

# LeanCat Benchmark (Part I: 1-Category Theory)

LeanCat Consortium

January 9, 2026

**Problem 1** (Basic - Easy). — *Theorem: Let  $\mathcal{C}$  be a category and  $\text{Id}_{\mathcal{C}}$  the identity functor. Then monoid of natural transformations  $\text{End}(\text{Id}_{\mathcal{C}})$  is commutative.*

```
import Mathlib
open CategoryTheory
variable {C : Type*} [Category.{v} C]
theorem id_comm (α β : (1 C) → (1 C)) : α » β = β » α := by
  sorry
```

**Problem 2** (Basic - Easy). — *Theorem: Let  $\mathcal{C}$  be a category and let  $f, g$  be morphisms in  $\mathcal{C}$  such that  $f \circ g$  is monic. Then  $g$  is monic.*

```
import Mathlib
open CategoryTheory
variable {C : Type*} [Category C]

theorem monic_of_comp_monic {X Y Z : C} (g : X → Y) (f : Y → Z)
  [Mono (g » f)] : Mono g := by
  sorry
```

**Problem 3** (Basic - Easy). — *Theorem: The forgetful functor  $\mathcal{T}\text{op} \rightarrow \mathcal{S}\text{et}$ ,  $\mathcal{G}\text{rp} \rightarrow \mathcal{S}\text{et}$ ,  $\mathcal{R}\text{ing} \rightarrow \mathcal{A}\text{b}$ ,  $\mathcal{T}\text{op}_* \rightarrow \mathcal{T}\text{op}$  are faithful but not full.*

```
import Mathlib
open CategoryTheory Limits

theorem forget_Top_faithful_not_full :
  (forget TopCat).Faithful ∧ ¬(forget TopCat).Full := by
  sorry

theorem forget_Grp_faithful_not_full :
  (forget Grp).Faithful ∧ ¬(forget Grp).Full := by
```

```
sorry
```

```
theorem forget_Ring_Ab_faithful_not_full :  
  (forget2 RingCat Ab).Faithful ∧ ¬ (forget2 RingCat Ab).Full := by  
  sorry
```

```
theorem forget_TopPointed_faithful_not_full :  
  (Under.forget (terminal TopCat)).Faithful ∧ ¬ (Under.forget (terminal  
    → TopCat)).Full := by  
  sorry
```

**Problem 4** (Basic - Easy). — *Theorem: Let  $\{*\} \in \mathcal{S}\text{et}$  be the terminal object in  $\mathcal{S}\text{et}$ . Then  $\text{hom}_{\mathcal{S}\text{et}}(\{*\}, -) : \mathcal{S}\text{et} \rightarrow \mathcal{S}\text{et}$  is an equivalence of categories.*

```
import Mathlib  
  
open CategoryTheory  
  
universe u  
  
def fromTerminalFunctor : Type u → Type u where  
  obj α := PUnit.{u} → α  
  map {α β} (f : α → β) := fun g => f ∘ g  
  map_id := by  
    intro α  
    funext g x  
    rfl  
  map_comp := by  
    intro α β γ f g  
    funext h x  
    rfl  
  
theorem fromTerminalEquivalence : fromTerminalFunctor.IsEquivalence := sorry
```

**Problem 5** (Basic - Medium). — *Theorem: Let  $\mathcal{C}$  be a category, if every idempotent in  $\mathcal{C}$  can be factored into an epimorphism followed by a monomorphism, then all idempotents split in  $\mathcal{C}$ .*

```
import Mathlib  
  
open CategoryTheory Idempotents  
  
variable {C : Type*} [Category.{v} C]  
  
theorem idempotent_splitting_from_epi_mono_factorization  
  (h : ∀ (X : C) (p : X → X) (hpp : p ≫ p = p),  
   ∃ (Y : C) (e : X → Y) (he : Epi e) (m : Y → X) (hm : Mono m), p = e ≫  
   ↣ m) :  
  IsIdempotentComplete C := by  
  sorry
```

**Problem 6** (Basic - Medium). — *Theorem: Let  $\mathcal{C}$  and  $\mathcal{D}$  be two categories. Let  $F : \mathcal{C} \rightarrow \mathcal{D}$  be a functor. Then  $F$  has a quasi-inverse if and only if*

1.  $F$  is fully faithful;
2.  $F$  is essentially surjective.

```
import Mathlib

open CategoryTheory

theorem functor_has_quasi_inverse_iff {C D : Type*} [Category C] [Category D]
   $\hookrightarrow$  (F : C  $\Rightarrow$  D) :
    ( $\exists$  G : D  $\Rightarrow$  C, (Nonempty (Functor.id C  $\cong$  F.comp G))  $\wedge$  (Nonempty (G.comp F
       $\hookrightarrow$   $\cong$  Functor.id D)))
     $\Leftrightarrow$  F.IsEquivalence := by
  sorry
```

**Problem 7** (Basic - Medium). — *Theorem: Let  $\mathcal{C}$  be a category and  $\text{Kar}(\mathcal{C})$  be its idempotent completion. Let  $I : \mathcal{C} \rightarrow \text{Kar}(\mathcal{C})$  be the inclusion. Then for any category  $\mathcal{D}$  in which idempotent splits and a functor  $F : \mathcal{C} \rightarrow \mathcal{D}$ , there is a unique (up to isomorphism) functor  $F' : \text{Kar}(\mathcal{C}) \rightarrow \mathcal{D}$  such that  $F' \circ I = F$ .*

```
import Mathlib

open CategoryTheory

variable {C D : Type*} [Category C] [Category D]

theorem karoubi_universal_property [IsIdempotentComplete D] (F : C  $\Rightarrow$  D) :
   $\exists!$  (F' : (Idempotents.Karoubi C)  $\Rightarrow$  D), (Idempotents.toKaroubi C)  $\gg$  F' = F
   $\hookrightarrow$  := by
  sorry
```

**Problem 8** (Basic - Medium). — *Theorem: Let  $G_1$  and  $G_2$  be two objects in the category  $\mathcal{G}\text{rp}$  of groups. The coproduct of  $G_1$  and  $G_2$  in  $\mathcal{G}\text{rp}$  is equivalent to the free product of  $G_1$  and  $G_2$ .*

```
import Mathlib

open CategoryTheory Limits

universe u
variable {G H : Grp.{u}}

def M : Bool  $\rightarrow$  Type u
| true  $\Rightarrow$  G
| false  $\Rightarrow$  H

instance : (b : Bool)  $\rightarrow$  Group (M (G := G) (H := H) b) := by
  intro b
  cases b  $\langle ; \rangle$  dsimp [M]  $\langle ; \rangle$  infer_instance

abbrev FreeProdGrp : Grp.{u} :=
  Grp.of (Monoid.CoprodI (ι := Bool) (M := M (G := G) (H := H)))
```

  

```
theorem freeProdGrp_iso_coprod [HasBinaryCoproduct G H] :
```

```
Nonempty (FreeProdGrp (G := G) (H := H) ≅ coprod G H) := by
sorry
```

**Problem 9** (Basic - Medium). — *Theorem:* There exists a morphism in  $\mathcal{R}\text{ing}$  such that it is epic but not surjective.

```
import Mathlib

open CategoryTheory

theorem exists_epic_not_surjective_in_Ring :
  ∃ (A B : RingCat) (f : A → B), Epi f ∧ ¬ Function.Surjective f := by
sorry
```

**Problem 10** (Basic - High). — *Theorem:* Let  $F : \mathcal{G}\text{rp} \rightarrow \mathcal{S}\text{et}$  be the functor that  $G \mapsto \{g \in G \mid g^2 = 1\}$ . Then  $F$  is representable.

```
import Mathlib

open CategoryTheory

def functor_involution : Grp.{u} ≈ Type u where
  obj := fun G => { g : G.carrier | g * g = 1 }
  map := fun {G H} f x => ⟨f.hom x.val, by
    refine Set.mem_setOf.mpr ?_
    rcases x with ⟨g, hg⟩
    simp only [Set.mem_setOf_eq] at hg
    rw [← f.hom.map_mul, hg]
    simp only [map_one]
  ⟩

theorem involution_functor_representable :
  CategoryTheory.Functor.IsCorepresentable functor_involution := by
sorry
```

**Problem 11** (Basic - High). — *Definition:* Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A morphism  $f : x \rightarrow y$  in  $\mathcal{C}$  is called **initial** if for any object  $c \in \mathcal{C}$ , a morphism  $g : U(c) \rightarrow U(x)$  is a morphism in  $\mathcal{C}$  whenever  $f \circ g : U(c) \rightarrow U(y)$  is a morphism in  $\mathcal{C}$ .

*Definition:* An initial morphism  $f : x \rightarrow y$  such that the underlying morphism  $U(f) : U(x) \rightarrow U(y)$  is monic is called an **embedding**.

*Definition:* If  $f : x \rightarrow y$  is an embedding, then  $(x, f)$  is called an **initial subobject** of  $y$ .

*Definition:* In a concrete category an object  $I$  is called **injective** provided that for any embedding  $m : A \rightarrow B$  and any morphism  $f : A \rightarrow C$  there exists a morphism  $g : B \rightarrow C$  extending  $f$ , i.e.,  $g \circ m = f$

*Definition:* A concrete category has **enough injectives** provided that each of its objects is an initial subobject of an injective object.

