1 Introduction

When formalizing mathematics in Lean, most Types have some universal property in some category. This universal property either describes the set of homomorphisms out of the object or the set of homomorphisms into the object. We often want to use a universal property both to define morphisms and to check if morphisms are equal.

For example the universal property of the polynomial ring $R[X]$ over a commutative ring R is given by the following isomorphism of functors.

$$\text{CommRing}(R[X], -) \cong \text{CommRing}(R, -) \times \text{Forget}(-) \tag{1}$$

In other words to define a ring homomorphism out of the polynomial ring $R[X]$ into a ring S, it suffices to provide a ring homomorphism $R \to S$ and an element of S. We will call this map $eval : \text{CommRing}(R, S) \times S \to \text{CommRing}(R[X], S)$. $eval$ is one direction of the isomorphism.

Additionally, if we have a ring hom $f : R[X] \to S$, we can recover a ring homomorphism $R \to S$ by composing with the canonical map C from $R \to R[X]$ and an element of S by applying f to X. We also know how $eval$ computes on X and C. We know that $eval(f, s)(X) = s$ and $eval(f, s) \circ C = f$.

This natural isomorphism also gives a method to check if two homomorphisms out of the polynomial ring are equal. Two ring homomorphisms f and g from $R[X]$ into S out of the polynomial ring are equal if $f(X) = g(X)$ and $f \circ C = g \circ C$, where C is the canonical morphism $R \to R[X]$.

As an example of how to use this universal property consider the following problem. Suppose R, S and T are commutative rings and $f : R \to S$ and $g : S \to T$ are ring homomorphisms. Given an element t of T we can define two a map $R[X] \to T$ in two different ways. Suppose $map(f)$ is the canonical morphism $R[X] \to S[X]$ given the map $f : R \to S$.

$map(f)$ can be defined using the universal property. $map(f)$ is equal to $eval(X, C \circ f)$

Suppose we want to check commutativity of the following diagram

$$\begin{tikzcd}
S[X] \arrow[dr, "eval(t{,}g)"] & \\
R[X] \arrow[u, "map(f)"] \arrow[r, "eval(t{,}g\circ f)"'] & T
\end{tikzcd}$$

Written equational this is the same as checking the following

$$eval(t, g) \circ map(f) = eval(t, g \circ f) \tag{2}$$

Unfolding *map* gives us

$$eval(t, g) \circ eval(X, C \circ f) = eval(t, g \circ f) \tag{3}$$

We can apply extensionality to this equation and we now have to check

$$eval(t, g)(eval(X, C \circ f)(X)) = eval(t, g \circ f)(X) \tag{4}$$

and

$$eval(t, g) \circ eval(X, C \circ f) \circ C = eval(t, g \circ f) \circ C \tag{5}$$

These equations can both be easily verified using the two equations $\forall f\ s, eval(f, s)(X) = s$ and $\forall f\ s, eval(f, s)(X) = \circ C = f$.

2 Representable Functor

In this section we will give a characterisation of a corepresentation of a copresheaf $F : \mathcal{C} \to \text{Set}$ for a category C.

Let $\mathcal{C}$ be a category and let $F : \mathcal{C} \to \text{Set}$ be a co-presheaf on $\mathcal{C}$. Then we will define a corepresentation of F to be the following. Given the following 5 things, we can recover an object R of $\mathcal{C}$ and the natural isomorphism of functors $\mathcal{C}(R, -) \cong F$.

1. An object R of $\mathcal{C}$
2. An element u of $F(R)$
3. For every object X of $\mathcal{C}$, a map $e_X : F(X) \to \mathcal{C}(R, X)$
4. For any object X of $\mathcal{C}$, and every element $f : F(X)$, $F(e_X(f))(u) = f$
5. For any two morphisms $f, g : \mathcal{C}(R, X)$, if $F(f)(u) = F(g)(u)$ then $f = g$

A representation of a functor $F : C^{op} \to \text{Set}$ is the dual concept to a corepresentable functor. We define it explicitly here.

1. An object R of $\mathcal{C}$
2. An element c of $F(R^{op})$
3. For every object X of $\mathcal{C}$, a map $r_X : F(R^{op}) \to \mathcal{C}(X, R)$
4. For any object X of $\mathcal{C}$, and every element $f \in F(X^{op})$, $F(r_X^{op}(f))(c) = f$

5. For any two morphisms $f, g : \mathcal{C}(X, R)$, if $F(f^{op})(c) = F(g^{op})(c)$ then $f = g$

We now prove that this definition is equivalent to the more usual definition. The usual definition is a pair of an object R of $\mathcal{C}$ and the following isomorphism of functors.

