09-subgroup.lean

```
import Mathlib
```

Substructures

Diffrent people formalize things differently. This is especially true when it comes to substructures and quotients.

In this file, we show how to use the Mathlib's API for substructures, from subsets to subgroups. It's a sophisticated hierarchy that I'm still trying to fully understand myself. For the philosophy behind this design, see MiL chapter 8.

1 Subsets

1.1 Objects

In the previous lectures, we regarded types as sets intuitively. This is not flexible when one wishes to restrict to only a fraction of the elements of a type. It's also hard to implement unions and intersections of sets this way. Mathlib provides a dedicated type `Set α`, consists of all the subsets of a type `α`.

```
section

variable (α : Type*) (s t u : Set α) (a : α)

#print Set
```

You can see that `Set α` is defined as `α → Prop`.

This means a subset `s : Set α` tells you, for each `a : α`, whether `a` belongs to `s` or not. This proposition is denoted by `a ∈ s`, `Set.mem s a`, or `s.Mem a`.

Note that you are not supposed to write `s a` directly. The function definition of `Set α` should be regarded as an implementation detail.

```
#check a ∈ s
#check s.Mem a
```

A subset can be constructed using the set-builder notation `{x : α | p x}` or `setOf p`, where `p : α → Prop` is a predicate on `α`.

```
#check setOf (fun x ↦ x = a)
example : Set α := {x : α | x = a}
```

The same as above

```
example : Set α := {a}
```

The same as above

```
example : Set α := Set.singleton a
```

The same as above
[TODO] Are they definitionally equal?

```
example : Set ℕ := {n | n > 514}
example (n : ℕ) : n ∈ {x | x > 514} ↔ n > 514 := by rfl

#check (∅ : Set α)
```

The empty subset

```
example : ∅ = {x : α | False} := by rfl
example : a ∈ (∅ : Set α) ↔ False := by rfl
#check Set.mem_empty_iff_false -- corresponding [@simp] lemma

#check (Set.univ : Set α)
```

The universal subset

```
example : Set.univ = {x : α | True} := by rfl
example : a ∈ Set.univ := by trivial
#check Set.mem_univ -- corresponding [@simp] lemma

#check sᶜ
```

The complement of a subset `Set.compl s`

```
example : sᶜ = {x | x ∉ s} := by rfl
example : a ∈ sᶜ ↔ a ∉ s := by rfl
#check Set.mem_compl -- corresponding [@simp] lemma

#check s ⊆ t
```

Subset relation `Set.Subset s t`

```
example : s ⊆ t ↔ ∀ x : α, x ∈ s → x ∈ t := by rfl
example (ha : a ∈ s) (hst : s ⊆ t) : a ∈ t := hst ha
#check Subset -- [TODO] the implicit {x : α}

#check s ∩ t
```

Intersection of two subsets `Set.inter s t`

```
example : a ∈ s ∩ t ↔ a ∈ s ∧ a ∈ t := by rfl

#check s ∪ t
```

Union of two subsets `Set.union s t`

```
example : a ∈ s ∪ t ↔ a ∈ s ∨ a ∈ t := by rfl
```

`ext` tactic reduces subset equality to element membership. Fundamentally this is implimented using function & propositional extensionality.

```
#check Set.ext
example : s ∩ t = t ∩ s := by ext x; simp [and_comm]
```

[EXR] De Morgan's law for sets

```
example : s ∩ (t ∪ u) = (s ∩ t) ∪ (s ∩ u) := by
  ext x
  constructor
  · intro ⟨h₁, h₂⟩
    rcases h₂ with (h₂ | h₂)
    · left
      exact ⟨h₁, h₂⟩
    · right
      exact ⟨h₁, h₂⟩
  · tauto_set

#help tactic tauto_set -- Wheelchair tactic for set equations

end
```

1.2 Morphisms

Functions between types induce functions between subsets.

```
section

variable {α β : Type*} (f : α → β) (s : Set α) (t : Set β) (a : α) (b : β)

#check Set.range f
example : Set.range f = {y | ∃ x, f x = y} := by rfl
example : Set.range f = {f x | x : α} := by rfl -- set-builder notation for range
example : b ∈ Set.range f ↔ ∃ x, f x = b := by rfl
#check Set.mem_range -- corresponding [@simp] lemma

#check f '' s
```

Image of a subset `Set.image f s`

```
#check Set.image f s
example : f '' s = {y | ∃ x ∈ s, f x = y} := by rfl
example : f '' s = {f x | x ∈ s} := by rfl -- set-builder notation for image
example : b ∈ f '' s ↔ ∃ x ∈ s, f x = b := by rfl
#check Set.mem_image -- corresponding [@simp] lemma

#check f ⁻¹' t
```