*Theorem:* The category  $\mathcal{T}\text{op}^{CH}$  of compact Hausdorff space has enough injectives.

```
import Mathlib

open CategoryTheory
```

```

namespace CAT_statement_S_0011

universe u uX

variable {X : Type uX} [Category.{vX} X]

namespace AHS

structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C → X
  [U_Faithful : U.Faithful]

attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

def IsInitialHom {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  ∀ {Z : C.C} (g : C.U.obj Z → C.U.obj A),
    (exists h : Z → B, C.U.map h = g » C.U.map f) →
      (exists k : Z → A, C.U.map k = g)

def IsEmbedding {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  IsInitialHom f ∧ Mono (C.U.map f)

def IsInjectiveObj {C : ConcreteCat (X:= X)} (I : C.C) : Prop :=
  ∀ {A B : C.C} (m : A → B),
    IsEmbedding m →
      ∀ (f : A → I), ∃ g : B → I, m » g = f

def HasEnoughInj {C : ConcreteCat (X:= X)} : Prop :=
  ∀ x : C.C, ∃ (I : C.C) (f : x → I),
    IsInjectiveObj I ∧ IsEmbedding f

end AHS

def CompHausConcrete : AHS.ConcreteCat (X := Type u) :=
{ C := CompHaus.{u}
  U := forget CompHaus}

theorem CompHaus_Has_E够Inj : AHS.HasEnoughInj (C := CompHausConcrete) := by
  sorry

end CAT_statement_S_0011

```

**Problem 12** (Basic - High). — *Theorem: Let  $\mathcal{C}$  be a category and let  $f : x \rightarrow y$  be a morphism in  $\mathcal{C}$ . Then  $f$  is a monomorphism in  $\mathcal{C}$  if and only if there exists a category  $\mathcal{D}$  and a faithful functor  $I : \mathcal{C} \rightarrow \mathcal{D}$  such that  $f$  is a section in  $\mathcal{D}$ .*

```
import Mathlib
```

```

open CategoryTheory

theorem mono_iff_exists_embedding_section
  {C : Type u} [Category C]
  {X Y : C} (f : X → Y) :
  Mono f ↔
    ∃ (D : Type v),
    ∃ (_ : Category D)
    (I : C ≈ D)
    (g : I.obj Y → I.obj X),
    (I.map f) ≫ g = 1 (I.obj X)
    ∧
    (∀ {A B : C}, Function.Injective (fun h : A → B ↪ I.map h)) :=
by
  sorry

```

**Problem 13** (Basic - High). — *Theorem: The category  $\mathcal{T}\text{op}^{CH}$  of compact Hausdorff space is dually equivalent to the category of commutative unital  $C^*$ -algebras and algebra homomorphisms.*

```

import Mathlib

open CategoryTheory

universe u

structure CommCStarAlgCat : Type (u + 1) where
  of ::

  carrier : Type u
  [commCStarAlgebra : CommCStarAlgebra carrier]

attribute [instance] CommCStarAlgCat.commCStarAlgebra

namespace CommCStarAlgCat

instance : CoeSort CommCStarAlgCat (Type u) :=
⟨CommCStarAlgCat.carrier⟩

instance : Category CommCStarAlgCat where
  Hom A B := A →*a [C] B
  id A := StarAlgHom.id C A
  comp f g := g.comp f

end CommCStarAlgCat

theorem gelfandDuality : Nonempty (CompHaus.{u} ≈ (CommCStarAlgCat.{u})op) :=
  by
  sorry

```

**Problem 14** (Basic - High). — *Definition: Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A morphism  $f : x \rightarrow y$  in  $\mathcal{C}$  is called **initial** if for any object  $c \in \mathcal{C}$ , a morphism  $g : U(c) \rightarrow U(x)$  is a morphism in  $\mathcal{C}$  whenever  $f \circ g : U(c) \rightarrow U(y)$  is a morphism in  $\mathcal{C}$ .*

*Definition: An initial morphism  $f : x \rightarrow y$  such that the underlying morphism  $U(f) : U(x) \rightarrow U(y)$  is monic is called an **embedding**.*

*Definition: In a concrete category an object  $I$  is called **injective** provided that for any embedding  $m : A \rightarrow B$  and any morphism  $f : A \rightarrow C$  there exists a morphism  $g : B \rightarrow C$  extending  $f$ , i.e.,  $g \circ m = f$*

*Theorem: In  $\mathcal{P}\mathcal{O}\mathcal{S}\text{et}$ , injective objects are precisely the suplattice.*

```

import Mathlib

open CategoryTheory

namespace CAT_statement_S_0014

universe u uX

variable {X : Type uX} [Category.{vX} X]

namespace AHS2

structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C ≈ X
  [U_Faithful : U.Faithful]

attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

def IsInitialHom {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  ∀ {Z : C.C} (g : C.U.obj Z → C.U.obj A),
  (exists h : Z → B, C.U.map h = g ▷ C.U.map f) →
  (exists k : Z → A, C.U.map k = g)

def IsEmbedding {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  IsInitialHom f ∧ Mono (C.U.map f)

def IsInjectiveObj {C : ConcreteCat (X:= X)} (I : C.C) : Prop :=
  ∀ {A B : C.C} (m : A → B),
  IsEmbedding m →
  ∀ (f : A → I), ∃ g : B → I, m ▷ g = f

end AHS2

namespace Poset
def PosetConcrete : AHS2.ConcreteCat (Type u) where
  C := PartOrd.{u}
  cat := inferInstance
  U := forget PartOrd
  U_Faithful := inferInstance

theorem injective_iff_suplattice (P : PartOrd.{u}) :
  AHS2.IsInjectiveObj (C := PosetConcrete) P ↔ ∀ (s : Set P), ∃ x, IsLUB s x
  → := by

```

```

sorry

end Poset

end CAT_statement_S_0014

```

**Problem 15** (Basic - High). — *Definition:* Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A morphism  $f : x \rightarrow y$  in  $\mathcal{C}$  is called **initial** if for any object  $c \in \mathcal{C}$ , a morphism  $g : U(c) \rightarrow U(x)$  is a morphism in  $\mathcal{C}$  whenever  $f \circ g : U(c) \rightarrow U(y)$  is a morphism in  $\mathcal{C}$ .

*Definition:* An initial morphism  $f : x \rightarrow y$  such that the underlying morphism  $U(f) : U(x) \rightarrow U(y)$  is monic is called an **embedding**.

*Definition:* In a concrete category an object  $I$  is called **injective** provided that for any embedding  $m : A \rightarrow B$  and any morphism  $f : A \rightarrow C$  there exists a morphism  $g : B \rightarrow C$  extending  $f$ , i.e.,  $g \circ m = f$

*Theorem:* In  $\mathcal{L}\text{at}_\wedge$ , the category of meet semilattice and meet preserving maps, injective objects are frames.

```

import Mathlib

open CategoryTheory

namespace CAT_statement_S_0015

universe u uX

variable {X : Type uX} [Category.{vX} X]

namespace AHS

structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C ≈ X
  [U_Faithful : U.Faithful]

attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

def IsInitialHom {C : ConcreteCat (X := X)} {A B : C.C} (f : A → B) : Prop :=
  ∀ {Z : C.C} (g : C.U.obj Z → C.U.obj A),
  (exists h : Z → B, C.U.map h = g ▷ C.U.map f) →
  (exists k : Z → A, C.U.map k = g)

def IsEmbedding {C : ConcreteCat (X := X)} {A B : C.C} (f : A → B) : Prop :=
  IsInitialHom f ∧ Mono (C.U.map f)

def IsInjectiveObj {C : ConcreteCat (X := X)} (I : C.C) : Prop :=
  ∀ {A B : C.C} (m : A → B),
  IsEmbedding m →
  ∀ (f : A → I), ∃ g : B → I, m ▷ g = f

end AHS

namespace SemilatInfCat

```

```

def forget : SemilatInfCat.{u} → Type u where
  obj A := A
  map {A B} f := f

instance : forget.Faithful where
  map_injective {A B} f g h := by
    ext x
    simpa using congrArg (fun k => k x) h

def SemilatInfCatConcrete : AHS.ConcreteCat (X := Type u) :=
{ C := SemilatInfCat.{u}
  U := forget }

class IsFrameObj (P : SemilatInfCat.{u}) (sSup : Set P.X → P.X) (sInf : Set
  ↪ P.X → P.X) : Prop where
  exists_sSup :
    (∀ (s : Set P.X), IsLUB s (sSup s))
  exists_sInf :
    (∀ (s : Set P.X), IsGLB s (sInf s))
  distributive :
    (∀ (a : P.X), ∀ (s : Set P.X),
      a ⊓ sSup s = sSup (Set.image (fun (b : P.X) => a ⊓ b) s))

theorem AHS_injective_iff_frameObj (P : SemilatInfCat) :
  AHS.InjectiveObj (C := SemilatInfCatConcrete) P ↔ ∃ (sSup : Set P.X →
    ↪ P.X) (sInf : Set P.X → P.X), IsFrameObj P sSup sInf := by
  sorry

end SemilatInfCat

end CAT_statement_S_0015

```

**Problem 16** (Basic - High). — *Definition:* Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A morphism  $f : x \rightarrow y$  in  $\mathcal{C}$  is called **initial** if for any object  $c \in \mathcal{C}$ , a morphism  $g : U(c) \rightarrow U(x)$  is a morphism in  $\mathcal{C}$  whenever  $f \circ g : U(c) \rightarrow U(y)$  is a morphism in  $\mathcal{C}$ .

*Definition:* An initial morphism  $f : x \rightarrow y$  such that the underlying morphism  $U(f) : U(x) \rightarrow U(y)$  is monic is called an **embedding**.

