Attempting to prove the First Isomorphism Theorem

Formalising Mathematics CW1

Leo Jones, CID: 02036984

February 2024

1 Introduction

Many hours of this project were dedicated to deciding what to do. I immediately wanted to prove something from Logic (MATH60132), but this was a third year module and will have to wait for a later coursework. The logic in first year consists of parts of IUM (which was mainly in lean already) and some concepts in Analysis I.

After reviewing my options, I wanted to prove something about countable sets, as this was one of my favourite sections of first year. In hindsight I should have realised that ”conceptual” proofs - based more on intuition than syntax - don’t always translate easily into lean (at least not at my skill level). I searched through Mathlib to find the notation I needed to use and couldn’t make sense of most of it. It had *finite* and *encodable* as prerequisites and I struggled to replicate these definitions meaningfully in my own code.

I had avoided group theory after seeing how many of the past projects used it but it was starting to look like the best option so I started by trying out all the groups/subgroups problem sheets [1] to familiarise myself. I’d forgotten how elementary first year groups was, and many of the theorems/problem sheet questions followed quite directly from the questions in the course repository.

I tried creating some definitions from scratch (cosets, cyclic groups, etc.) but still hadn’t found anything interesting to do. I’d seen that a past project had tried and failed to prove the First Isomorphism Theorem (easily the most interesting thing from the year one groups content) so I was hesitant to do it myself but eventually I gave in to the temptation. Here is my attempt.

2 Classical Proof

The First Isomorphism theorem is mentioned without proof in first year group theory, so we are following the proof from the second year Groups and Rings module [2].

Theorem 2.1 (**1st Isomorphism Theorem**)**.** *Let $\phi : G \to H$ be a group homomorphism. The map $gKer(\phi) \mapsto \phi(g)$ is an isomorphism of groups:*

$$G/Ker(\phi) \xrightarrow{\sim} \phi(G)$$

Proof. We know that $Ker(\phi)$ is a normal subgroup of G, so $G/Ker(\phi)$ is a group.

The rule $gKer(\phi) \mapsto \phi(g)$ is a function $G/Ker(\phi) \to \phi(G)$, because ϕ is constant on each coset $gKer(\phi)$.

Let us check that this function is a homomorphism. We know that the product of $g_1 Ker(\phi)$ and $g_2 Ker(\phi)$ equals $g_1 g_2 Ker(\phi)$. But the image of $g_1 g_2 Ker(\phi)$ is $\phi(g_1 g_2) = \phi(g_1)\phi(g_2)$, which is the product of the images of $g_1 Ker(\phi)$ and $g_2 Ker(\phi)$, so we are fine.

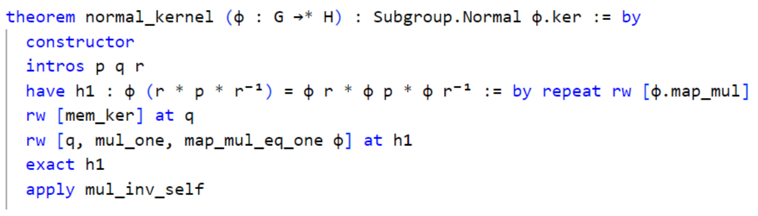
Our function is visibly surjective onto $Im(\phi)$. It is also injective. To check this it is enough to prove that its kernel is the unit element of $G/Ker(\phi)$, which is the trivial coset $Ker(\phi)$. If a coset $gKer(\phi)$ goes to e_H, we have $\phi(g) = e_H$, but then $g \in Ker(\phi)$, hence $gKer(\phi) = Ker(\phi)$q.

To conclude, -$G/Ker(\phi) \to \phi(G)$ is a bijective homomorphism, hence an isomorphism.

□

3 Proof in Lean

Using the predefined Normal structure and Kernel definition for groups, I started by proving that the Kernel is a normal subgroup. This was achievable with the tactics and skills I had already used on the problem sheets.