Preimage of a subset `Set.preimage f t`

```
#check Set.preimage f t
example : f ⁻¹' t = {x | f x ∈ t} := by rfl
example : a ∈ f ⁻¹' t ↔ f a ∈ t := by rfl
#check Set.mem_preimage -- corresponding [@simp] lemma
```

Note the following is not a definitional equality. The last step invokes `propext`, which destroys definitional equality. It has some unfortunate consequences in later discussions, when additional structure is involved.

```
example : f '' Set.univ = Set.range f := by
  ext x
  rw [Set.mem_range, Set.mem_image]
  simp only [Set.mem_univ, true_and]
#check Set.image_univ -- corresponding [@simp] lemma
```

The so-called Galois connection between image and preimage.

```
example : f '' s ⊆ t ↔ s ⊆ f ⁻¹' t := by
  constructor
  · intro h x hx
    simp only [Set.mem_preimage]
    apply h
    simp only [Set.mem_image]
    use x
  · intro h y hy
    rcases hy with ⟨x, hxs, hxy⟩
    specialize h hxs
    simp only [Set.mem_preimage] at h
    rw [hxy] at h; exact h
#check Set.image_subset_iff -- corresponding [@simp] lemma
```

We recall some definitions about functions.

```
#check Function.comp
#check Function.Injective
#check Function.Surjective
#check Function.Bijective
```

For those seeking a more challenging exercise, try proving the Bernstein–Schroeder theorem. See MiL chapter 3 for an answer.

```
#check Function.Embedding.schroeder_bernstein

end
```

2 Subsemigroups

2.1 Objects

A `Subsemigroup G` is a subset of a `Semigroup G` that is closed under the multiplication.

It's actually a bundled structure consisting of a subset and a proof of closure. To use it like a subset, Mathlib registers `Subsemigroup G` as an instance of `SetLike G`. It provides

coercion from `Subsemigroup G` to `Set G`, so for `H : Subsemigroup G`, you can use `a ∈ H` to mean `a` belongs to the underlying subset of `H`.

```
section

variable (G : Type*) [Semigroup G]
variable (H₁ H₂ : Subsemigroup G) (a b : G)
example (ha : a ∈ H₁) (hb : b ∈ H₁) : a * b ∈ H₁ := mul_mem ha hb
```

the whole semigroup as a subgroup

```
#check (⊤ : Subsemigroup G)
example : (⊤ : Subsemigroup G) = ⟨Set.univ, by simp⟩ := by rfl
#synth Top (Subsemigroup G)
```

the empty subset as a subgroup

```
#check (⊥ : Subsemigroup G)
example : (⊥ : Subsemigroup G) = ⟨∅, by simp⟩ := by rfl
#synth Bot (Subsemigroup G)
```

The partial order structure on subsemigroups is inherited from subset relation of subsets.

```
example : H₁ ≤ H₂ ↔ H₁.carrier ⊆ H₂.carrier := by rfl
```

intersection of two subsemigroups

```
#check H₁ ⊓ H₂
#synth Min (Subsemigroup G)

example : H₁ ⊓ H₂ = ⟨H₁ ∩ H₂, by
    intro a b ha hb
    rcases ha with ⟨ha₁, ha₂⟩
    rcases hb with ⟨hb₁, hb₂⟩
    constructor
    all_goals apply mul_mem
    all_goals assumption
    ⟩ :=
  rfl
```

product of two subsemigroups.

```
#check H₁ ⊔ H₂
#synth Max (Subsemigroup G)
```

Definition of `H₁ ⊔ H₂` is more involved, relying the lattice structure of `Subsemigroup G`. it is defined as the infimum of all subsemigroups containing both `H₁` and `H₂`, where the infimum is given by intersection.

```
#synth CompleteLattice (Subsemigroup G)
```

This is characterized by the following properties.

```
#synth SemilatticeSup (Subsemigroup G)
example : H₁ ≤ H₁ ⊔ H₂ := by apply le_sup_left
example : H₂ ≤ H₁ ⊔ H₂ := by apply le_sup_right
example (K : Subsemigroup G) (h1 : H₁ ≤ K) (h2 : H₂ ≤ K) : H₁ ⊔ H₂ ≤ K :=
  sup_le h1 h2

end
```

2.2 Morphisms

Let's see how `MulHom` interacts with `Subsemigroup`.

```
section

variable {G₁ G₂ : Type*} [Semigroup G₁] [Semigroup G₂]
         (f : G₁ →ₙ* G₂) (H₁ : Subsemigroup G₁) (H₂ : Subsemigroup G₂)
```

The image of a subsemigroup under a `MulHom` is also a subsemigroup.

```
#check Subsemigroup.map f H₁
#check H₁.map f

example : Subsemigroup.map f H₁ = ⟨f '' H₁, by
    rintro x y ⟨a, ha, rfl⟩ ⟨b, hb, rfl⟩
    use a * b, H₁.mul_mem ha hb
    rw [map_mul]
    ⟩ := by rfl
```