*Definition:* In a concrete category an object  $I$  is called **injective** provided that for any embedding  $m : A \rightarrow B$  and any morphism  $f : A \rightarrow C$  there exists a morphism  $g : B \rightarrow C$  extending  $f$ , i.e.,  $g \circ m = f$

*Theorem:* In  $\mathcal{Ab}$  the injective objects are precisely the divisible abelian groups.

```

import Mathlib
open CategoryTheory
namespace CAT_statement_S_0016
universe u uX

```

```

variable {X : Type uX} [Category.{vX} X]

namespace AHS

structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C → X
  [U_Faithful : U.Faithful]

attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

def IsInitialHom {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  ∀ {Z : C.C} (g : C.U.obj Z → C.U.obj A),
  (exists h : Z → B, C.U.map h = g » C.U.map f) →
  (exists k : Z → A, C.U.map k = g)

def IsEmbedding {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  IsInitialHom f ∧ Mono (C.U.map f)

def IsInjectiveObj {C : ConcreteCat (X:= X)} (I : C.C) : Prop :=
  ∀ {A B : C.C} (m : A → B),
  IsEmbedding m →
  ∀ (f : A → I), ∃ g : B → I, m » g = f

end AHS

def AddCommGrpConcrete : AHS.ConcreteCat (X := Type u) :=
{ C := AddCommGrp.{u}
  U := forget AddCommGrp}

theorem AddCommGrp.injective_iff_divisible (A : AddCommGrp.{u}) :
  AHS.IsInjectiveObj (C:= AddCommGrpConcrete) A ↔ ∀ (n : ℕ) (hn : n ≠ 0) (a
  ↤ : A), ∃ b : A, n • b = a := by
  sorry

end CAT_statement_S_0016

```

**Problem 17** (Basic - High). — *Definition:* Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A morphism  $f : x \rightarrow y$  in  $\mathcal{C}$  is called **initial** if for any object  $c \in \mathcal{C}$ , a morphism  $g : U(c) \rightarrow U(x)$  is a morphism in  $\mathcal{C}$  whenever  $f \circ g : U(c) \rightarrow U(y)$  is a morphism in  $\mathcal{C}$ .

*Definition:* An initial morphism  $f : x \rightarrow y$  such that the underlying morphism  $U(f) : U(x) \rightarrow U(y)$  is monic is called an **embedding**.

*Definition:* In a concrete category an object  $I$  is called **injective** provided that for any embedding  $m : A \rightarrow B$  and any morphism  $f : A \rightarrow C$  there exists a morphism  $g : B \rightarrow C$  extending  $f$ , i.e.,  $g \circ m = f$

*Theorem:* In  $\mathcal{T}\text{op}$ , the injective objects are precisely the retracts of powers  $C^I$  of the space  $C := (\{0, 1, 2\}, \{\emptyset, \{0, 1\}, \{0, 1, 2\}\})$ .

```

import Mathlib

open CategoryTheory Limits TopologicalSpace

```

```

namespace CAT_statement_S_0017

universe u uX

variable {X : Type uX} [Category.{vX} X]

namespace AHS

structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C → X
  [U_Faithful : U.Faithful]

attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

def IsInitialHom {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  ∀ {Z : C.C} (g : C.U.obj Z → C.U.obj A),
    (exists h : Z → B, C.U.map h = g » C.U.map f) →
      (exists k : Z → A, C.U.map k = g)

def IsEmbedding {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  IsInitialHom f ∧ Mono (C.U.map f)

def IsInjectiveObj {C : ConcreteCat (X:= X)} (I : C.C) : Prop :=
  ∀ {A B : C.C} (m : A → B),
    IsEmbedding m →
      ∀ (f : A → I), ∃ g : B → I, m » g = f

end AHS

def S : TopCat.{u} :=
  letI : TopologicalSpace (Fin 3) := generateFrom {{0, 1} : Set (Fin 3)}
  TopCat.of (ULift.{u} (Fin 3))

def TopCatConcrete : AHS.ConcreteCat (X := Type u) :=
  { C := TopCat.{u}
    U := forget TopCat}

theorem Inj_in_TopCat {Y : TopCat.{u}} :
  AHS.IsInjectiveObj (C:= TopCatConcrete) Y ↔ ∃ (I : Type u), Nonempty
    ↳ (Retract Y (piObj (fun (_ : I) => S))) := by
    sorry

end CAT_statement_S_0017

```

**Problem 18** (Basic - High). — *Definition:* Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A morphism  $f : x \rightarrow y$  in  $\mathcal{C}$  is called **initial** if for any object  $c \in \mathcal{C}$ , a morphism  $g : U(c) \rightarrow U(x)$  is a morphism in  $\mathcal{C}$  whenever  $f \circ g : U(c) \rightarrow U(y)$  is a morphism in  $\mathcal{C}$ .

*Definition:* An initial morphism  $f : x \rightarrow y$  such that the underlying morphism  $U(f) : U(x) \rightarrow U(y)$  is monic is called an **embedding**.

*Definition:* In a concrete category an object  $I$  is called **injective** provided that for any embedding  $m : A \rightarrow B$  and any morphism  $f : A \rightarrow C$  there exists a morphism  $g : B \rightarrow C$  extending  $f$ , i.e.,  $g \circ m = f$

*Theorem: In the category  $\mathcal{T}op^{CH}$  of compact Hausdorff space, injective objects are precisely the retracts of powers  $[0, 1]^I$  of the unit interval.*

```

import Mathlib
open CategoryTheory
namespace CAT_statement_S_0018
universe u uX
variable {X : Type uX} [Category.{vX} X]
namespace AHS
structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C ≈ X
  [U_Faithful : U.Faithful]
attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

def IsInitialHom {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  ∀ {Z : C.C} (g : C.U.obj Z → C.U.obj A),
  (exists h : Z → B, C.U.map h = g ▷ C.U.map f) →
  (exists k : Z → A, C.U.map k = g)

def IsEmbedding {C : ConcreteCat (X:= X)} {A B : C.C} (f : A → B) : Prop :=
  IsInitialHom f ∧ Mono (C.U.map f)

def IsInjectiveObj {C : ConcreteCat (X:= X)} (I : C.C) : Prop :=
  ∀ {A B : C.C} (m : A → B),
  IsEmbedding m →
  ∀ (f : A → I), ∃ g : B → I, m ▷ g = f

end AHS

def CompHausConcrete : AHS.ConcreteCat (X := Type u) :=
{ C := CompHaus.{u}
  U := forget CompHaus}

theorem tieze_urysohn {Y : CompHaus.{u}} :
  AHS.IsInjectiveObj (C:= CompHausConcrete) Y ↔ ∃ i : Type u, Nonempty
    ↳ (Retract Y (.of (∀ i : i, unitInterval))) := by
    sorry

end CAT_statement_S_0018

```

**Problem 19** (Adjunction - Easy). — *Theorem: A functor  $G : \mathcal{D} \rightarrow \mathcal{C}$  has a left adjoint if and only if for each  $c \in \mathcal{C}$ , the comma category  $(c \downarrow G)$  has an initial object.*

```

import Mathlib
open CategoryTheory Limits

variable {C : Type*} {D : Type*} [Category.{v₁} C] [Category.{v₂} D]

theorem functor_hasLeftAdjoint_iff_structuredArrow_hasInitial
  (G : D → C) :
  G.IsRightAdjoint ↔ ∀ c : C, HasInitial (StructuredArrow c G) := by
  sorry

```

**Problem 20** (Adjunction - Medium). — *Theorem: Let  $\mathcal{C}$  and  $\mathcal{D}$  be categories and let  $F : \mathcal{C} \rightarrow \mathcal{D}$  be a functor that admits a right adjoint  $G$ . Then  $F$  is fully faithful if and only if  $u : \text{Id}_{\mathcal{C}} \rightarrow G \circ F$  is isomorphism.*

```

import Mathlib
open CategoryTheory

variable {C D : Type*} [Category C] [Category D] (F : C → D) (G : D → C)

theorem fully_faithful_iff_unit_isIso (adj : F ⊢ G) :
  (F.Full ∧ F.Faithful) ↔ IsIso adj.unit := by
  sorry

```

**Problem 21** (Adjunction - Medium). — *Theorem: Let  $\mathcal{C}$  and  $\mathcal{D}$  be categories and let  $F : \mathcal{C} \rightarrow \mathcal{D}$  be a functor that admits a right adjoint  $G$ . Then  $G$  is an equivalence of categories if and only if  $F$  is fully faithful and  $G$  is conservative.*

```

import Mathlib
open CategoryTheory

variable {C : Type u₁} [Category.{v₁} C] {D : Type u₂} [Category.{v₂} D]

theorem
  ↳ right_adjoint_isEquivalence_iff_left_full_faithful_and_right_conservative
    (F : C → D) (G : D → C) (adj : F ⊢ G) :
    G.IsEquivalence ↔ (F.Full ∧ F.Faithful) ∧ G.ReflectsIsomorphisms := by
    sorry

```

**Problem 22** (Adjunction - Medium). — *Theorem: Let  $\mathcal{C}$  and  $\mathcal{D}$  be locally small categories and let  $F : \mathcal{C} \rightarrow \mathcal{D}$  be a functor. Then  $F$  admits a right adjoint if and only if for each  $d \in \mathcal{D}$ ,  $\text{hom}_{\mathcal{D}}(F(-), d) : \mathcal{C}^{\text{op}} \rightarrow \text{Set}$  is representable.*

```

import Mathlib
open CategoryTheory

variable {C : Type u₁} [Category.{v₁} C] {D : Type u₂} [Category.{v₂} D]

theorem isLeftAdjoint_iff_yoneda_comp_op_isRepresentable (F : C → D) :
  F.IsLeftAdjoint ↔ ∀ (d : D), (F.op ≫ yoneda.obj d).IsRepresentable := by

