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 21, 2026*

Contents

1. Introduction

This is my submission for Coursework 1 of MATH70040-Formalising Mathematics, on the topic of group actions. In this project, I formalise the definition of group actions and prove basic properties using Lean 4 with mathlib. I deliberately use a custom `GroupAction` class (rather than `MulAction`) to mirror the textbook development and keep the proofs close to the algebraic arguments. The Main Theorm is

> **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. Also the map $\Phi : G \to S_X$ defined by $\Phi(g) = \varphi_g$ is a group homomorphism with the property that for all $g \in G$ and $x \in X$, $\Phi(g)(x) = g \cdot x$.

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 that mirrors the textbook proof. The code is intentionally explicit and avoids advanced automation, so that the steps correspond to the usual algebraic reasoning.

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: associativity of the action and the identity action. This keeps the development minimal and makes it easy to map each Lean line to a textbook line. The cost is that I cannot use the existing lemmas and automation for `MulAction`, but the benefit is transparency.

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

3. Definitions

In this section, I define the basic notions of group actions, including the definition of a group action, and some examples of group actions.

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 :   g  g  x, act (g  * g ) x = act g  (act g  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. However, since the subsequent theorems about group actions will involve groups, I will use `Group` in the later sections.
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, we compute $\sigma_{g^{-1}}(\sigma_g(x)) = (g^{-1}g) \cdot x = 1 \cdot x = x$, and similarly $\sigma_g(\sigma_{g^{-1}}(x)) = x$. Then ϕ is defined by $\phi(g) = \sigma_g$. 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 (g  g  : G) : phi (g  * g ) = phi g  * phi g  := by
  -- Reduce equality of permutations to pointwise equality.
  apply Equiv.ext
  intro x
  calc
    phi (g  * g ) x = GroupAction.act (g  * g ) x := rfl
    _ = GroupAction.act g  (GroupAction.act g  x) := GroupAction.ga_mul g  g  x
    _ = (phi g  * phi g ) 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 :
    (  : G → Equiv.Perm X),
    (  g x,   g x = GroupAction.act g x)
    (  g  g ,   (g  * g ) =   g  *   g )
    (  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 $(g_1g_2) \cdot x = g_1 \cdot (g_2 \cdot x) = g_1 \cdot x = x$, so $g_1g_2 \in G_x$. 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 g g hg hg
        calc
          -- Closure under multiplication uses the action axiom.
          GroupAction.act (g * g ) x = GroupAction.act g (GroupAction.act g x) := by
            simpa using (GroupAction.ga_mul g g x)
          _ = GroupAction.act g x := by
            rw [hg ]
          _ = x := hg
      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 development follows the textbook argument: apply g^{-1} to both sides, use associativity of the action, then use the identity axiom. The map $g \mapsto \sigma_g$ is then a group homomorphism into the symmetric group on X.

7. Reflection

Writing the proofs in Lean forced me to make every step explicit: I had to specify which group identity was used, how associativity of the action is applied, and how permutation

multiplication is defined. The resulting code is not shorter than a textbook proof, but it is far more precise. An external examiner can read the surrounding explanations and then see that each line of code matches a specific mathematical claim. This is the main benefit of formalisation: we trade brevity for clarity and machine-checked correctness.
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 (OpenAI Codex) to help draft Lean proofs, refactor code for readability, and produce an initial report outline. All AI-generated content was reviewed and edited by me. I verified that the final Lean code matches the intended mathematics and that the written explanations accurately describe the formal development.