The preimage of a subsemigroup under a `MulHom` is also a subsemigroup.

```
#check Subsemigroup.comap
#check H₂.comap f

example : Subsemigroup.comap f H₂ = ⟨f ⁻¹' H₂, by
    intro x y hx hy
    simp only [Set.mem_preimage] at hx hy ⊢
    rw [map_mul]
    exact mul_mem hx hy
    ⟩ := by rfl
```

To define the range of a `f : G₁ →ₙ* G₂`, a common idea is to adopt `(τ : Subsemigroup G₁).map f`. Unfortunately, this makes the underlying set being `f '' (univ : Set G₁)`, which is not definitionally equal to `Set.range f`. It will also cause `x ∈ τ` conditions in later proofs, redundant and annoying.

Hencee Mathlib define the range with some refinement: They manually replace the underlying set of `(τ : Subsemigroup G₁).map f` with `Set.range f`.

See Note range copy pattern for an official explanation.

```
#check MulHom.srange
```

the desired definitional equality

```
example : MulHom.srange f = ⟨Set.range f, by
    rintro x y ⟨a, rfl⟩ ⟨b, rfl⟩
    use a * b
    rw [map_mul]
    ⟩ := by rfl

example (x : G₂) : x ∈ MulHom.srange f ↔ x ∈ Set.range f := by rfl
#check MulHom.mem_srange -- corresponding Mathlib theorem

end
```

3 Submonoids

A `Submonoid M` is a subsemigroup of a `Monoid M` that contains the identity element.

```
section

variable (G : Type*) [Monoid G]
variable (H₁ H₂ : Submonoid G) (a b : G)

example : a ∈ H₁ → b ∈ H₁ → a * b ∈ H₁ := by apply mul_mem
example : (1 : G) ∈ H₁ := by apply one_mem
```

The whole monoid as a submonoid. Note the use of `with`, to extend the underlying `Subsemigroup` with the proof of containing `1`.

```
#check (⊤ : Submonoid G)
example : (⊤ : Submonoid G) = {(⊤ : Subsemigroup G) with
    one_mem' := by
      change 1 ∈ (⊤ : Set G)
      apply Set.mem_univ
    } := by rfl
#synth Top (Submonoid G)
```

The trivial submonoid consisting of only the identity element. Note the difference from `⊥ : Subsemigroup G`, which is the empty set.

```
#check (⊥ : Submonoid G)
example : (⊥ : Submonoid G) = {
    carrier := {1}
    one_mem' := by rfl
    mul_mem' := by
      rintro x y hx hy
      simp only [Set.mem_singleton_iff] at hx hy ⊢
      rw [hx, hy, one_mul]
    } := by rfl
#synth Bot (Submonoid G)
```

We don't repeat tedious lattice structure part, which is similar to those for `Subsemigroup`.

```
#synth CompleteLattice (Submonoid G)

end
```

3.1 Morphisms

`MonoidHom` interacts with `Submonoid` similarly to `MulHom` and `Subsemigroup`.

```
section

variable {G₁ G₂ : Type*} [Monoid G₁] [Monoid G₂]
         (f : G₁ →* G₂) (H₁ : Submonoid G₁) (H₂ : Submonoid G₂)
```

We still have image and preimage of submonoids, which can be built on top of those for subsemigroups, with extra care to verify the identity element membership.

```
#check Submonoid.map
example : Submonoid.map f H₁ = { Subsemigroup.map f.toMulHom H₁.toSubsemigroup with
      one_mem' := by
        simp
        use 1, H₁.one_mem
        rw [map_one]
    } := by rfl

#check Submonoid.comap
example : Submonoid.comap f H₂ = { Subsemigroup.comap f.toMulHom H₂.toSubsemigroup with
      one_mem' := by simp
    } := by rfl
```

Range is also specially handled as `MulHom.range`.

```
#check MonoidHom.mrange
example (x : G₂) : x ∈ MonoidHom.mrange f ↔ x ∈ Set.range f := by rfl
#check MonoidHom.mem_mrange -- corresponding Mathlib theorem
```

With the presence of identity element, we can define the kernel of a `MonoidHom`.

```
#check MonoidHom.mker
example : MonoidHom.mker f = (⊥ : Submonoid G₂).comap f := by rfl
```

[EXR] manual definition of `mker`

```
example : MonoidHom.mker f = {
      carrier := {x | f x = 1}
      one_mem' := by rw [Set.mem_setOf, map_one]
      mul_mem' := by
        rintro x y hx hy
        simp only [Set.mem_setOf] at hx hy ⊢
        rw [map_mul, hx, hy, one_mul]
    } := by rfl

example (x : G₁) : x ∈ MonoidHom.mker f ↔ f x = 1 := by rfl
```

```
#check MonoidHom.mem_mker -- corresponding Mathlib theorem

end
```