```

```
sorry
```

**Problem 23** (Adjunction - Medium). — *Theorem: Let  $A, B \in \mathcal{R}\text{ing}$  and let  $\phi : A \rightarrow B$  be a morphism in  $\mathcal{R}\text{ing}$ . It induces a functor  $\phi_* : {}_B\mathcal{A}\text{b} \rightarrow {}_A\mathcal{A}\text{b}$ ,  $(N, l_N) \mapsto (N, l_N \circ (\phi \otimes \text{id}))$ . Then the functor  $\phi_*$  admits a left adjoint  $\phi^* := B \otimes_A - : {}_A\mathcal{A}\text{b} \rightarrow {}_B\mathcal{A}\text{b}$  and a right adjoint  $\phi^! := \text{hom}_A(B, -) : {}_A\mathcal{A}\text{b} \rightarrow {}_B\mathcal{A}\text{b}$ .*

```
import Mathlib

open CategoryTheory

theorem ring_hom_induced_functor_has_adjoint
  {A B : RingCat} (φ : A → B) :
  ∃ (φ_pull : ModuleCat B ≅ ModuleCat A)
    (φ_push : ModuleCat A ≅ ModuleCat B)
    (φ_coind : ModuleCat A ≅ ModuleCat B),
  Nonempty (Adjunction φ_push φ_pull) ∧ Nonempty (Adjunction φ_pull
  ← φ_coind) := by
  sorry
```

**Problem 24** (Adjunction - Medium). — *Theorem: Let  $U : \mathcal{A}\text{b} \rightarrow \mathcal{G}\text{rp}$  be the forgetful functor. Then it admits a left adjoint.*

```
import Mathlib
open CategoryTheory
universe u

theorem forget_CommGrp_to_Grp_admits_left_adjoint :
  (forget₂ CommGrp.{u} Grp.{u}).IsRightAdjoint := by
  sorry
```

**Problem 25** (Monad - Medium). — *Theorem: Let  $\mathcal{C}, \mathcal{D}, \mathcal{E}$  be categories and  $U : \mathcal{D} \rightarrow \mathcal{C}$ ,  $V : \mathcal{E} \rightarrow \mathcal{C}$ ,  $F : \mathcal{D} \rightarrow \mathcal{E}$  be functors such that  $V \circ F = U$ . Suppose  $U, V$  have left adjoints and  $\mathcal{D}$  have coequalizers. If  $V$  reflects split epimorphisms to regular epimorphisms (or equivalently, the counit of the adjunction of  $U$  is a regular epimorphism), then  $F$  has a left adjoint.*

```
import Mathlib

open CategoryTheory

variable {C D E : Type*} [Category C] [Category D] [Category E]

namespace CategoryTheory

class Functor.ReflectsSplitEpimorphismsToRegularEpimorphisms (F : Functor C
  ← D) : Prop where
  reflects : ∀ {X Y} {f : X → Y} [IsSplitEpi (F.map f)], Nonempty (RegularEpi
  ← f)

end CategoryTheory

variable (U : D ≅ C) (V : E ≅ C) (F : D ≅ E)
```

```

theorem exists_left_adjoint_of_comp_eq (h : F ≫ V = U) (hU :
  ↳ U.IsRightAdjoint) (hV : V.IsRightAdjoint)
  (hV_refl : V.ReflectsSplitEpimorphismsToRegularEpimorphisms) :
  ↳ F.IsRightAdjoint := by
  sorry

```

**Problem 26** (Adjunction - Medium). — *Theorem: Let  $F, G, H$  be functors such that  $F \dashv G \dashv H$ . Then  $F$  is fully faithful if and only if  $H$  is fully faithful.*

```

import Mathlib

open CategoryTheory.Functor

variable {C : Type u₁} [Category.{v₁} C] {D : Type u₂} [Category.{v₂} D]
variable {F : C ⇒ D} {G : D ⇒ C} {H : C ⇒ D}

theorem fullyFaithful_iff_of_adjoint (hFG : F ⊣ G) (hGH : G ⊣ H) :
  (F.Full ∧ F.Faithful) ↔ (H.Full ∧ H.Faithful) := by
  sorry

```

**Problem 27** (Adjunction - Medium). — *Theorem: Let  $(\mathbb{Z}, \leq)$  be a poset, regarded as a category, then  $f \in \text{End}(\mathbb{Z})$  has left adjoint if and only if it has a right adjoint.*

```

import Mathlib

open CategoryTheory

theorem int_endofunctor_hasLeftAdjoint_iff_hasRightAdjoint (f : Z ⇒ Z) :
  f.IsRightAdjoint ↔ f.IsLeftAdjoint := by
  sorry

```

**Problem 28** (Adjunction - Medium). — *Theorem: Let  $(\mathbb{N}, \leq)$  be a poset, regarded as a category. There is a sequence of distinct functors  $G_n : \mathbb{N} \rightarrow \mathbb{N}$  such that  $G_0(x) = x + 1$  and  $G_{n+1} \dashv G_n$  for each  $n \in \mathbb{N}$ .*

```

import Mathlib

open CategoryTheory

theorem exists_sequence_of_distinct_adjoint_nat :
  ∃ G : ℕ → (ℕ ⇒ ℕ),
  Function.Injective G ∧
  (∀ x, (G 0).obj x = x + 1) ∧
  (∀ n, Nonempty (G (n + 1) ⊣ G n)) := by
  sorry

```

**Problem 29** (Adjunction - High). — *Theorem: Let  $(-)^\times : \mathcal{R}\text{ing} \rightarrow \mathcal{G}\text{rp}$  mapping a ring to its group of units. Then it admits a left adjoint.*

```

import Mathlib
open CategoryTheory

def RingCat.units : RingCat.{u} ≅ Grp.{u} where
  obj R := .of Rx
  map f := Grp.ofHom (Units.map f.hom)

theorem exists_leftAdjoint_unitFunctor :
  ∃ (left : Grp.{u} ≅ RingCat.{u}), Nonempty (left ≫ RingCat.units.{u}) :=
   $\begin{array}{c} \hookrightarrow \\ \text{by} \end{array}$ 
  sorry

```

**Problem 30** (Reflective - High). — *Theorem: There are categories  $\mathcal{C}$ ,  $\mathcal{D}$  and  $\mathcal{E}$  such that  $\mathcal{C}$  is a subcategory of  $\mathcal{D}$ ,  $\mathcal{D}$  be a subcategory of  $\mathcal{E}$  and  $\mathcal{C}$  is reflective in  $\mathcal{E}$ , but  $\mathcal{C}$  is not reflective in  $\mathcal{D}$ .*

```

import Mathlib
open CategoryTheory Functor

universe u v

namespace CategoryTheory

open Category Adjunction

variable {C : Type u1} {D : Type u2} {E : Type u3}
variable [Category.{v1} C] [Category.{v2} D] [Category.{v3} E]

class Reflective2 (R : D ≅ C) extends R.Faithful where
  L : C ≅ D
  adj : L ≫ R

end CategoryTheory

theorem exists_not_reflective :
  ∃ (E C D : Type u)
  ( $\begin{array}{c} \_ : \text{Category}.\{\text{v}\} \text{ E} \\ \hookrightarrow \text{D} \end{array}$ )
  ( $\begin{array}{c} \_ : \text{Category}.\{\text{v}\} \text{ C} \\ \_ : \text{Category}.\{\text{v}\} \text{ D} \end{array}$ ) (i : C ≅ D)
  ( $\begin{array}{c} \_ : \text{Faithful} \ i \\ \_ : \text{Faithful} \ j \end{array}$ )
  IsEmpty (Reflective2 i)  $\wedge$  Nonempty (Reflective2 (i » j)) := by
  sorry

```

**Problem 31** (Reflective - High). — *Theorem: Neither  $\mathcal{S}et$  nor  $\mathcal{T}op$  has a proper isomorphism-closed full subcategory that is both reflective and coreflective.*

```

import Mathlib
open CategoryTheory

theorem not_reflective_and_coreflective (P : ObjectProperty (Type u))
  (h : P.IsClosedUnderIsomorphisms) (hproper :  $\exists X : \text{Type } u, \neg P X$ ) :
  IsEmpty (Reflective P.i)  $\vee$  IsEmpty (Coreflective P.i) := by
  sorry

```

**Problem 32** (Reflective - High). — *Theorem: Set has precisely three full, isomorphism-closed, reflective subcategories.*

```
import Mathlib

open CategoryTheory Functor Limits

namespace CAT_statement_S_0032

def IsIsoClosed (P : Type u → Prop) : Prop :=
  ∀ {X Y : Type u}, Nonempty (X ≈ Y) → P X → P Y

def SubcategoryEquiv (P Q : Type u → Prop) : Prop :=
  ∀ X, P X ↔ Q X

def IsReflectiveSubcategory (P : Type u → Prop) : Prop :=
  Nonempty (Reflective (ObjectProperty.ι P))

theorem Set_has_precisely_three_reflective_subcategories :
  ∃ (P₁ P₂ P₃ : Type u → Prop),
    IsIsoClosed P₁ ∧ IsReflectiveSubcategory P₁ ∧
    IsIsoClosed P₂ ∧ IsReflectiveSubcategory P₂ ∧
    IsIsoClosed P₃ ∧ IsReflectiveSubcategory P₃ ∧
    ¬ SubcategoryEquiv P₁ P₂ ∧ ¬ SubcategoryEquiv P₂ P₃ ∧ ¬
    ¬ SubcategoryEquiv P₁ P₃ ∧
    ∀ (Q : Type u → Prop), IsIsoClosed Q → IsReflectiveSubcategory Q →
      (SubcategoryEquiv Q P₁ ∨ SubcategoryEquiv Q P₂ ∨ SubcategoryEquiv Q
       → P₃) := by
  sorry

end CAT_statement_S_0032
```

**Problem 33** (Reflective - High). — *Theorem: Top<sup>CH</sup> has precisely two full, isomorphism-closed, coreflective subcategories.*

```
import Mathlib

open CategoryTheory Topology

namespace CAT_statement_S_0033

structure FullCoreflectiveSubcategory (C : Type u) [Category.{v} C] where
  obj : ObjectProperty C
  iso_closed : obj.IsClosedUnderIsomorphisms
  coreflective : Coreflective obj.ι

theorem CompHaus_has_precisely_two_coreflective_subcategories :
  Nat.card (FullCoreflectiveSubcategory CompHaus) = 2 := by
  sorry

end CAT_statement_S_0033
```

**Problem 34** (Concrete - Medium). — *Definition: Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A **universal arrow** over  $x \in \mathcal{B}$  is a structured arrow  $u : x \rightarrow U(c)$  with domain  $x$  that has the following universal property: for each structured arrow  $f : x \rightarrow U(b)$  with domain  $x$  there exists a unique morphism  $\underline{f} : c \rightarrow b$  such that  $\underline{f} \circ u = f$ .*

*Definition:* Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A **free object** over  $x \in \mathcal{B}$  is an object  $c \in \mathcal{C}$  such that there exists a universal arrow  $(u, c)$  over  $x$ .