$$\mathcal{C}(R, -) \cong F \tag{6}$$

Given a map $f \in \mathcal{C}(R, X)$, then $F(f)(u)$ is an element of $F(X)$. This gives one direction of the isomorphism. The other direction is given by e_X, the map that is Axiom 3 of our definition of corepresentation. The fact that these two are two sided inverses of each other is given by Axiom 4, for one direction. The other direction, i.e. proving $e_X(F(f)(u)) = f$, is given by applying extensionality. After applying extensionality we have to check $F(e_X(F(f)(u)))(u) = F(f)(u)$. Axiom 4 proves this equality

To check the naturality of this isomorphism we have to check that given any $g \in \mathcal{C}(X, Y)$ and $f \in \mathcal{C}(R, X)$ $F(g)(F(f)(u)) = F(g \circ f)(u)$. This is just functoriality of F.

2.1 Examples of Representable Functors in Lean

2.1.1 Free Module

We'll show the definition of the universal property in Lean as well and how each part corresponds to each of the five things above. The free module is over a set S is the corepresentation of the functor $M \mapsto \mathrm{Hom}_{Set}(S, \mathrm{Forget}(M))$, from Mod → Set.

1) An object R of $\mathcal{C}$

```
@[derive add_comm_group, derive module R]
def free_module (R : Type) [comm_ring R] (S : Type) : Type :=
```

The free module over a Type `S` is an object in the category of `R` modules. This is a Type and a `module R`, which in this particular implementation are not bundled.

2) An element u of $F(R)$

```
def X (a : S) : free_module R S :=
```

This is the canonical map of sets `S` to `free_module R S`.

3) For every object X of $\mathcal{C}$, a map $e_X : F(X) \to \mathcal{C}(R, X)$

```
def extend (f : S → M) : free_module R S →ₗ[R] M :=
```

This is the canonical way of extending a map `S → M` to a map `free_module R S →ₗ[R] M`.

4) For any object X of $\mathcal{C}$, and every element $f : F(X)$, $F(e_X(f))(u) = f$

```
@[simp] lemma extend_X (f : S → M) (a : S) : extend f (X a : free_module
    R S) = f a :=
```

This lemma says that when you extend a map on a basis to a map on the whole module,

5) For any two morphisms $f, g : \mathcal{C}(R, X)$, if $F(f)(u) = F(g)(u)$ then $f = g$

```
@[ext] lemma hom_ext {f g : free_module R S →ₗ[R] M} (h : ∀ a, f (X a) =
    g (X a)) : f = g :=
```

2.1.2 Product and Coproduct of Abelian Groups

Given two abelian (additive) groups, `A` and `B`, the universal property of the product is given by the following. The product of abelian groups A and B is the representation of the functor $\text{Ab}(-, A) \times \text{Ab}(-, B)$, from Ab^{op} to Set.

1) The object is the product of sets `A × B` with the obvious group structure.

2) There are two projection maps `fst : A × B →+ A` and `snd : A × B →+ B`. (The types `A` and `B` are both explicit arguments to each of these definitions).

3) Given two maps `C →+ A` and `C →+ B`, we can make a map `C →+ A × B`

```
def prod (f : C →+ A) (g : C →+ B) : C →+ A × B :=
```

4) We have two lemmas relating `fst` and `prod` and `snd` and `prod`.

```
lemma fst_comp_prod (f : C →+ A) (g : C →* B) : (fst A B).comp (f.prod
   g) = f :=

lemma snd_comp_prod (f : C →+ A) (g : C →* B) : (snd A B).comp (f.prod
   g) = g :=
```

5) Two maps into the product are equal if each component is equal

```
lemma hom_ext {f g : C →+ A × B}
  (h1 : (fst A B).comp f = (fst A B).comp g)
  (h2 : (snd A B).comp f = (snd A B).comp g) : f = g :=
```

The coproduct of abelian groups is the same object as the product, but with a different universal property. The coproduct of A and B is the corepresentation of the functor $\text{Ab}(A, -) \times \text{Ab}(B, -)$ from $\text{Ab} \to \text{Set}$.

1) The object is the product of sets `A × B` with the obvious group structure.

2) There are two embeddings `inl : A →+ A × B` and `inr : B →+ A × B`. (The types `A` and `B` are both explicit arguments to each of these definitions).

3) Given two maps `A →+ C` and `B →+ C`, we can make a map `A × B →+ C`.

```
def coprod (f : A →+ C) (g : B →+ C) : A × B →+ C :=
```

4) We have two lemmas relating `inl` and `coprod` and `inr` and `coprod`.

```
lemma coprod_comp_inl (f : A →+ C) (g : B →+ C) : (f.coprod g).comp (
    inl A B) = f :=

lemma coprod_comp_inr (f : A →+ C) (g : B →+ C) : (f.coprod g).comp (
    inr A B) = g :=
```

5) We have an extensionality lemma saying two maps out of the coproduct are equal if they are equal after composition with each embedding

```
lemma hom_ext {f g : A × B →+ C}
  (h1 : f.comp (inl A B) = g.comp (inl A B))
  (h2 : f.comp (inr A B) = g.comp (inr A B)) : f = g :=
```

2.1.3 Integers

The integers are the initial ring. So they are the corepresention of the constant functor sending every ring to the set with one element.

