian group structure and the associated multiplication operation on P with high priority to make these the default. For general hygiene, we also mark these instances as scoped to limit them to their designated namespace.

4.6 Torsion freeness of P

Torsion freeness[11] follows the mathematical sketch in Section 3.7. We begin by showing, for an element $g \in P$, that g^2 is in the kernel of the metabelian extension describing P, and is given by an explicit formula (see Listing 11). We prove this in Lean by matching on the structure of the group Q and invoking the `aesop` tactic with the appropriate configuration.

```
/-- The function taking an element of P to its
    square, which lies in the kernel K. -/
def P.sq : P → K
  | ((p, q, r), .e) => (p + p, q + q, r + r)
  | ((_, q, _), .b) => (0, q + q + 1, 0)
  | ((p, _, _), .a) => (p + p + 1, 0, 0)
  | ((_, _, r), .c) => (0, 0, r + r + 1)

/-- A proof that the function sq indeed takes an
    element of P to its square in K. -/
theorem sq_square : ∀ g : P, g * g = (P.sq g, .e)
  | ((p, q, r), x) =>
    match x with
    | .e | .a | .b | .c => by ...
```

Listing 11. Square of torsion element

The result stated in Listing 12 shows that the group K in the construction of P is a torsion-free additive group, using typeclass inference. As $K = \mathbb{Z}^3$, the facts that $\mathbb{Z}$ is torsion-free and the products of torsion-free groups are torsion-free are sufficient for Lean to infer this.

```
/-- The kernel Z^3 is torsion-free. -/
instance K.torsionFree : AddTorsionFree K :=
    inferInstance
```

Listing 12. The kernel is torsion-free

Thus, g^2 is contained in a torsion-free group. However, it can be deduced from general group theory that if g is a torsion element, then g^2 is also a torsion element. It follows that if g is a torsion element, then $g^2 = e$. On the other hand the explicit formula for g^2 lets us conclude that if $g^2 = e$ then $g = e$. The proof of this in Listing 13 is again by cases and automation.

```
/-- The only element of P with order dividing 2 is
    the identity. -/
theorem square_free : ∀ {g : P}, g * g = (1 : P) →
    g = (1 : P)
  | ((p, q, r), x) => by
    match x with
    | .e => ...
    | .a | .b | .c => ...
```

Listing 13. The group P has no elements of order two

It follows that if g is a torsion element, then $g = e$, i.e. P is torsion-free. This argument is summarised in Listing 14.

[11] ./UnitConjecture/TorsionFree.html

```
/-- P is torsion-free. -/
instance P.torsionFree : TorsionFree P where
  torsionFree := ...
```

Listing 14. P is torsion-free

The explicit description of the group P is once again convenient here for proving these facts. While the above statements can be quite difficult to prove using an arbitrary description of the group P, in this case it merely amounts to a proof by four cases on Q followed by reductions and simplifications that can be automated. The kernel subgroup also happens to be explicitly identified in the construction of P as a metabelian group, which allows for a convenient description of the function 11 that takes an element of P to its square.

4.7 The Unit Conjecture

With the group P constructed, its torsion-freeness proved, and the group ring constructed, Gardam's disproof of the Unit conjecture is a simple verification[12]. Note that P is a product of types with decidable equality, so has decidable equality – this is part of Lean. Similarly the field $\mathbb{F}_2$ has decidable equality. It follows that $\mathbb{F}_2[P]$ has decidable equality.