*Theorem:* Let  $(\mathcal{C}, U)$  be a construct such that  $U$  is representable by an object  $x$ . Then for any set  $I$  and any object  $d \in \mathcal{C}$  the following conditions are equivalent:

1.  $d$  is a free object over  $I$ .
2.  $d$  is an  $I$ -th copower of  $x$ .

```
import Mathlib

open CategoryTheory Limits Functor Opposite

namespace CAT_statement_S_0034

variable {C : Type u} [Category.{v} C]

def IsFreeObject (U : C → Type v) (d : C) (I : Type v) : Prop :=
  ∃ (η : I → U.obj d), ∀ {y : C} (f : I → U.obj y), ∃! (g : d → y), U.map g ◦
  η = f

def IsCopower (x d : C) (I : Type v) : Prop :=
  ∃ (ι : I → (x → d)), Nonempty (IsColimit (Cofan.mk d ι))

theorem free_iff_copower_ofRepresentable
  (U : C → Type v) [Faithful U]
  (x : C) (hU : U ≈ coyoneda.obj (op x))
  (I : Type v) (d : C) :
  IsFreeObject U d I ↔ IsCopower x d I := by
  sorry

end CAT_statement_S_0034
```

**Problem 35** (Concrete - High). — *Definition:* A full concrete embedding is called a realization.

*Theorem:* There is a construct  $(\mathcal{C}, U)$  such that every construct has a realization to  $(\mathcal{C}, U)$ .

```
import Mathlib

open CategoryTheory

namespace CAT_statement_S_0035

structure Construct where
  C : Type u
  [str : Category.{v} C]
  U : C → Type u
  [faithful : Functor.Faithful U]

attribute [instance] Construct.str Construct.faithful

def IsRealization (S T : Construct.{u, v}) (F : S.C → T.C) : Prop :=
  F ≫ T.U = S.U ∧ Functor.Full F ∧ Function.Injective F.obj

theorem exists_universal_construct :
```

```

 $\exists (T : \text{Construct}.\{u, v\}), \forall (S : \text{Construct}.\{u, v\}), \exists (F : S.C \Rightarrow T.C),$ 
 $\quad \hookrightarrow \text{IsRealization } S T F := \text{by}$ 
 $\quad \text{sorry}$ 

end CAT_statement_S_0035

```

**Problem 36** (Concrete - High). — *Definition: A category  $\mathcal{C}$  is called **concretizable** over a category  $\mathcal{B}$  if there exists a faithful functor  $U : \mathcal{C} \rightarrow \mathcal{B}$ .*

*Theorem: There exist categories that are not concretizable over  $\mathcal{S}\text{et}$ .*

```

import Mathlib

open CategoryTheory

theorem exists_category_not_concretizable_over_Type :
   $\exists (C : \text{Type } u) (- : \text{Category}.\{v\} C), \neg \exists (F : C \Rightarrow \text{Type } v), F.Faithful :=$ 
   $\quad \hookrightarrow \text{by}$ 
   $\quad \text{sorry}$ 

```

**Problem 37** (Concrete - High). — *Theorem: There are precisely two concrete functors from  $\mathcal{S}\text{et}$  to  $\mathcal{T}\text{op}$ , but a proper class of concrete functors from  $\mathcal{T}\text{op}$  into itself.*

```

import Mathlib

open CategoryTheory

namespace CAT_statement_S_0037

universe u v w

variable {X : Type uX} [Category.{vX} X]

structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C  $\Rightarrow$  X
  [U_Faithful : U.Faithful]

attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

def IsConcreteFunc {A B : ConcreteCat (X := X)} (F : A.C  $\Rightarrow$  B.C) : Prop :=
  Nonempty ((F » B.U)  $\cong$  A.U)

def SetConcrete : ConcreteCat (X := Type u) :=
  { C := Type u
    U := 1 (Type u) }

def TopConcrete : ConcreteCat (X := Type u) :=
  { C := TopCat.{u}
    U := (forget TopCat) }

```

```

def ConcreteFuncsIso (A B : ConcreteCat (X := Type u)) : Type _ :=
{ F : A.C => B.C // IsConcreteFunc (A := A) (B := B) F }

theorem only_two_concrete_functors_from_Set_to_Top_iso :
  Nat.card (ConcreteFuncsIso SetConcrete TopConcrete) = 2 := by
  sorry

end CAT_statement_S_0037

```

**Problem 38** (Concrete - High). — *Definition:* Let  $\mathcal{C}$  be a category and let  $c \in \mathcal{C}$  be an object. A **regular subobject** of  $c$  is a pair  $(x, i)$  where  $i$  is a regular monomorphism.

*Definition:* Let  $\mathcal{C}$  be a category.  $\mathcal{C}$  is called **regular wellpowered** if no object in  $\mathcal{C}$  has a proper class of pairwise non-isomorphic regular subobjects.

*Definition:* A category  $\mathcal{C}$  is called **concretizable** over a category  $\mathcal{B}$  if there exists a faithful functor  $U : \mathcal{C} \rightarrow \mathcal{B}$ .

*Theorem:* Let  $\mathcal{C}$  be a category that admits finite limits. Then  $\mathcal{C}$  is concretizable over  $\mathcal{S}\text{et}$  if and only if  $\mathcal{C}$  is regular wellpowered.

```

import Mathlib

namespace CAT_statement_S_0038

open CategoryTheory Limits

universe u v w

variable {C : Type u} [Category C] [HasFiniteLimits C]

def IsConcretizable (X : Type v) [Category X] (D : Type u) [Category D] : Prop
  := ∃ U : D ≈ X, U.Faithful

variable (C)

class RegularWellPowered : Prop where
  regularSubobject_small : ∀ (X : C), Small.{v} {P : Subobject X} //
  Nonempty (RegularMono P.arrow) }

theorem concretizable_iff_regular_wellpowered :
  IsConcretizable (Type u) C ↔ RegularWellPowered C := by
  sorry

end CAT_statement_S_0038

```

**Problem 39** (Concrete - High). — *Theorem:* Let  $\mathcal{F}\text{rm}$  be the construct whose objects are frames, i.e. distributive suplattices, and whose morphisms are frame homomorphisms. Then there is a unique concrete functor  $T : \mathcal{T}\text{op}_0^{\text{op}} \rightarrow \mathcal{F}\text{rm}$  over  $\mathcal{S}\text{et}$ , where  $\mathcal{T}\text{op}_0$  is the category of  $T_0$  topological spaces.

```

import Mathlib

open CategoryTheory Topology

universe u v w

```

```

variable {X : Type uX} [Category.{vX} X]

namespace CAT_statement_S_0039

structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C → X
  [U_Faithful : U.Faithful]

attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

def IsConcreteFunc {A B : ConcreteCat (X := X)} (F : A.C → B.C) : Prop :=
  Nonempty ((F » B.U) ≈ A.U)

def forgetFrm : Frm.{u} → Type u where
  obj X := X
  map {X Y} f := f

instance : forgetFrm.Faithful where
  map_injective {X Y} f g h := by
    ext x
    simpa using congrArg (fun k => k x) h

structure T0TopCat where
  toTop : TopCat.{u}
  is_t0 : T0Space (↑toTop)

namespace T0TopCat

instance : CoeSort T0TopCat (Type u) := <fun X => X.toTop>
instance (X : T0TopCat) : TopologicalSpace X := X.toTop.str
attribute [instance] T0TopCat.is_t0

instance : Category T0TopCat :=
  InducedCategory.category (fun X : T0TopCat => X.toTop)

def forget_0 : T0TopCat → TopCat :=
  inducedFunctor (fun X : T0TopCat => X.toTop)

instance : forget_0.Faithful :=
{ map_injective := by
  intro X Y f g h
  simpa [forget_0] using h }

@[simp] def of (X : Type u) [TopologicalSpace X] [T0Space X] : T0TopCat :=
  <TopCat.of X, inferInstance>