1) The object is the Type of integers with the normal ring structure.

2) This part is not defined in Lean. There is no real use for it, it would just be an element of the unit type.

3) This is the canonical map from the integers into any ring.

4) This part is not defined in Lean, if it was it would just be an equality of two elements of the unit type. This is not useful.

5) There is a lemma saying that for any ring `R` any two maps from the integers into `R` are equal.

2.1.4 Polynomials

The polynomial ring over a commutative ring R is the corepresentation of the functor $S \mapsto S \times \text{CommRing}(R, S)$.

1) The object is the polynomial ring `polynomial R`, with the obvious ring structure.

2) There is a ring homomorphism `C : R →+* polynomial R`, and an element `X : polynomial R`

3) There is an evaluation homomorphism to evaluate a polynomial of `R` in an arbitrary commutative ring `S` given an element of `S` and a ring homomorphism `R →+* S`.

```
def eval₂ (f : R →+* S) (x : S) : polynomial R →+* S :=
```

4) There are two lemmas saying what the evaluation map does to both X and C.

```
lemma eval₂_X : eval₂ f x X = x :=

lemma eval₂_comp_C : (eval₂ f x).comp C = f :=
```

5) There is a lemma stating that two ring homomorphisms out of the polynomial ring are equal if they are equal on X and equal on C.

```
lemma hom_ext {f g : polynomial R →+* S}
  (h1 : f X = g X) (h2 : f.comp C = g.comp C) : f = g :=
```

3 Adjunction

Given a functor $F : \mathcal{C} \to \mathcal{D}$ we will define a left adjoint of F to a corepresention of $\mathcal{D}(F(A), -)$ for every object A of $\mathcal{C}$. We will call this map of object sets G. We can now prove that G is a functor. Explicitly, this is the following data.

1. A map of object set $G : \mathrm{Obj}(\mathcal{D}) \to \mathrm{Obj}(\mathcal{C})$
2. For every object X of $\mathcal{D}$, a map $\eta_X \in \mathcal{D}(X, F(G(X)))$
3. For every object X of $\mathcal{D}$ and Y of $\mathcal{C}$, a map of sets $e_{X,Y} : \mathcal{D}(X, F(Y)) \to \mathcal{C}(G(X), Y)$
4. For every object X of $\mathcal{D}$ and Y of $\mathcal{C}$, and every element $f \in \mathcal{D}(X, F(Y))$, $F(e_{X,Y}(f)) \circ \eta_X = f$
5. $f, g \in \mathcal{C}(G(X), Y)$ are equal iff $F(f) \circ \eta_X = F(g) \circ \eta_X$

We can prove that G is in fact a functor. Given objects A and B of $\mathcal{D}$ and a map $f : \mathcal{D}(, B)$,A then define $G(f)$ to be $e_{A,G(B)}(\eta_B \circ f)$.

Then

$$G(\mathrm{id}_A) = e_{A,G(A)}(\eta_A \circ \mathrm{id}_A) = e_{A,G(A)}(F(\mathrm{id}_{G(A)}) \circ \eta_A) = \mathrm{id}_{G(A)} \tag{7}$$

Also for $f : \mathcal{D}(A, B)$ and $g : \mathcal{C}(B, C)$ then we apply the extensionality lemma

$$\begin{array}{cr} & F(G(g \circ f)) \circ \eta_A \\ = & F(e_{A,G(C)}(\eta_C \circ g \circ f)) \circ \eta_A \\ = & \eta_C \circ g \circ f \\ = & F(e_{A,G(B)}(\eta_C \circ g)) \circ \eta_B \circ f \\ = & F(e_{A,G(B)}(\eta_C \circ g)) \circ F(e_{B,G(C)}(\eta_B \circ f)) \\ = & F(G(g) \circ G(f)) \end{array} \tag{8}$$

This works similarly for right adjoints.

4 Method for Checking Equalities

The basic method for checking equalities of morphisms is to write every morphism in terms of the universal property whenever possible, and then use extensionality as many times as possible and then repeatedly rewrite using Axiom 4 of the corepresentable or representable functor axioms.