As mentioned earlier, Gardam has given an explicit formula for a non-trivial unit α and its inverse.

```
  /-- The non-trivial unit α. -/
  def α : 𝔽₂[P] := p  +  q * a  +  r * b  +  s * a *
    b

  /-- The inverse α' of the non-trivial unit α -/
  def α' : 𝔽₂[P] := p'  +  q' * a  +  r' * b  +  s' *
     a * b
  [-
```

Listing 15. The unit

```
-]
```

We define a *trivial element* of a free module to be one that has a unique non-zero coordinate. To show that α is non-trivial we produce distinct coordinates that are non-zero 16.

```
/-- The definition of an element of a free module
    being trivial, i.e., of the form kx for x : X
    and k nonzero.-/
def trivialNonZeroElem {R X : Type _} [Ring R]
    [DecidableEq X] (a : FreeModule R X) : Prop :=
∃! x : X, FreeModule.coordinates x a ≠ 0

/-- A proof that the unit is non-trivial. -/
theorem α_nonTrivial : ¬ (trivialNonZeroElem α) :=
    by ...
```

Listing 16. Non-trivial unit

[12] ./UnitConjecture/GardamGroup.html

We simply use the `native_decide` tactic to show, using Lean's evaluation, that α is a unit in P.

```
/-- A proof of the existence of a non-trivial unit
    in F_2[P]. -/
def Counterexample :
  {u : (𝔽₂[P])ˣ //
  ¬(trivialNonZeroElem u.val)} :=
  ⟨⟨α, α', by native_decide, by native_decide⟩,
  α_nonTrivial⟩
```

Listing 17. The counterexample

Putting these together, we conclude Gardam's theorem.

```
/-- The statement of Kaplansky's Unit Conjecture:
The only units in a group ring, when the group is
    torsion-free and the ring is a field, are the
    trivial units. -/
def UnitConjecture : Prop :=
  ∀ {F : Type _} [Field F] [DecidableEq F]
  {G : Type _} [Group G] [DecidableEq G]
    [TorsionFree G],
    ∀ u : (F[G])ˣ, trivialNonZeroElem (u : F[G])

/-- Giles Gardam's result - Kaplansky's Unit
    Conjecture is false. -/
theorem GardamTheorem : ¬ UnitConjecture :=
   fun conjecture => Counterexample.prop <|
    conjecture (F := 𝔽₂) (G := P) Counterexample.val
```

Listing 18. Gardam's theorem

4.8 Extra theorems for verification

Essentially the only main way in which a formally proved result can be wrong is if the statement, or some definition involved in the statement, is wrong. One can greatly reduce the chance of this happening by proving extra results about the definitions which are not used in the main result. This approach was taken in the Liquid Tensor Experiment [44] where *test* theorems were proved. Definitions from mathlib have effectively been very well tested by use in many results.

A relatively complex definition we make is that of the *Group Ring* of a group, which depends on our construction of *Free Modules*. To test the correctness of this definition, we prove a few results that are not needed in the main theorem (the injectivity results were suggested by Gardam).

- The group ring of a group is a ring, in particular the product is associative.
- The inclusion map $G \longrightarrow R[G]$ given by $g \mapsto g \cdot 1$ is a monoid homomorphism. Further, if $R = k$ is a field, then the map $g \mapsto g \cdot 1$ is injective.
- The inclusion map $R \longrightarrow R[G]$ given by $r \mapsto r \cdot 1$ is a ring homomorphism and is injective.

Note that these are highly non-generic properties. Further, injectivity will fail for degenerate structures. So it is highly

unlikely that we could prove these if there were errors in our definition. An additional check on the definition of Free Modules is that we defined two different quotients and proved that they were equivalent. Further, we proved a universal property for the quotient.

Another test (which we first accidentally carried out) is to slightly vary the formulas for Gardam's units, and check that these are not units.

Our statement involved the definition of *non-trivial* units. While this is a relatively simple definition, it involved our construction of the group ring and not just its structure. To avoid errors arising from this, we proved that our definition was equivalent to the standard one.

Observe that the correctness of our formalization of a disproof Kaplansky's unit conjecture does not need our group P to coincide with the group Gardam considered, as we have a self-contained construction of P, proof that it is a group and a proof that it is torsion-free. It would be bizarre if our construction of P were wrong but still gave a counterexample to Kaplansky's conjecture (and we are confident that we did in fact consider the correct group), but this will not affect the correctness of the formalization.