```

```

def L : T0TopCatop → Type u :=
{ obj := fun X ⇒ TopologicalSpace.Opens ((X.unop).toTop)
  map := by
    intro ⟨X⟩ ⟨Y⟩ ⟨f⟩
    obtain ⟨g⟩ : (Y.toTop → X.toTop) := forget_0.map f
    intro U
    exact TopologicalSpace.Opens.comap g U
  map_id := by
    intro X
    ext U x
    rfl
  map_comp := by
    intro X Y Z f g
    ext U x
    rfl }

instance : L.Faithful where
  map_injective {X Y} f g h := by
    apply Quiver.Hom.unop_inj
    apply Functor.map_injective T0TopCat.forget_0
    ext x
    haveI : T0Space (T0TopCat.forget_0.obj (Opposite.unop X)) :=
      (Opposite.unop X).is_t0
    apply Inseparable.eq
    rw [inseparable_iff_forall_isOpen]
    intro U hU
    let U_op : TopologicalSpace.Opens (T0TopCat.forget_0.obj (Opposite.unop
      ← X)) := ⟨U, hU⟩
    have h_eq := congr_fun h U_op
    dsimp [L] at h_eq
    have h_set := congr_arg (SetLike.coe) h_eq
    rw [Set.ext_iff] at h_set
    exact h_set x

end T0TopCat

def FrmConcrete : ConcreteCat (X := Type u) :=
{ C := Frm.{u}
  U := (forgetFrm) }

def T0TopCatopConcrete : ConcreteCat (X := Type u) :=
{ C := T0TopCatop
  U := (T0TopCat.L) }

def ConcreteFuncsIso (A B : ConcreteCat (X := Type u)) : Type _ :=
{ F : A.C → B.C // IsConcreteFunc (A := A) (B := B) F }

theorem unique_concrete_functors_from_T0TopCatop_to_Frm_iso :
  Nat.card (ConcreteFuncsIso T0TopCatConcrete FrmConcrete) = 1 := by
  sorry

end CAT_statement_S_0039

```

**Problem 40** (Concrete - High). — *Definition: Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A **universal arrow** over  $x \in \mathcal{B}$  is a structured arrow  $u : x \rightarrow U(c)$  with domain  $x$  that has the following universal property: for each structured arrow  $f : x \rightarrow U(b)$  with domain  $x$  there exists a unique morphism  $\underline{f} : c \rightarrow b$  such that  $\underline{f} \circ u = f$ .*

*Definition: Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A **free object** over  $x \in \mathcal{B}$  is an object  $c \in \mathcal{C}$  such that there exists a universal arrow  $(u, c)$  over  $x$ .*

*Definition: Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ .  $\mathcal{C}$  is said to **have free objects**, if for each  $x \in \mathcal{B}$  there is a free object over  $x$ .*

*Theorem: Let  $\mathcal{L}\text{at}_\vee^\infty$  be the category of suplattices. The construct  $\mathcal{L}\text{at}_\vee^\infty$  has free objects.*

```
import Mathlib
open CategoryTheory
namespace CAT_statement_S_0040
universe u v w
variable {X : Type uX} [Category.{vX} X]
structure ConcreteCat (X : Type v) [Category X] where
  C : Type u
  [cat : Category C]
  U : C → X
  [U_Faithful : U.Faithful]
attribute [instance] ConcreteCat.cat ConcreteCat.U_Faithful

abbrev StructuredArrowOver (x : X) (C : ConcreteCat (X := X)) : Type _ :=
StructuredArrow x C.U

def IsUniversalArrowOver (x : X) {C : ConcreteCat (X := X)} (u :
  StructuredArrowOver x C) : Prop :=
  ∃ (v : StructuredArrowOver x C),
    ∃! (g : u.right → v.right), u.hom ≫ C.U.map g = v.hom

def IsFreeObjectOver (x : X) {C : ConcreteCat (X := X)} (z : C.C) : Prop :=
  ∃ (f : StructuredArrowOver x C), f.right = z ∧ IsUniversalArrowOver (x :=
  x) (C := C) f

def HasFreeObject (C : ConcreteCat (X := X)) : Prop :=
  ∃ (x : X), ∃ (z : C.C), IsFreeObjectOver (x := x) (z := z)

structure SupLatCat where
  carrier : Type u
  [inst : CompleteSemilatticeSup carrier]

attribute [instance] SupLatCat.inst

instance : CoeSort SupLatCat (Type u) := <SupLatCat.carrier>
```

```

def of (α : Type u) [CompleteSemilatticeSup α] : SupLatCat := <α>

structure Hom (A B : SupLatCat.{u}) where
  toFun : A → B
  map_sSup' : ∀ s : Set A, toFun (sSup s) = sSup (toFun ` ` s)

instance (A B : SupLatCat) : CoeFun (Hom A B) (fun _ ⇒ A → B) := <Hom.toFun>

@[simp] lemma Hom.map_sSup {A B : SupLatCat} (f : Hom A B) (s : Set A) :
  f (sSup s) = sSup (f ` ` s) := f.map_sSup' s

@[ext] lemma Hom.ext {A B : SupLatCat} {f g : Hom A B}
  (h : ∀ a, f a = g a) : f = g := by
  cases f with
  | mk fto fmap =>
    cases g with
    | mk gto gmap =>
      have hto : fto = gto := funext (by intro a; exact h a)
      cases hto
      have : fmap = gmap := by
        apply Subsingleton.elim
      cases this
      rfl

def id (A : SupLatCat) : Hom A A :=
{ toFun := (_root_.id : A → A)
  map_sSup' := by
    intro s
    simp }

def comp {A B C : SupLatCat} (f : Hom A B) (g : Hom B C) : Hom A C :=
{ toFun := fun a => g (f a)
  map_sSup' := by
    intro s
    calc
      g (f (sSup s)) = g (sSup (f ` ` s)) := by
        simp
      _ = sSup (g ` ` (f ` ` s)) := by
        simp
      _ = sSup ((fun x => g (f x)) ` ` s) := by
        simp [Set.image_image] }

instance : Category SupLatCat where
  Hom A B := Hom A B
  id A := id A
  comp f g := comp f g
  id_comp := by intro A B f; ext a; rfl
  comp_id := by intro A B f; ext a; rfl
  assoc := by intro A B C D f g h; ext a; rfl

```

```

def forget : SupLatCat → Type u :=
{ obj := fun A ⇒ A.carrier
  map := fun {X Y} (f : X → Y) ⇒ f.toFun
  map_id := by intro A; rfl
  map_comp := by intro A B C f g; rfl }

instance : forget.Faithful where
  map_injective := by
    intro X Y f g h
    apply Hom.ext
    intro x
    simpa using congrArg (fun k ⇒ k x) h

def SupLatCatConcrete : ConcreteCat (X := Type u) :=
{ C := SupLatCat.{u}
  U := (forget) }

theorem SupLat_Has_Free_Object :
  HasFreeObject SupLatCatConcrete := by
  sorry

end CAT_statement_S_0040

```

**Problem 41** (Concrete - High). — *Definition:* Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A **universal arrow** over  $x \in \mathcal{B}$  is a structured arrow  $u : x \rightarrow U(c)$  with domain  $x$  that has the following universal property: for each structured arrow  $f : x \rightarrow U(b)$  with domain  $x$  there exists a unique morphism  $\underline{f} : c \rightarrow b$  such that  $\underline{f} \circ u = f$ .

*Definition:* Let  $(\mathcal{C}, U)$  be a concrete category over  $\mathcal{B}$ . A **free object** over  $x \in \mathcal{B}$  is an object  $c \in \mathcal{C}$  such that there exists a universal arrow  $(u, c)$  over  $x$ .

*Theorem:* Let  $\mathcal{C}$  be the non-full subcategory of  $\mathcal{L}\text{at}_Y^\infty$  whose objects are suplattice and morphisms are meet-and join-preserving maps. In the construct  $\mathcal{C}$ , there exists a free object over  $x$  if and only if the cardinality of  $x$  is not greater than 2, i.e.  $|x| \leq 2$ .

```

import Mathlib

open CategoryTheory

universe u v w

namespace CAT_statement_S_0041

structure FreeObject {C : Type u} [Category.{v} C] [HasForget.{w} C] (x :
  Type w) where
  (obj : C)
  (emb : x → (forget C).obj obj)
  (uniq : ∀ (Y : C) (f : x → (forget C).obj Y), ∃! (g : obj → Y), emb »
    (forget C).map g = f)

```

```

lemma complete_lattice_category (X : Type u) :
  Nonempty (FreeObject (C := CompleteLat) X) ↔ Cardinal.mk X ≤ 2 := by
  sorry

end CAT_statement_S_0041

```

**Problem 42** (Limit - Easy). — *Theorem: Let  $\mathcal{C}$  and  $\mathcal{D}$  be categories and let  $F : \mathcal{C} \rightarrow \mathcal{D}$  be a fully faithful functor. Then  $F$  reflects any limits and colimits admitted in the codomain category.*

```

import Mathlib

open CategoryTheory Limits Functor

variable {C : Type u₁} [Category.{v₁} C] {D : Type u₂} [Category.{v₂} D]

lemma fully_faithful_reflects_limits_and_colimits (F : C → D) [Full F]
  ↵ [Faithful F] :
  ReflectsLimits F ∧ ReflectsColimits F := by
  sorry

```

**Problem 43** (Limit - Easy). — *Theorem: The one point set  $\{*\}$  form a separator in Set, and the two point set  $\{a, b\}$  form a coseparating set in Set.*

```

import Mathlib

open CategoryTheory Function Classical

theorem PUnit_isSeparator : IsSeparator (PUnit : Type u) := by
  sorry

theorem ULiftBool_isCoseparator : IsCoseparator (ULift.{u} Bool) := by
  sorry

```

**Problem 44** (Limit - Easy). — *Theorem: Filtered colimits commute with finite limits in Set.*

```

import Mathlib

open CategoryTheory Limits

variable {J : Type u} [SmallCategory J] [FinCategory J]
variable {K : Type u} [SmallCategory K] [IsFiltered K]
variable (F : J → K → Type u)

theorem filteredColimitsCommuteWithFiniteLimits :
  Nonempty (colimit (limit F) ≈ limit (colimit F.flip)) := by
  sorry

