Formalising the classification of groups of order pq

Peiran Wu

University of St Andrews

25 September 2024

University of
St Andrews

About Lean

- Open-source functional programming language
- Interactive theorem prover – used to confirm the correctness of mathematical proofs
- Accompanied by "Mathlib", a community-maintained open-source mathematical library
- Google DeepMind's AlphaProof (together with AlphaGeometry 2)
- Lots of gaps in the group theory section of Mathlib
- Demo – how to read Lean code

Groups of order pq

- Last year, there was a study group on Lean in St Andrews.
- As an exercise, I wanted to classify the groups of order 4, but someone beat me to it.
- Just before Christmas, I formalised the classification of groups of order 6 (140 LoC, not golfed) and felt it wouldn't be too hard to generalise.
- Scott Harper and I started working together on the classification of groups of order pq.

Let p and q be positive prime numbers with $p \leqslant q$. Let G be a group of order pq. Then exactly one of the following holds:

(1) $G \cong \mathrm{C}_{pq}$.

(2) $p = q$ *and* $G \cong \mathrm{C}_p \times \mathrm{C}_p$.

(3) $p \mid q - 1$, G *is non-abelian, and* $G \cong \mathrm{C}_q \rtimes \mathrm{C}_p$.

University of St Andrews

The statement

Let p and q be positive prime numbers with $p \leqslant q$. Let G be a group of order pq. Then exactly one of the following holds:

(1) $G \cong \mathrm{C}_{pq}$.

(2) $p = q$ *and* $G \cong \mathrm{C}_p \times \mathrm{C}_p$.

(3) $p \mid q - 1$, G *is non-abelian, and* $G \cong \mathrm{C}_q \rtimes \mathrm{C}_p$.

A few things we had to take into consideration:

(a) Implicit: the semidirect product involves a non-trivial homomorphism $\mathrm{C}_p \to \mathrm{Aut}\,\mathrm{C}_q$.

(b) Implicit: the choice of this homomorphism does not matter.

(c) Implicit: such a homomorphism exists.

(d) The natural-language statement has the usual form of a classification result. This may not be user-friendly in Lean.

Rephrasing

We ended up choosing the following main theorems plus some corollaries:

Let p and q be positive prime numbers with $p < q$.

(i) *If G is a non-cyclic group of order p^2, then G is isomorphic to $\mathrm{C}_p \times \mathrm{C}_p$.*

(ii) *If G is a group of order pq and $p \nmid q - 1$, then G is cyclic.*

(iii) *If $p \mid q - 1$, then there exists a non-cyclic group of order pq.*

(iv) *If G is a non-cyclic group of order pq and $\varphi : \mathrm{C}_p \to \operatorname{Aut} \mathrm{C}_q$ is any non-trivial homomorphism of groups, then G is isomorphic to $\mathrm{C}_q \rtimes_\varphi \mathrm{C}_p$.*

Finiteness and cardinality

"Let G be a finite group of cardinality n."

1️⃣ `(G : Type*) [Group G] [Finite G] (n : ℕ) (h : Nat.card G = n)`

2️⃣ `(G : Type*) [Group G] [Fintype G] (n : ℕ) (h : Fintype.card G = n)`

```
class inductive Finite (α : Sort*) : Prop
  | intro {n : ℕ} : α ≃ Fin n → Finite _

class Fintype (α : Type*) where
  elems : Finset α
  complete : ∀ x : α, x ∈ elems

def Nat.card (α : Type*) : ℕ := ...

def Fintype.card (α : Type*)  [Fintype α] : ℕ := ...
```

University of St Andrews

Finite vs Fintype

```
instance Finite.of_fintype (α : Type*) [Fintype α] : Finite α

noncomputable def Fintype.ofFinite (α : Type*)  [Finite α] : Fintype α
```

`Fintype` is meant to be used for computable definitions and in general requires more work (without resorting to `classical`). For example, we initially had to prove the following.

```
instance MulEquiv.fintype (α β : Type*) [DecidableEq α] [DecidableEq β]
    [Mul α] [Mul β] [Fintype α] [Fintype β] : Fintype (α ≃* β) where
  ...

instance Fintype.decidableEqMulEquivFintype (α β : Type*)
    [DecidableEq β] [Fintype α] [Mul α] [Mul β] : DecidableEq (α ≃* β) :=
  fun a b => decidable_of_iff ((a : α → β) = b)
    (Injective.eq_iff DFunLike.coe_injective)
```

University of St Andrews

Switching from `Fintype` to `Finite`

Previously, Lagrange's Theorem:

```
theorem Subgroup.card_subgroup_dvd_card
    {α : Type u_1} [Group α] (s : Subgroup α) [Fintype α] [Fintype s] :
    Fintype.card ↥s ∣ Fintype.card α :=
  ...
```

The `[Fintype s]` is needed unless we use `classical`.

As of June:

```
theorem Subgroup.card_subgroup_dvd_card
    {α : Type u_1} [Group α] (s : Subgroup α) :
    Nat.card ↥s ∣ Nat.card α :=
  ...
```

This also covers infinite groups.

Homomorphisms

```
variable (G₁ G₂ : Type*) [Group G₁] [Group G₂]
```

The type of group homomorphisms from `G₁` to `G₂` is `MonoidHom G₁ G₂`, notation `G₁ →* G₂`.

