IMPERIAL

Department of Mathematics
Imperial College London

Coursework 1: Group Actions

Student name: *Frankie Feng-Cheng WANG*

Course: *MATH70040-Formalising Mathematics* – Lecturer: *Dr Bhavik Mehta*
Due date: *January 22, 2026*

Contents

1. Introduction

This report presents a Lean 4 formalisation of basic results about group actions. I define a minimal action class, construct the associated permutation representation, and show that stabilizers form subgroups. The Lean development follows the standard textbook proofs but makes every step explicit, so an algebraist unfamiliar with Lean can still map each line of code to a familiar argument.

1.1. Main theorem.

Theorem. Let X be a G-set. For each $g \in G$, the map $\varphi_g : X \to X$ defined by $\varphi_g(x) = g \cdot x$ is a permutation of X. The map $\Phi : G \to \mathrm{Sym}(X)$ given by $\Phi(g) = \varphi_g$ is a group homomorphism, and for all $g \in G$ and $x \in X$, $\Phi(g)(x) = g \cdot x$.

1.2. Reader's guide. This report is written for an external examiner who knows abstract algebra but not Lean. Each section contains (1) a short mathematical explanation in plain English and (2) a Lean excerpt mirroring the textbook proof. The code is intentionally explicit and avoids heavy automation so that the structure of the proof is visible.

2. Background and Design Choices

2.1. Why a custom action class. Mathlib already provides `MulAction`, but I define a small custom class `GroupAction` with just two axioms: compatibility with multiplication and the identity action. This keeps the development minimal and makes it easy to map Lean lines to textbook lines. The cost is that I cannot use existing `MulAction` lemmas, but the benefit is transparency.

2.2. What Lean is checking. Lean treats each statement as a typed object. When I define `sigmaPerm`, Lean checks that the inverse function is a two-sided inverse. When I define `phi`, Lean checks that its image lies in the group `Equiv.Perm X` and that multiplication is permutation composition. This explicit checking prevents hidden gaps that can slip into informal proofs.

3. Definitions

This section introduces the group action class and the standing assumptions used throughout the file.

3.1. Lean code. I first declare a minimal class for group actions:

```
    class GroupAction (G : Type*) [Monoid G] (X : Type*) where
    -- The action map: apply g to x.
    act : G → X → X
    -- Compatibility with multiplication in G.
    ga_mul : ∀ g₁ g₂ x, act (g₁ * g₂) x = act g₁ (act g₂ x)
    -- The identity element acts as the identity function.
    ga_one : ∀ x, act 1 x = x
```

Note that I use `Monoid` here because the definition of a group action does not require inverses. Later results about permutation representations and stabilizers require a `Group`, so the subsequent sections assume `[Group G]`.
Throughout the file I work with a group G acting on a type X:

```
    -- Fix a group and an action instance.
    variable {G : Type*} [Group G] {X : Type*} [GroupAction G X]
```

4. Permutation Representation

The core construction is the map $\sigma_g : X \to X$ given by $\sigma_g(x) = g \cdot x$. The inverse of σ_g is $\sigma_{g^{-1}}$, so σ_g is a permutation. This yields the permutation representation $\phi : G \to \mathrm{Sym}(X)$.

4.1. Proof sketch (informal). To show that σ_g is a bijection, compute $\sigma_{g^{-1}}(\sigma_g(x)) = (g^{-1}g) \cdot x = 1 \cdot x = x$, and similarly $\sigma_g(\sigma_{g^{-1}}(x)) = x$. The representation is defined by $\phi(g) = \sigma_g$, and multiplicativity follows from the action axiom:

$$\phi(g_1 g_2)(x) = (g_1 g_2) \cdot x = g_1 \cdot (g_2 \cdot x) = (\phi(g_1)\phi(g_2))(x).$$

4.2. Lean code. First, define the underlying action map:

```
/-- The action map `sigma g : X → X`
given by `x ↦ g • x`. -/
def sigma (g : G) : X → X :=
-- We use the action directly.
fun x => GroupAction.act g x
```

Then package it as a permutation using the inverse action:

```
/-- The permutation of `X` induced by `g`,
with inverse given by `g⁻¹`. -/
def sigmaPerm (g : G) : Equiv.Perm X := by
refine
    { toFun := sigma g
        invFun := sigma g⁻¹
        left_inv := ?_
        right_inv := ?_ }
· intro x
calc
-- Apply the action axiom, then cancel g⁻¹ * g.
GroupAction.act g⁻¹ (GroupAction.act g x) =
GroupAction.act (g⁻¹ * g) x := by
simpa using (GroupAction.ga_mul g⁻¹ g x).symm
_ = GroupAction.act (1 : G) x := by simp
_ = x := GroupAction.ga_one x
· intro x
calc
-- The symmetric calculation for g * g⁻¹.
GroupAction.act g (GroupAction.act g⁻¹ x) =
GroupAction.act (g * g⁻¹) x := by
simpa using (GroupAction.ga_mul g g⁻¹ x).symm
_ = GroupAction.act (1 : G) x := by simp
_ = x := GroupAction.ga_one x
```

Define the representation and its basic properties:

```
/-- The permutation representation `phi : G → Equiv.Perm X`
induced by the action. -/
def phi (g : G) : Equiv.Perm X :=
sigmaPerm g