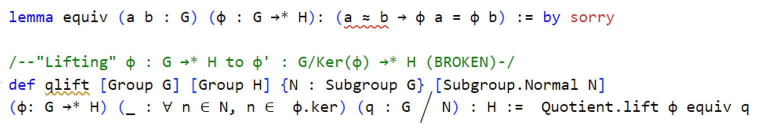
```
theorem normal_kernel (φ : G →* H) : Subgroup.Normal φ.ker := by
  constructor
  intros p q r
  have h1 : φ (r * p * r⁻¹) = φ r * φ p * φ r⁻¹ := by repeat rw [φ.map_mul]
  rw [mem_ker] at q
  rw [q, mul_one, map_mul_eq_one φ] at h1
  exact h1
  apply mul_inv_self
```

I found using the *have* tactic (accompanied by sorry) useful in constructing the skeleton of the proof, proving it in smaller steps and then removing the *have*'s (where needed) to simplify.

I then had to construct the required function, after which I could prove the desired properties when defining it as an isomorphism.

A paper I read [3] directed me to a function *Quotient.lift*. When "quotienting" the domain by an equivalence relation, we can "lift" our function to be defined on the equivalence classes of elements sent to the same value in the range. In our context, elements in the kernel (this is the equivalence relation) are all sent to the identity, therefore ϕ respects this relation and we can lift ϕ to be defined on the quotient $G/Ker(\phi)$.

More clearly, in Lean:



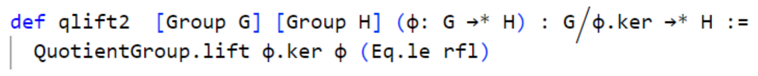
```
lemma equiv (a b : G) (ϕ : G →* H): (a ≈ b → ϕ a = ϕ b) := by sorry

/--"Lifting" ϕ : G →* H to ϕ' : G/Ker(ϕ) →* H (BROKEN)-/
def qlift [Group G] [Group H] {N : Subgroup G} [Subgroup.Normal N]
(ϕ: G →* H) (_ : ∀ n ∈ N, n ∈  ϕ.ker) (q : G ⧸ N) : H :=  Quotient.lift ϕ equiv q
```

Quotient.lift takes our function ϕ, our choice of quotient ($q : G/Ker(\phi)$), and a proof that the equivalence relation (implied by quotient, but given by _) is constant on ϕ, *equiv*.

I got stuck here for a while trying to get this proof to work. It's trivial that two elements in the Kernel will get sent to the same value in the domain, but Lean has a way of making really trivial statements very difficult to prove sometimes. My main issue was $\approx$ was not compatible with group elements, but *Quotient.lift* would not accept a different type of equivalence. I even tried to define an equivalence relation from scratch, but this did not help.

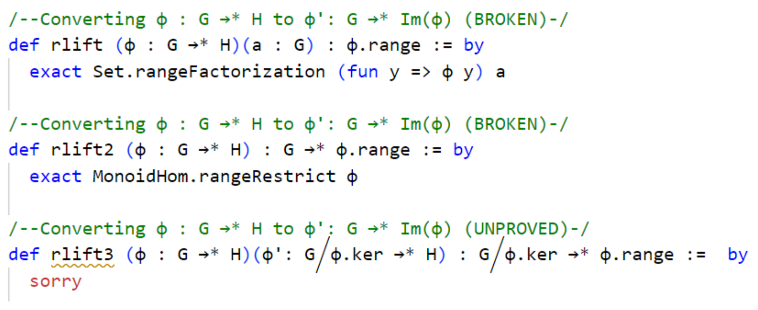
Eventually I discovered a more specific instance *QuotientGroup.lift* in the problem sheets, with the condition instead being that your normal group is contained in the kernel, which was a lot simpler to prove (as in our case they are equal).



```
def qlift2  [Group G] [Group H] (ϕ: G →* H) : G⧸ϕ.ker →* H :=
  QuotientGroup.lift ϕ.ker ϕ (Eq.le rfl)
```

I then had to figure out a way to convert my function $\phi : G/Ker(\phi) \to *H$ to $\phi' : G/Ker(\phi) \to *Im(\phi)$. I decided to do this as a separate (image) function and then combine them when defining the isomorphism. I tried this a couple ways but kept getting Type errors that I didn't know how to fix.