3.2 Exercise

As an exercise, let's define addition on `AddSubmonoid A` with the intrinsic definition, and show that it coincides with the supremum.

```
section

variable {A : Type*} [AddCommMonoid A]

instance : Add (AddSubmonoid A) := ⟨fun B₁ B₂ ↦ {
  carrier := {x | ∃ b₁ ∈ B₁, ∃ b₂ ∈ B₂, x = b₁ + b₂}
  zero_mem' := ⟨0, B₁.zero_mem, 0, B₂.zero_mem, by rw [add_zero]⟩
  add_mem' := by
    rintro x y ⟨b₁, hb₁, b₂, hb₂, rfl⟩ ⟨c₁, hc₁, c₂, hc₂, rfl⟩
    use b₁ + c₁, B₁.add_mem hb₁ hc₁
    use b₂ + c₂, B₂.add_mem hb₂ hc₂
    abel
}⟩

example (B₁ B₂ : AddSubmonoid A) : B₁ ⊔ B₂ = B₁ + B₂ := by
  apply le_antisymm
  · apply sup_le
    · intro x hx
      use x, hx, 0, B₂.zero_mem
      rw [add_zero]
    · intro x hx
      use 0, B₁.zero_mem, x, hx
      rw [zero_add]
  · intro x hx
    rcases hx with ⟨b₁, hb₁, b₂, hb₂, rfl⟩
    haveI : B₁ ≤ B₁ ⊔ B₂ := le_sup_left
    replace hb₁ : b₁ ∈ B₁ ⊔ B₂ := this hb₁
    haveI : B₂ ≤ B₁ ⊔ B₂ := le_sup_right
    replace hb₂ : b₂ ∈ B₁ ⊔ B₂ := this hb₂
    exact add_mem hb₁ hb₂

end
```

4 Subgroups

4.1 Objects

There's nothing new about `Subgroup G` of a `Group G` compared to `Subsemigroup` and `Submonoid`. It just adds the closure under taking inverses.

```
section
```

```
variable (G : Type*) [Group G]
variable (H₁ H₂ : Subgroup G) (a b : G)
example : a ∈ H₁ → b ∈ H₁ → a * b ∈ H₁ := by apply mul_mem
example : (1 : G) ∈ H₁ := by apply one_mem
example : a ∈ H₁ → a⁻¹ ∈ H₁ := by apply inv_mem
```

We skip the lattice structure again.

```
end
```

4.2 Morphisms

`MonoidHom` works for `Subgroup` as well.

```
section

variable {G₁ G₂ : Type*} [Group G₁] [Group G₂]
         (f : G₁ →* G₂) (H₁ : Subgroup G₁) (H₂ : Subgroup G₂)
```

Image and preimage of subgroups, upgraded to show closure under inverses.

```
#check Subgroup.map
#check Subgroup.comap
```

For groups, `mker` and `mrange` has been upgraded to `ker` and `range` respectively.

```
#check MonoidHom.ker
#check MonoidHom.range

example : MonoidHom.ker f = (⊥ : Subgroup G₂).comap f := by rfl
example : MonoidHom.ker f = {MonoidHom.mker f with
      inv_mem' := by simp
    } := by rfl

end
```

4.3 Normal Subgroups

For later discussions on quotient groups, we introduce normal subgroups here.

```
section

#check Subgroup.Normal
```

`Subgroup.Normal` is a bundled structure consisting of a proof of normality.

```
example {G : Type*} [Group G] (H : Subgroup G) :
    H.Normal ↔ ∀ h ∈ H, ∀ g : G, g * h * g⁻¹ ∈ H := by
  constructor
  · intro ⟨h⟩
    exact h
  · intro h
    exact ⟨h⟩
```

The kernel of a group homomorphism is a normal subgroup.

```
example {G₁ G₂ : Type*} [Group G₁] [Group G₂]
    (f : G₁ →* G₂) : (f.ker).Normal := by
  constructor
  intro x hx y
  rw [MonoidHom.mem_ker]
  rw [map_mul, map_mul, hx, map_inv, mul_one, mul_inv_cancel]
```

Actually, Mathlib contains an instance for kernels, so that Lean automatically recognizes the normality of kernels.

```
#check MonoidHom.normal_ker
example {G₁ G₂ : Type*} [Group G₁] [Group G₂]
    (f : G₁ →* G₂) : (f.ker).Normal := inferInstance

end
```

[TODO]

- Indexed infimum and supremum of substructures
- `xxxClass` for substructures as canonical maps