```

```
-- `phi` agrees with the action on elements.
lemma phi_apply (g : G) (x : X) : phi g x = GroupAction.act g x := rfl

lemma phi_mul (g₁ g₂ : G) : phi (g₁ * g₂) = phi g₁ * phi g₂ := by
-- Reduce equality of permutations to pointwise equality.
apply Equiv.ext
intro x
calc
phi (g₁ * g₂) x = GroupAction.act (g₁ * g₂) x := rfl
_ = GroupAction.act g₁ (GroupAction.act g₂ x) := GroupAction.ga_mul g₁
 ↪  g₂ x
_ = (phi g₁ * phi g₂) x := rfl

lemma phi_one : phi (1 : G) = (1 : Equiv.Perm X) := by
apply Equiv.ext
intro x
calc
phi (1 : G) x = GroupAction.act (1 : G) x := rfl
_ = x := GroupAction.ga_one x
_ = (1 : Equiv.Perm X) x := by simp [Equiv.Perm.one_apply]
```

Finally, package the existence statement:

```
-- Existence of the permutation representation with the required
 ↪  properties.
theorem groupActionToPermRepresentation :
∃ (ψ : G → Equiv.Perm X),
(∀ g x, ψ g x = GroupAction.act g x) ∧
(∀ g₁ g₂, ψ (g₁ * g₂) = ψ g₁ * ψ g₂) ∧
(ψ 1 = 1) := by
exact ⟨phi, ⟨phi_apply, ⟨phi_mul, phi_one⟩⟩⟩
```

5. Stabilizers

For a fixed $x \in X$, the stabilizer is the subset

$$G_x = \{g \in G \mid g \cdot x = x\}.$$

In Lean this is first defined as a set, then shown to be the carrier of a subgroup, and finally packaged as a `Subgroup`.

5.1. Proof sketch (informal).

The identity element fixes every x, so $1 \in G_x$. If g_1 and g_2 fix x, then $(g_1 g_2) \cdot x = g_1 \cdot (g_2 \cdot x) = g_1 \cdot x = x$, so $g_1 g_2 \in G_x$. If g fixes x, then $g^{-1} \cdot x = g^{-1} \cdot (g \cdot x) = (g^{-1} g) \cdot x = x$, so $g^{-1} \in G_x$.

5.2. Lean code.

Start from the set definition:

```
/-- The stabilizer set
`G_x = { g ∈ G | g • x = x }`. -/
def stabilizerSet (x : X) : Set G :=
-- Membership means g fixes x.
    { g : G | GroupAction.act g x = x }
```

Package it as a subgroup by checking closure:

```lean
/-- The stabilizer `G_x` as a subgroup of `G`. -/
def stabilizer (x : X) : Subgroup G := by
exact
    { carrier := stabilizerSet (G := G) (X := X) x
        one_mem' := by
        -- The identity fixes every x.
        simp [stabilizerSet, GroupAction.ga_one x]
        mul_mem' := by
        intro g₁ g₂ hg₁ hg₂
        calc
        -- Closure under multiplication uses the action axiom.
        GroupAction.act (g₁ * g₂) x = GroupAction.act g₁
         ↪  (GroupAction.act g₂ x) := by
        simpa using (GroupAction.ga_mul g₁ g₂ x)
        _ = GroupAction.act g₁ x := by
        rw [hg₂]
        _ = x := hg₁
        inv_mem' := by
        intro g hg
        calc
        -- If g fixes x then g⁻¹ fixes x.
        GroupAction.act g⁻¹ x = GroupAction.act g⁻¹ (GroupAction.act g
         ↪  x) := by
        rw [hg]
        _ = GroupAction.act (g⁻¹ * g) x := by
        simpa using (GroupAction.ga_mul g⁻¹ g x).symm
        _ = GroupAction.act (1 : G) x := by simp
        _ = x := GroupAction.ga_one x }
```

Then show the set is the carrier of a subgroup:

```lean
/-- The stabilizer set is the carrier of a subgroup of `G`. -/
theorem stabilizerSet_isSubgroup (x : X) :
∃ H : Subgroup G, (H : Set G) = stabilizerSet (G := G) (X := X) x := by
refine ⟨stabilizer (G := G) (X := X) x, rfl⟩
```

6. Main Theorem

The Lean theorem `groupActionToPermRepresentation` packages the permutation representation together with its key properties. The proof uses the lemmas from the previous section: `phi_apply`, `phi_mul`, and `phi_one`.

7. Reflection

Writing the proofs in Lean forced me to make every step explicit. In particular:

- I had to specify exactly where associativity of the action is used.
- I had to show explicitly that $\sigma_{g^{-1}}$ is an inverse.
- I had to unfold the definition of permutation multiplication.

The resulting code is not shorter than a textbook proof, but it is more precise. An external examiner can read the surrounding explanations and then see that each Lean line matches a specific mathematical claim.
I estimate the formalisation took over fifteen hours, mostly due to understanding Lean's typeclass resolution and the mechanics of equivalences and permutations in mathlib. The final development is compact but captures the standard argument for the permutation representation and the stabilizer subgroup.

8. AI Assistance Statement

I used AI assistance to help draft Lean proofs and improve code readability. All AI-generated content was reviewed and verified by me to ensure mathematical accuracy.

9. References

John B. Fraleigh, Victor J. Katz, *A First Course in Abstract Algebra*, Addison–Wesley, 2003, Section 16 (Group Actions).