I figured out that the cause of these errors was that I was passing ϕ through the quotient function first, so that when it was passed into the image function, it would be taking the image of the new function $\phi' : G/Ker(\phi) \rightarrow *H$ which would then not match with our desired isomorphism. The only way I could think of to fix this was passing two functions into the image function so that it was not overwritten. This worked for the isomorphism in terms of types, but I unfortunately could not prove it in my definition.

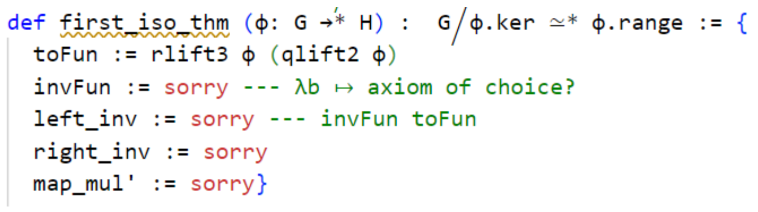


```
/--Converting ϕ : G →* H to ϕ': G →* Im(ϕ) (BROKEN)-/
def rlift (ϕ : G →* H)(a : G) : ϕ.range := by
  exact Set.rangeFactorization (fun y => ϕ y) a

/--Converting ϕ : G →* H to ϕ': G →* Im(ϕ) (BROKEN)-/
def rlift2 (ϕ : G →* H) : G →* ϕ.range := by
  exact MonoidHom.rangeRestrict ϕ

/--Converting ϕ : G →* H to ϕ': G →* Im(ϕ) (UNPROVED)-/
def rlift3 (ϕ : G →* H)(ϕ': G ⧸ ϕ.ker →* H) : G ⧸ ϕ.ker →* ϕ.range :=  by
  sorry
```

The last section of my code is defining the required isomorphism. I did not have time to finish the other properties. I wasn't sure whether to do *invFun* using the preimage of ϕ or possibly by using the inverse of *QuotientGroup.mk'* and then our original homomorphism.



```
def first_iso_thm (ϕ: G →* H) :  G ⧸ ϕ.ker ≃* ϕ.range := {
  toFun := rlift3 ϕ (qlift2 ϕ)
  invFun := sorry --- λb ↦ axiom of choice?
  left_inv := sorry --- invFun toFun
  right_inv := sorry
  map_mul' := sorry}
```

4 Conclusion

It was unfortunate that I ran out of time on this project, as it felt I spent a lot of time trying different things or fixing small errors. I think the large amount of troubleshooting time was partly due to being new to Lean, and having to pick up the syntax and theory (and partly due to the nature of programming). It was very frustrating at times, but also rewarding when you finally figured out something you'd been stuck on for hours.

I feel far more confident in Lean than a week ago. I've picked up some new tactics, figured out what the error messages are trying to tell me (most of the time), and have acquired skills in scouring the Mathlib library. Hopefully I'll have time to pick this project back up in the future (with the skills to complete it) or maybe even build on it for the next coursework.

References

1. Buzzard K. Formalising Mathematics. Lecture notes for MATH60007/70007/97069. Imperial College London. Available from: `https://github.com/ImperialCollegeLondon/formalising-mathematics-2024`
2. Skorobogatov AN. Groups and Rings. Lecture notes for MATH50005. Imperial College London
3. Rowett M. Groups Theory in Lean. Available from: `https://tqft.net/web/notes/load.php?name=students/20180219-MitchRowett-ASC-report-on-Lean`