Similarly, `MulEquiv G₁ G₂`, notation `G₁ ≃* G₂`, is the type of group isomorphisms (or in fact, semigroup isomorphisms).

An object of type `G₁ ≃* G₂` is a concrete isomorphism. When we only care that the groups are isomorphic, we can write `Nonempty (G₁ ≃* G₂)`.

How to say a homomorphism `φ : G₁ →* G₂` is non-trivial? At first we came up with `φ.ker ≠ ⊤` and `φ.range ≠ ⊥`. It turned out we can just write `φ ≠ 1`.

Cyclic groups

In Mathlib, `ZMod n` is the type of integers modulo n for non-zero n and we know that it is a commutative ring and it is cyclic as a group.

```
instance ZMod.commRing (n : ℕ) : CommRing (ZMod n)

instance ZMod.instIsAddCyclic (n : ℕ) : IsAddCyclic (ZMod n)
```

Unfortunately for us, additive notation is used for its group structure. We replace this with multiplicative notation in a new type:

```
abbrev MulZMod (n : ℕ) := Multiplicative (ZMod n)

instance isCyclic_multiplicative {α : Type u} [AddGroup α] [IsAddCyclic α] :
    IsCyclic (Multiplicative α) :=
  ...
```

Stating (iv) in Lean

Let p and q be positive prime numbers with $p < q$. If G is a non-cyclic group of order pq and $\varphi : \mathrm{C}_p \to \operatorname{Aut} \mathrm{C}_q$ is any non-trivial homomorphism of groups, then G is isomorphic to $\mathrm{C}_q \rtimes_\varphi \mathrm{C}_p$.

```
variable {p q : ℕ} (hp : p.Prime) (hq : q.Prime) (hpq : p < q)
variable {G : Type*} [Group G] [Finite G]

theorem mulEquiv_semidirectProduct_of_not_isCyclic_of_card
    (h : Nat.card G = p * q) (h' : ¬IsCyclic G)
    (φ : MulZMod p →* MulAut (MulZMod q)) (hφ : φ ≠ 1) :
    Nonempty (G ≃* MulZMod q ⋊[φ] MulZMod p) :=
  ...
```

Proof of (iv)

Let p and q be positive prime numbers with $p < q$. If G is a non-cyclic group of order pq and $\varphi : \mathrm{C}_p \to \mathrm{Aut}\,\mathrm{C}_q$ is any non-trivial homomorphism of groups, then G is isomorphic to $\mathrm{C}_q \rtimes_\varphi \mathrm{C}_p$.

- Let G be a non-cyclic group of order pq. Let $\varphi : \mathrm{C}_p \to\!\ast\, \mathrm{Aut}\,\mathrm{C}_q$ be non-trivial.
- By Sylow's Theorems, there is a subgroup H of order q and a subgroup K of order p in G; furthermore the number n of subgroups of order q in G is $1 \bmod q$.
- The index $|G : \mathrm{N}_G(H)|$ is also n and divides p by a corollary of Lagrange's Theorem. Therefore H is normal in G.
- The set $HK = \{hk : h \in H, k \in K\}$ has cardinality $|H|\,|K|\,/\,|H \cap K|$. Since H and K intersect trivially, HK has cardinality pq. Therefore HK is the biggest set (and hence subgroup) in G.
- K acts on H by conjugation, giving rise to some $\psi : K \to\!\ast\, \mathrm{Aut}\,H$ and some $G \simeq\!\ast\, H \rtimes_\psi K$.
- Since there are instances of $H \simeq\!\ast\, \mathrm{C}_q$ and $K \simeq\!\ast\, \mathrm{C}_p$, and ψ is compatible with φ, we obtain some $G \simeq\!\ast\, \mathrm{C}_q \rtimes_\varphi \mathrm{C}_p$ by a congruence lemma for semidirect products.

A key lemma

$\mathrm{Aut}\,\mathrm{C}_q$ is isomorphic to C_{q-1}.
(Hence if $p \mid q-1$, then there exists a unique non-trivial homomorphic image of C_p in $\mathrm{Aut}\,\mathrm{C}_q$; and if $p \nmid q-1$, then any $\mathrm{C}_p \to* \mathrm{Aut}\,\mathrm{C}_q$ is trivial.)

We laid out many options for a proof assuming Mathlib and chose the one with the least amount of mathematical content:

```
variable (p : ℕ) [Fact (p.prime)]

def addEquivAddAutZMod : AddAut (ZMod p) ≃* (ZMod p)ˣ where ...

def mulEquivMulAutMulZMod : MulAut (MulZMod p) ≃* (ZMod p)ˣ :=
  AddEquiv.toMultiplicative.mulEquiv.symm.trans <| addEquivAddAutZMod p

lemma mulAut_MulZMod_isCyclic : IsCyclic (MulAut (MulZMod p)) := ...

lemma card_mulAut_mulZMod :
    Nat.card (MulAut (MulZMod p)) = p - 1 := ...
```

Concluding the project

- The code is completely `sorry`-free at about 1000 lines.
- We have learned a lot about Lean and Mathlib.
- We are in the process of refactoring, generalising our lemmas, and submitting pull requests to Mathlib in small chunks.
- We feel there is room for more automation, and more documentation and organisation.
- Many more theorems in group theory and lemmas about finite objects in general await formalisation. And maybe a computational library?