```

**Problem 45** (Limit - Easy). — *Theorem: Let  $\omega$  be the ordinal of natural numbers. Consider  $F : \omega^{op} \rightarrow \mathcal{R}\text{ing}$  with  $F_n := \mathbb{Z}/p^n\mathbb{Z}$  and  $f_n : F_{n+1} \rightarrow F_n$ . Then the limit exists.*

```

import Mathlib

open CategoryTheory Limits Opposite

variable (p : ℕ)

noncomputable def pAdicFunctor : ℕop → RingCat where
  obj n := RingCat.of (ZMod (p ^ (unop n)))
  map {m n} f := RingCat.ofHom <|
    ZMod.castHom (pow_dvd_pow p (leOfHom f.unop)) (ZMod (p ^ (unop n)))
  map_id := by
    intro n
    ext x
    simp
  map_comp := by
    intro x y z f g
    ext x
    simp
  theorem pAdic_limit_exists : HasLimit (pAdicFunctor p) := by
    sorry

```

**Problem 46** (Limit - Easy). — *Theorem: Let  $\mathcal{C}$  and  $\mathcal{D}$  be a small category and let  $F, G : \mathcal{C} \rightarrow \mathcal{D}$  be two functors. Then we have  $\text{Nat}(F, G) \cong \int_{c \in \mathcal{C}} \text{hom}_{\mathcal{D}}(F(c), G(c))$ .*

```

import Mathlib

open CategoryTheory Limits

variable {C : Type u} [SmallCategory C]
variable {D : Type u} [SmallCategory D]
variable (F G : C → D)

def homIntegrandBifunctor : Cop × C → Type u :=
  (Functor.prod F.op G) » (Functor.hom D)

theorem natTransIsoEnd :
  Nonempty (NatTrans F G ≈ end_ (curryObj (homIntegrandBifunctor F G))) := 
  by
  sorry

```

**Problem 47** (Limit - Easy). — *Theorem: There is no equivalence of categories between  $\text{Set}$  and  $\text{Set}^{op}$ .*

```

import Mathlib

open CategoryTheory

theorem no_equiv_between_Set_and_op : ¬ Nonempty (Equivalence (Type u) (Type
  ↵ u)op) := by
  sorry

```

**Problem 48** (Limit - Easy). — *Theorem: A reflective subcategory  $\mathcal{C}$  of a cocomplete category  $\mathcal{D}$  is also cocomplete.*

```

import Mathlib
open CategoryTheory Limits

variable {C : Type u} [Category.{v} C] {D : Type u} [Category.{v} D]

theorem hasColimits_of_reflective (i : C → D) [Reflective i] [HasColimits D]
  ↵ :
  HasColimits C := by
  sorry

```

**Problem 49** (Limit - Medium). — *Theorem: Let  $\mathcal{C}$  and  $\mathcal{E}$  be two categories and let  $F : \mathcal{C} \rightarrow \mathcal{E}$  be a functor. Let  $\bullet$  be the terminal category consisting of a unique object  $\bullet$  and a unique morphism. Then a colimit of  $F$  is a left Kan extension of  $F$  along  $K : \mathcal{C} \rightarrow \bullet$ , i.e.  $\text{Lan}_K F(\bullet) = \text{colim } F$ .*

```

import Mathlib
open CategoryTheory Limits

universe u₁ v₁ u₂ v₂

variable {C : Type u₁} [Category.{v₁} C]
variable {E : Type u₂} [Category.{v₂} E]

theorem colimit_is_leftKanExtension_along_to_terminal
  (F : C → E) (K : C → PUnit) [HasColimit F] [K.HasLeftKanExtension F] :
  Nonempty ((K.leftKanExtension F).obj PUnit.unit ≈ colimit F) := by
  sorry

```

**Problem 50** (Limit - Medium). — *Definition: Let  $\mathcal{C}$  be a locally small category. An object  $c \in \mathcal{C}$  is called compact if  $\text{hom}_{\mathcal{C}}(c, -)$  preserves filtered colimits.*

*Theorem: For  $\mathcal{S}$ et, an object is compact if and only if it is a finite set.*

```

import Mathlib
open CategoryTheory Limits

lemma isCompactObject_iff_finite_type (X : Type u) :
  PreservesFilteredColimits (coyoneda.obj (Opposite.op X)) ↔ Finite X := by
  sorry

```

**Problem 51** (Limit - Medium). — *Theorem: Let  $\mathcal{C}$  be a category. Then  $\mathcal{C}$  admits all small limits if and only if  $\mathcal{C}$  admits all small products and pullbacks.*

```

import Mathlib
open CategoryTheory Limits

variable {C : Type u} [Category.{v} C]

```

```

theorem has_limits_iff_has_products_and_pullbacks :
  HasLimitsOfSize.{v, v} C ↔ (∀ (J : Type v), HasLimitsOfShape (Discrete J)
    → C) ∧ HasLimitsOfShape WalkingCospan C := by
  sorry

```

**Problem 52** (Limit - Medium). — *Theorem: Let  $X, Y, Z$  be objects in  $\mathcal{S}\text{et}$  with morphisms  $f : X \rightarrow Z$  and  $g : Y \rightarrow Z$ . Then  $\{(x, y) \in X \times Y \mid f(x) = g(y)\}$  is the pullback  $X \times_Z Y$  of  $X$  and  $Y$  over  $Z$ .*

```

import Mathlib

open CategoryTheory Limits Functor Types Function Pullback

theorem Function.isPullback_pullback {X Y Z : Type u} (f : X → Z) (g : Y →
  Z) :
  IsPullback (C := Type u) (fst (f := f) (g := g)) snd f g := by
  sorry

```

**Problem 53** (Limit - Medium). — *Theorem: Let  $\mathcal{D}$  be a small-complete locally small category, a functor  $G : \mathcal{D} \rightarrow \mathcal{C}$  has a left adjoint if and only if  $G$  is continuous and for each  $c \in \mathcal{C}$ , the comma category  $(c \downarrow G)$  admits an initial object.*

```

import Mathlib

open CategoryTheory Limits

variable {D : Type u} [Category.{v} D] [HasLimits D] [LocallySmall.{v} D]
variable {C : Type u} [Category.{v} C]
variable (G : D ⇒ C)

theorem has_left_adjoint_iff_continuous_and_initials :
  G.IsRightAdjoint ↔ PreservesLimits G ∧ ∀ (c : C), HasInitial
    → (StructuredArrow c G) := by
  sorry

```

**Problem 54** (Limit - Medium). — *Theorem: Let  $\mathcal{B}$  be a complete category. Then  $\mathcal{B}$  has an initial object if and only if there exists a small set  $I$  and an  $I$ -indexed family of objects  $x_i$  such that, for every  $s \in \mathcal{B}$ , there is an  $i \in I$  and an arrow  $x_i \rightarrow s$ .*

```

import Mathlib

open CategoryTheory Limits

variable {B : Type u} [Category.{v} B]

theorem hasInitial_iff_exists_weakly_initial [HasLimits B] :
  HasInitial B ↔ ∃ (I : Type v) (x : I → B), ∀ (s : B), ∃ (i : I), Nonempty
    (x i → s) := by
  sorry

```

**Problem 55** (Limit - Medium). — *Theorem: The forgetful functor  $U : \mathcal{G}\text{rp}, \mathcal{A}\text{b}, \mathcal{R}\text{ing} \rightarrow \mathcal{S}\text{et}$  creates limits, but they do not preserve coproducts.*

```

import Mathlib
open CategoryTheory Limits

theorem forget_GrpCreatesLimits_but_not_coproducts :
  Nonempty (CreatesLimits (forget Grp.{u})) ∧
  ¬ Nonempty (PreservesColimitsOfShape (Discrete Bool) (forget Grp.{u})) :=
  -- by
  sorry

theorem forget_AbCreatesLimits_but_not_coproducts :
  Nonempty (CreatesLimits (forget Ab.{u})) ∧
  ¬ Nonempty (PreservesColimitsOfShape (Discrete Bool) (forget Ab.{u})) :=
  -- by
  sorry

theorem forget_RingCatCreatesLimits_but_not_coproducts :
  Nonempty (CreatesLimits (forget RingCat.{u})) ∧
  ¬ Nonempty (PreservesColimitsOfShape (Discrete Bool) (forget
    -- RingCat.{u})) := by
  sorry

```

**Problem 56** (Limit - Medium). — *Theorem: Let  $\mathcal{C}$  and  $\mathcal{D}$  be categories and let  $G : \mathcal{D} \rightarrow \mathcal{C}$  be a functor. Then  $G : \mathcal{D} \rightarrow \mathcal{C}$  has a left adjoint if and only if the right Kan extension  $\text{Ran}_G \text{Id}_{\mathcal{D}} : \mathcal{C} \rightarrow \mathcal{D}$  exists and is preserved by  $G$  (i.e.  $G \circ \text{Ran}_G \text{Id}_{\mathcal{D}} \simeq \text{Ran}_K(G \circ \text{Id}_{\mathcal{D}})$ ).*

```

import Mathlib
open CategoryTheory Functor

variable {C : Type u₁} [Category.{v₁} C] {D : Type u₂} [Category.{v₂} D]

theorem hasLeftAdjoint_iff_ran_id_preserved (G : D ≈ C) :
  G.IsRightAdjoint ↔
  ∃ (R : C ≈ D) (α : G ≫ R → 1 D),
  R.IsRightKanExtension α ∧
  (R ≫ G).IsRightKanExtension ((associator G R G).inv ≫ whiskerRight α G
    -- > (leftUnit G).hom) := by
  sorry

```

**Problem 57** (Limit - Medium). — *Theorem: A functor that reflects equalizers (or finite products) reflects isomorphisms.*

```

import Mathlib
open CategoryTheory Limits

variable {C : Type u} [Category.{v} C] {D : Type u'} [Category.{v'} D]

theorem reflectsIsomorphisms_of_reflects_equalizers (F : C ≈ D)
  [ReflectsLimitsOfShape WalkingParallelPair F] : F.ReflectsIsomorphisms :=
  -- by
  sorry

```

```

theorem reflectsIsomorphisms_of_reflects_finite_products (F : C → D)
  [ReflectsLimitsOfShape (Discrete PEmpty) F] [ReflectsLimitsOfShape
    ↪ (Discrete WalkingPair) F] :
  F.ReflectsIsomorphisms := by
  sorry

```

**Problem 58** (Limit - Medium). — *Definition:* Let  $\mathcal{C}$  be a locally small category. An object  $c \in \mathcal{C}$  is called **compact** if  $\text{hom}_{\mathcal{C}}(c, -)$  preserves filtered colimits.

