Formalising Mathematics MATH70040
Coursework 1

My Original Plan

When deciding what to do for this project I considered a few options. The one I considered the most, before landing on my actual idea, was the proof that a number x is divisible by 9 if and only if the sum of digits of x is divisible by 9. After some thought and research, I realised this wouldn't be something accessible for me to prove on Lean due to the exact format of the proof. Having not seen digits before, I was dubious to commit to a whole proof revolving around them. Following careful consideration, I decided to attempt to state/prove the First Isomorphism Theorem.

I then split this theorem into 5 parts:

1) Complete all definitions and proofs that these definitions are groups
2) Prove that the kernel is normal
3) Prove that the kernel is trivial if and only if the homomorphism is injective
4) Stating and understanding the quotient groups and maps
5) State/Prove the First Isomorphism Theorem

The Start of the Project

Once I had my idea set in stone, I knew my difficulty would be in getting to a point where I could state the First Isomorphism Theorem. When I started my coding, I decided to start with stating all the definitions I would need and proving that some of these definitions were indeed groups.

The first real problem that I encountered was after I'd already spent 2 days on the coding and my entire page was red with errors. I believed, quite irrationally, that if I ignored the errors they would maybe disappear, solve themselves or in the worst-case scenario, I would come back to them all later. This proved to be my undoing when at one point I had over 20 error messages on the page. Asking questions, and collaboration are highly encouraged so that we can learn so, when I finally realised, I couldn't solve this by myself, I came to Kevin with a problem that was error number 14 out of 20. Upon his advice, I discerned that I couldn't ignore my errors and I spent the rest of the day trying to fix these errors.

In the beginning the main type of error I was facing were errors in Types. In written Mathematics it does no harm to restate further along that G is a group as it once was before. However, what I didn't know was that Lean only lets you define G as a type once, without sending out error messages.

The first thing I defined was a group homomorphism, this is already defined in the data sets which I imported but I defined this anyway as we would be using them a lot during the coding.

```
19    class group_hom_1 (f : G → H): Prop :=
20    (map_mul' (a b : G) : f (a * b) = f a * f b)
```

I then moved onto defining the image, map, co-map, and kernel and showing that defined from a homomorphism f from G to H, image and map are subgroups of H and co-map and kernel are subgroups of G. These are all defined intuitively with the most important being the kernel. I will now explain this specific code in detail.

Explaining my Code

```
62  /- Here I am defining the kernel and showing it is a subgroup of G-/
63  definition ker_subgroup (f : G →* H): subgroup G :=
64    {carrier := {g : G | f g = 1 },
65    one_mem' := begin simp, end,
66    mul_mem' :=
67    begin
68    simp,
69    intros j k m n,
70    have h:= f.map_mul j k,
71    rw m at h,
72    rw n at h,
73    norm_num at h,
74    apply h,
75    end,
76    inv_mem' := begin simp,
77    end,
78  }
```

To prove ker_subgroup was a group we needed to make sure the 3 axioms that Lean defines for a group are satisfied. This is one_mem' (that the identity is in ker_subgroup), mul_mem' (that if a and b are in ker_subgroup so is a*b) and lastly inv_mem' (that if a is in ker_subgroup so is a^{-1}). These were all solved using the tactics above with one_mem' and inv_mem' being solved by simp. Similar approaches were used for the definitions and proofs for map, co-map, and image. I found this part of the coding very surprising. Once I'd managed to state the definitions, proving them was not only rewarding but incredibly interesting. It took me a while to get the hang of the operations, but I remained persistent with my solutions and eventually saw the 'goals accomplished' sign.

Proving the Kernel is Normal

I now moved onto step 2 of my code, and I needed to prove the Kernel is Normal. I reached quite an important part of my own process during this section of my work. I realised, firstly how difficult I personally find it to ask for help and secondly, how much I needed to ask for help. After spending hours trying to find a way to deconstruct my statement so I could try and solve it definitionally, I did ask for help. So instead of being stuck with (ker_subgroup f).normal, I wanted to prove that:

```
⊢ ∀ (n : G), n ∈ ker_subgroup f → ∀ (g : G), g * n * g⁻¹ ∈ ker_subgroup f
```

I managed to do this by using the constructor tactic, one I hadn't heard of before. I also learnt about using the "hint" tactic in Lean. I already knew about the Library Search tactic,

but having these extra tactics helped a lot with my code further down the line and solved a lot of my initial problems at each stage.

By this point in my code, I'd started to understand the importance of the error messages and had begun to try and read them. This was much harder than I thought I would be, and it really tested my knowledge on parts of the code I thought I was confident on. Beginning to comprehend the error code was a big turning point in learning Lean for me and led me trying methodical tactics to solve the problem as opposed to applying random ones.

After proving the kernel is normal, I went onto define injective, surjective, left inverse and right inverse. I also defined the trivial subgroup and proved this was a subgroup of G. Despite mathematically defining and stating theorems being the easiest part of the maths, this seemed to me to be the opposite case in Lean. I originally defined G as a set.

```
def trivial_subgroup : set G := {1}
```

Whilst in theory this was much simpler to state, it meant when I got to the next section on showing that when the kernel is trivial the homomorphism if injective, the lemma I was trying to state was giving me errors as the kernel is a subgroup, but the trivial subgroup was defined to be a set. This problem meant I had to go back and change my definition of a trivial subgroup, into something that would be the same Type as the kernel. Hence the trivial subgroup became:

```
117  definition trivial_subgroup [group G]: subgroup G :=
118  {carrier := {x | x = 1 },
119    one_mem' := begin simp, end,
120    mul_mem' := begin simp,intros m n h j, rw h,rw j, simp, end,
121    inv_mem' := begin simp, end,
122  }
```

This left me with a similar definition to the ones I defined initially and a similar proof to the kernel being a subgroup.