On the other hand, our construction of Metabelian groups may be useful elsewhere. As a test for its correctness we prove that a Metabelian group satisfies the defining short exact sequence (our main theorem is a much stronger test of the correctness of our construction).

4.9 Concluding Remarks

We hope that such a blend of computation and proving will be useful for many results, both formal proofs and proved algorithms. While computations have been used many times in formal proofs, they have not been used much in conjunction with mathlib, which is a large collection of formalised mathematics.

This formalization was fairly quick, suggesting that the gap between formalization and human exposition is not too large, at least for certain results (including, like Gardam's theorem, some important mathematical results).

References

[1] Yves Bertot. 2008. A short presentation of Coq. In *International Conference on Theorem Proving in Higher Order Logics*. Springer, 12–16.

[2] Thomas F Bloom. 2021. On a density conjecture about unit fractions. *arXiv preprint arXiv:2112.03726* (2021).

[3] Edwin Brady. 2013. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of functional programming* 23, 5 (2013), 552–593.

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

[5] Kevin Buzzard, Johan Commelin, and Patrick Massot. 2020. Formalising perfectoid spaces. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 299–312.

[6] Marcelo Campos, Simon Griffiths, Robert Morris, and Julian Sahasrabudhe. 2023. An exponential improvement for diagonal Ramsey. arXiv:2303.09521 [math.CO]

[7] William Carter. 2014. New examples of torsion-free non-unique product groups. *Journal of Group Theory* 17, 3 (2014), 445–464.

[8] Sander R Dahmen, Johannes Hölzl, and Robert Y Lewis. 2019. Formalizing the solution to the cap set problem. *arXiv preprint arXiv:1907.01449* (2019).

[9] Nicolaas Govert De Bruijn. 1994. A survey of the project AUTOMATH. In *Studies in Logic and the Foundations of Mathematics*. Vol. 133. Elsevier, 141–161.

[10] Giles Gardam. 2021. A counterexample to the unit conjecture for group rings. *Annals of Mathematics* 194, 3 (2021), 967–979.

[11] Georges Gonthier et al. 2008. Formal proof–the four-color theorem. *Notices of the AMS* 55, 11 (2008), 1382–1393.

[12] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, et al. 2013. A machine-checked proof of the odd order theorem. In *International conference on interactive theorem proving*. Springer, 163–179.

[13] Georges Gonthier and Assia Mahboubi. 2010. An introduction to small scale reflection in Coq. *Journal of Formalized Reasoning* 3, 2 (Jan. 2010), 95–152. https://doi.org/10.6092/issn.1972-5787/1979

[14] W. T. Gowers, Ben Green, Freddie Manners, and Terence Tao. 2023. On a conjecture of Marton. arXiv:2311.05762 [math.NT]

[15] Benjamin Grégoire and Laurent Théry. 2006. A purely functional library for modular arithmetic and its application to certifying large prime numbers. In *International Joint Conference on Automated Reasoning*. Springer, 423–437.

[16] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. 2017. A formal proof of the Kepler conjecture. In *Forum of mathematics, Pi*, Vol. 5. Cambridge University Press.

[17] Thomas C Hales. 2005. A proof of the Kepler conjecture. *Annals of mathematics* (2005), 1065–1185.

[18] Thomas C Hales. 2007. The Jordan curve theorem, formally and informally. *The American Mathematical Monthly* 114, 10 (2007), 882–894.

[19] John Harrison. 2009. Formalizing an analytic proof of the prime number theorem. *Journal of Automated Reasoning* 43, 3 (2009), 243–261.

[20] John Harrison. 2009. HOL light: An overview. In *International Conference on Theorem Proving in Higher Order Logics*. Springer, 60–66.

[21] Marijn Heule, Warren Hunt, Matt Kaufmann, and Nathan Wetzler. 2017. Efficient, verified checking of propositional proofs. In *International Conference on Interactive Theorem Proving*. Springer, 269–284.