4.1 Examples of Checking Equalities

4.1.1 Polynomial Example

We demonstrate how to prove that given a polynomial `p` over a commutative ring `R`, commutative rings `S` and `T`, ring homs, `f : R →+* S` and `g : S →+* T` and `x : T`, that

```
eval₂ g x (map f p) = eval₂ (g.comp f) x p
```

Here, `map f` is the canonical map `polynomial R →+* polynomial S` given by extending the ring homomorphism `f`.

The first step is to write this equality as an equality of morphisms. So we should try to prove

```
(eval₂ g x).comp (map f) = eval₂ (g.comp f) x
```

Then we write the morphism `map` using the universal property. `map f` is equal to `eval₂ (f.comp C) X`.

We now have to prove

```
(eval₂ g x).comp (eval₂ (f.comp C) X) = eval₂ (g.comp f) x
```

We can apply the extensionality theorem for maps out of the polynomial ring. We have two equalities to prove

```
((eval₂ g x).comp (eval₂ (f.comp C) X)) X = eval₂ (g.comp f) x X
```

and

```
((eval₂ g x).comp (eval₂ (f.comp C) X)).comp C = (eval₂ (g.comp f) x X).
    comp C
```

We will focus on the first equalities. Using the theorem about how `eval₂` computes on `X` three times, we can simplify the equality to `x = x` which is true by reflexivity.

For the second equality we can use the theorem about how `eval₂` computes on `C`. Applying it three times gives the equality (and associativity of composition) `g.comp f = g.comp f`, which is true by reflexivity.

4.1.2 Polynomial Associativity Example

The Free Module functor which we will call F is the left adjoint to the forgetful functor $\mathrm{Forget} : \mathrm{Mod}_R \to \mathrm{Set}$.

We will call the map $A \to \mathrm{Forget}(F(A))$, X and use subscripts for application. We might not always write the forgetful functor explicitly.

The map $(A \to \mathrm{Forget}(B)) \to \mathrm{Mod}_R(F(A), B)$ will be called *extend*.

We will define multiplication on $F(\mathbb{N})$ as a morphism of Type

$$\mathrm{Mod}_R(F(\mathbb{N}), [F(\mathbb{N}), F(\mathbb{N})]) \tag{9}$$

Square brackets indicate the hom object in Mod_R.

The definition of multiplication is as follows

$$\mathit{extend}(m \mapsto \mathit{extend}(n \mapsto X_{m+n})) \tag{10}$$

We would like to use our extensionality lemma to prove associativity of multiplication. In order to do this, we have to state associativity as an equality of morphisms, as opposed to an equality of elements of the free module. We use two operations to do this, both of which are versions of linear map composition as a linear map.

For modules A, B, and C we have two versions of linear map composition which we call R and L.

$$\begin{gathered} R \in \mathrm{Mod}_R([A, B], [[B, C], [A, C]]) \\ L \in \mathrm{Mod}_R([B, C], [[A, B], [A, C]]) \end{gathered} \tag{11}$$

Then the map $a, b, c \mapsto \mathrm{mul}(\mathrm{mul}(a)(b))(c)$ can be written as

$$\mathrm{Forget}(R)(\mathrm{mul}) \circ \mathrm{mul} \tag{12}$$

Similarly the map $a, b, c \mapsto \mathrm{mul}(a)(\mathrm{mul}(b)(c))$ can be written as

$$\mathrm{Forget}(L)(\mathrm{mul}) \circ (R \circ \mathrm{mul}) \tag{13}$$

These linear maps both have Type $\mathrm{Mod}_R(F(\mathbb{N}), [F(\mathbb{N}), F(\mathbb{N})])$.

We can apply the extensionality lemma three times (using functional extensionality as well).

We then have to check that for any $i, j, k \in \mathbb{N}$ that

$$(R)(\mathrm{mul}) \circ \mathrm{mul}(X_i)(X_j)(X_k) = (L)(\mathrm{mul}) \circ (R \circ \mathrm{mul})(X_i)(X_j)(X_k) \tag{14}$$

Unfolding the definitions of linear map composition and applying Axiom 4 several times gives the following equality.

$$X_{(i+j)+k} = X_{i+(j+k)} \tag{15}$$

The associativity of multiplication of polynomials was reduced to the associativity of addition of natural numbers.

5 Potential Improvements

Having to unfold the definition of linear map composition is unsatisfying as well as having to directly apply funext. Probably it would be better to express the universal property as a representation of a functor $\mathrm{Mod}_R \to \mathrm{Mod}_R$ and to develop some theory of representable functors in enriched categories.

Given a functor $F : \mathrm{Mod}_R \to \mathrm{Mod}_R$, then if X is a corepresentation of F, the object $[X, A]$ is a representation of the functor $B \mapsto [B, F(A)]$. The hom object inherits a universal property from X and linear composition can probably be written in terms of this universal property.