The Kernel is Trivial if and only if the Homomorphism is Injective

Instead of stating this as an if and only if statement, I broke this statement and the respective proofs up into 2 separate parts. My first part was

```
132  lemma inj_then_kernel_eq_trivial (f: G →* H) (a x y :G)
133  (h : injective f) : ker_subgroup f = trivial_subgroup :=
```

Working in this direction I didn't have many problems, but I was surprised at how long it took me to solve things that were definitionally equal but not syntactically equal. For example, in normal mathematics it's very easy to show that if an element is in the trivial subgroup, that element is in the kernel. Whereas in Lean we need to break it down into this element in the trivial subgroup must be the identity, and then show this is in the kernel. I sometimes reached blocks in my coding when I was trying to prove something was true, that to a mathematician is so obviously true.

In the opposite direction, I worded this slightly differently as I had some problems with proving injectivity with the hypothesis ker_subgroup = trivial_subgroup. So I changed the hypothesis to

```
166    lemma kernel_eq_trivial_then_inj (f: G →* H) (x y :G)
167    (h : ∀(x y : G), (x*y⁻¹) ∈ ker_subgroup f ↔ x*y⁻¹ = 1) : injective f :=
```

I had a few more issues with this direction than with my previous direction. This is due to a problem I had when originally stating this lemma.

```
lemma kernel_eq_trivial_then_inj (f: G →* H) (x y :G)
(h : (x*y⁻¹) ∈ ker_subgroup f ↔ x*y⁻¹ = 1) : injective f :=
```

The above was my original lemma. With some help, I managed to solve my problem. I originally had defined x and y to be specific values of G, when I needed this to be true of all x and y in G. The reason I couldn't prove my original lemma was because it was a false statement. Once I changes the lemma to include the for all x and y in G, I could work on the solution and eventually finish the proof with lots of hypothesises stated and proved along the way.

After the completion of this if and only if, I created two last definitions – bijective and isomorphism. This is so that once I got to the point of proving The First Isomorphism Theorem, as I'd already shown the kernel was normal, I just had to show there was an isomorphism.

Understanding Quotient Groups

This section was the one that took me the longest to do the least and caused the most problems. After letting N be a normal subgroup, it took me a long time to realise there was already information in the mathlib on quotient groups. I spent a long time trying to create my own quotient group based on equivalence relations, this was incredibly hard, and I got nowhere after a very long time. I reached a point where there was so much that I didn't understand that I couldn't ask the right questions to understand it.

This lead me the biggest lesson I learnt from this project; I learnt to start small and build my way up. I learnt to identify exactly what it was that I didn't understand and how to ask questions on that. I would work backwards through my code making sure I made a note of what I wasn't understanding. This led me to a point where I could ask better directed questions and help people help me better. A piece of advice I was given at the start of my coding was to ask good questions to receive helpful answers. This wasn't something I'd thought about before, but it's something I've learnt from this course that I plan to apply elsewhere.

The first thing I needed to understand were the two predefined homomorphisms in the quotient group library. These are

```
/- the group homomorphism from `G` to `G / N` -/
example : G →* G / N := quotient_group.mk' N
```

```
/- The inbuilt homomorphism is `G ⧸ N →* H` -/
example : G ⧸ N →* H := quotient_group.lift N φ hφ
```

The diagram below shows how these two inbuilt homomorphisms work to show an alternative method to get from G to H.

I then stated and completed a proof that for all g in the kernel ϕ(g) = 1. This was all the coding that I managed to complete. I reached problems when thinking about how we define the image and how it would be possible to create a function from G/N to Im ϕ. This caused me a lot of problems and in the end, I realised that I'd taken on too much of a challenge for this first project having only been coding for a few weeks.

Conclusion

I didn't quite manage to get to the last part of my original plan. However, I did manage to complete everything before that without having to sorry my code. This is something that surprised me. As I've mentioned throughout this report, the statements and definitions were the hardest parts for me to get to grips with due to my struggles reading the error codes and understanding the Types. This meant that during my proofs it felt like a reward for stating the lemma. This motivated me to persevere and complete all my proofs.

Whilst I am still learning Lean, I initially thought it would be easier to write about an area of maths that I know very well. What I learnt was how different we as people look at a problem than a computer does. A computer is 100% logic and there are no hand wavey nonmathematical elements. It made me realise a lot about topics I learnt in first year that since then I've taken to granted. I had to take a step back and look at quotients from a new perspective and a new way to define them.

I can't say that I solved all my problems during this project however, I am a lot more confident using Lean than I was 2 weeks ago. I constantly surprised myself by how motivated I was to complete each section and to state each lemma, the 100% logic at times was my worst enemy and at others was my best friend, but above all this it sparked intense happiness and pride. When I complete a question in my normal mathematical life, I feel proud, but there is always some doubt in the back of my mind. Did I omit a small detail? With Lean this wasn't something I worried about, once I'd proved my lemmas or statements, I knew with 100% certainty that I'd completed them, this gave me an intense sense of satisfaction, and this is what motivated me through this project.

Even though I didn’t complete what I set out to do, I faced many troubles along the way (much more than I anticipated) and I surprised myself by pushing through. Instead of a statement and proof of the first isomorphism theorem, you have the code for a lot of the steps in that direction and I hope in my own time to eventually complete this statement and proof.