[22] G Higman. 1940. The Units of Group Rings. Londres. *Proc. London Math. Soc* 46 (1940).

[23] F Immler. 2018. A Verified ODE Solver and the Lorenz Attractor. *J Autom Reasoning* 61 (2018), 73–111.

[24] Irving Kaplansky. 1970. "Problems in the theory of rings" revisited. *The American Mathematical Monthly* 77, 5 (1970), 445–454.

[25] Matt Kaufmann and J Strother Moore. 1996. ACL2: An industrial strength version of Nqthm. In *Proceedings of 11th Annual Conference on Computer Assurance. COMPASS'96*. IEEE, 23–34.

[26] Jannis Limperg and Asta Halkjær From. 2023. Aesop: White-Box Best-First Proof Search for Lean. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 253–266.

[27] Per Martin-Löf and Giovanni Sambin. 1984. *Intuitionistic type theory*. Vol. 9. Bibliopolis Naples.

[28] Patrick Massot, Floris van Doorn, and Oliver Nash. 2022. Formalising the *h*-principle and sphere eversion. *arXiv preprint arXiv:2210.07746* (2022).

[29] The mathlib Community. 2020. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New Orleans, LA, USA) *(CPP 2020)*. Association for Computing Machinery, New York, NY, USA, 367–381. https://doi.org/10.1145/3372885.3373824

[30] Norman Megill and David A Wheeler. 2019. *Metamath: a computer language for mathematical proofs*. Lulu. com.

[31] Dale Miller and Gopalan Nadathur. 2012. *Programming with higher-order logic*. Cambridge University Press.

[32] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean theorem prover (system description). In *International Conference on Automated Deduction*. Springer, 378–388.

[33] Leonardo de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*. Springer, 625–635.

[34] Alan G Murray. 2021. More counterexamples to the unit conjecture for group rings. *arXiv preprint arXiv:2106.02147* (2021).

[35] Adam Naumowicz and Artur Korniłowicz. 2009. A brief overview of Mizar. In *International Conference on Theorem Proving in Higher Order Logics*. Springer, 67–72.

[36] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer.

[37] U Norell. 2009. Dependently Typed Programming in Agda, Advanced Functional Programming, P. Koopman, R. Plasmeijer, and D. Swierstra (Eds.), Vol. 5832 of LNCS.

[38] Donald S Passman. 2011. *The algebraic structure of group rings*. Courier Corporation.

[39] Christine Paulin-Mohring. 2015. Introduction to the calculus of inductive constructions.

[40] Lawrence C Paulson. 1989. The foundation of a generic theorem prover. *Journal of Automated Reasoning* 5, 3 (1989), 363–397.

[41] T. Peterfalvi. 2000. *Character Theory for the Odd Order Theorem*. Cambridge University Press. https://doi.org/10.1017/CBO9780511565861

[42] S David Promislow. 1988. A simple example of a torsion-free, non unique product group. *Bulletin of the London Mathematical Society* 20, 4 (1988), 302–304.

[43] Robert Sandling. 1981. Graham Higman's thesis "Units in group rings". In *Integral representations and applications*. Springer, 93–116.

[44] Peter Scholze. 2022. Liquid tensor experiment. *Experimental Mathematics* 31, 2 (2022), 349–354.

[45] Daniel Selsam, Sebastian Ullrich, and Leonardo de Moura. 2020. Tabled typeclass resolution. *arXiv preprint arXiv:2001.04301* (2020).

[46] Morten Heine B Sørensen and Pawe l Urzyczyn. 1998. Curry-Howard Isomorphism. *Univ. of Copenhagen, Univ. of Warsaw* (1998).

[47] Laurent Théry and Guillaume Hanrot. 2007. Primality proving with elliptic curves. In *International Conference on Theorem Proving in Higher Order Logics*. Springer, 319–333.

[48] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study.

Received 2023-09-19; accepted 2023-11-25