*Theorem:* A topological space  $X$  is compact if and only if it is a compact object in the category  $\mathcal{O}p(X)$ , the category of open subsets of  $X$ .

```

import Mathlib
open CategoryTheory
namespace CAT_statement_S_0058
universe u
variable (X : Type u) [TopologicalSpace X]
abbrev Op (X : Type u) [TopologicalSpace X] := TopologicalSpace.Opens X
theorem compactSpace_iff_finitelyPresented_top :
  CompactSpace X ↔ IsFinitelyPresentable (C := Op X) (T : Op X) := by
  sorry
end CAT_statement_S_0058

```

**Problem 59** (Limit - Medium). — *Definition:* A functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  is said to **lift limits** if for every diagram  $D : \mathcal{I} \rightarrow \mathcal{C}$  and every limit  $L$  of  $F \circ D$ , there exists a limit  $L' \in \mathcal{D}$  such that  $F(L') \cong L$ .

*Theorem:* There is a functor that lifts limits but is not faithful.

```

import Mathlib
open CategoryTheory Limits
namespace CAT_statement_S_0059
universe w' w'₁ w w₁ v₁ v₂ v₃ u₁ u₂ u₃
variable {C : Type u₁} [Category.{v₁} C]
variable {D : Type u₂} [Category.{v₂} D]
variable {J : Type w} [Category.{w'} J] {K : J ⇒ C}
structure LiftableCone₂ (K : J ⇒ C) (F : C ⇒ D) (c : Cone (K ≫ F)) where
  liftedCone : Cone K
  validLift : F.mapCone liftedCone ≈ c
  isLimit : IsLimit liftedCone

```

```

class LiftsLimit (K : J → C) (F : C → D) where
  lifts : ∀ c, IsLimit c → LiftableCone₂ K F c

theorem exists_functor_lifts_limit_and_not_faithful :
  ∃ (C : Type (u₁+1)) (_ : Category.{u₁} C) (D : Type u₂) (_ :
    ↪ Category.{u₂} D) (F : C → D), (∀ (J : Type u₁) (_ : Category.{w'} J)
    ↪ (K : J → C), Nonempty (LiftsLimit K F)) ∧
  F.Faithful := by
  sorry

end CAT_statement_S_0059

```

**Problem 60** (Limit - Medium). — *Theorem: Suppose  $\mathcal{B}$  is locally small, complete, has a small coseparating set  $S$ , and has the property that every family of subobjects has an intersection. Then  $\mathcal{B}$  has an initial object.*

```

import Mathlib

open CategoryTheory Limits

theorem has_initial_of_locally_small_complete_coseparating {B : Type u}
  [Category.{v} B]
  [LocallySmall.{w} B] [HasLimitsOfSize.{w, w} B] {S : Set B} [Small.{w} S]
  (hS : IsCoseparating S) (h : ∀ (A : B), ∀ (s : Set (Subobject A)), ∃ (f :
    ↪ Subobject A),
  IsGLB s f) : HasInitial B := by
  sorry

```

**Problem 61** (Limit - Medium). — *Theorem: Let  $\mathcal{C} = \mathcal{D} = \text{Vec}_{\mathbb{k}}$  the category of finite dimension  $\mathbb{k}$ -vector spaces. Then the coend is the trace of matrices.*

```

import Mathlib

open CategoryTheory Limits

theorem coend_hom_is_trace_of_matrices
  (k : Type u) [Field k] :
  ∀ (F : (ModuleCat k)ᵒᵖ → ModuleCat k → ModuleCat k),
  (∀ X Y, (F.obj (Opposite.op X)).obj Y ≈ ModuleCat.of k (X →ₗ[k] Y)) →
  ∃ (T : ModuleCat k),
  (∃ (tr : ∀ X, (F.obj (Opposite.op X)).obj X → T),
  Nonempty (IsColimit (Cofan.mk T tr))) := by
  sorry

```

**Problem 62** (Limit - Medium). — *Definition: Let  $\mathcal{C}$  be a category. Let  $S$  be a family of subobjects  $(s_n, i_n)$  of an object  $c \in \mathcal{C}$ , indexed by a class  $I$ . A subobject  $(x, i : x \rightarrow c)$  of  $c$  is called an **intersection** of  $S$  provided that the following two conditions are satisfied:*

- (1)  $i$  factors through each  $i_n$  i.e., for each  $n$  there exists an  $f_n : x \rightarrow s_n$  with  $i = i_n \circ f_n$ ,
- (2) if a morphism  $f : z \rightarrow c$  factors through each  $i_n$ , then it factors through  $i$ .

*Definition: A category  $\mathcal{C}$  is said to have **intersections** if for each object  $c \in \mathcal{C}$  and every family of subobjects of  $c$ , there exists an intersection.*

*Definition: A category is said to be **strongly complete** if it is complete and has intersections.*

*Theorem: A strongly cocomplete category with a separating set is strongly complete.*

```

import Mathlib
open CategoryTheory Limits

namespace CAT_statement_S_0062

universe u v
variable {C : Type u} [Category.{v} C]

def IsIntersectionOf {B : C} (A : Subobject B) (S : Set (Subobject B)) : Prop
← :=
(∀ Ai, Ai ∈ S → A ≤ Ai) ∧
(∀ A' : Subobject B, (∀ Ai, Ai ∈ S → A' ≤ Ai) → A' ≤ A)

def HasIntersections (C : Type u) [Category.{v} C] : Prop :=
∀ (B : C) (S : Set (Subobject B)),
∃ A : Subobject B, IsIntersectionOf (C := C) (B := B) A S

class StronglyComplete (C : Type u) [Category.{v} C] : Prop where
complete: HasLimits C
hasinter: HasIntersections C

class StronglyCocomplete (C : Type u) [Category.{v} C] : Prop where
dual: StronglyComplete (C:=Cop)

theorem strongly_complete_of_strongly_cocomplete_of_separating_set
← [StronglyComplete Cop] {G : Set C} [Small.{v} G] (hG : IsSeparating G) :
StronglyComplete C := by
sorry

end CAT_statement_S_0062

```

**Problem 63 (Limit - High).** — *Definition: Let  $\mathcal{C}$  be a locally small category. An object  $c \in \mathcal{C}$  is called **compact** if  $\text{hom}_{\mathcal{C}}(x, -)$  preserves filtered colimits.*

*Theorem: For  $\mathcal{G}\text{rp}$ , an object is compact if and only if it is finitely presented as a group. Every group can be realized as a direct limit of finitely presented groups.*

```

import Mathlib
open CategoryTheory

namespace CAT_statement_S_0063

universe u

def IsFinitelyPresentedGrp (X : Type u) [Group X] : Prop :=
∃ (α : Type u) (rels : Set (FreeGroup α)), Finite α ∧ rels.Finite ∧
Nonempty (X ≈* PresentedGroup rels)

theorem isCompactObject_Grp_iff_finite_presented (X : Type u) [Group X] :
CategoryTheory.IsFinitelyPresentable (Grp.of X) ↔ IsFinitelyPresentedGrp
→ X := by
sorry

```

```

theorem group_realized_as_direct_limit_of_finitely_presented_groups (X : Type
  ↵ u) [Group X] :
  ∃ (J : Type u) (inst1 : CategoryTheory.SmallCategory J) (inst2 :
    ↵ CategoryTheory.IsFiltered J) (F : CategoryTheory.Functor J Grp), ∀ (j
    ↵ : J), IsFinitelyPresentedGrp (F.obj j) ∧ Nonempty (X ≈*
    ↵ Grp.FilteredColimits.colimit F) := by
  sorry

end CAT_statement_S_0063

```

**Problem 64** (Limit - High). — *Definition: Let  $\mathcal{C}$  be a locally small category. An object  $c \in \mathcal{C}$  is called **compact** if  $\text{hom}_{\mathcal{C}}(x, -)$  preserves filtered colimits.*

*Theorem: Let  $A$  be a ring. For the category of right  $A$ -modules  $\mathcal{A}b_A$ , an object is compact if and only if it is a finitely presentable  $A$ -module. Every  $A$ -module can be realized as a direct limit of finitely presented  $A$ -module.*

```

import Mathlib

open CategoryTheory

universe u v w

theorem isCompactObject_Grp_iff_finite.presented {A : Type u} [Ring A] (X :
  ↵ Type v) [Group X] [AddCommGroup X] [Module A X] :
  ∃ CategoryTheory.IsFinitelyPresentable (ModuleCat.of A X) ↔
  ∃ Module.FinitePresentation A X := by
  sorry

theorem module_realized_as_direct_limit_of_finitely.presented_modules (A :
  ↵ Type u) [Ring A] (X : Type v) [AddCommGroup X] [Module A X] :
  ∃ (J : Type w) (inst1 : CategoryTheory.SmallCategory J) (inst2 :
    ↵ CategoryTheory.IsFiltered J) (F : CategoryTheory.Functor J (ModuleCat
    ↵ A)), ∀ (j : J), Module.FinitePresentation A (F.obj j) ∧ Nonempty (X
    ↵ ≈I[A] ModuleCat.FilteredColimits.colimit F) := by
  sorry

```

**Problem 65** (Limit - High). — *Theorem: Let  $\mathcal{C}$  be a complete, wellpowered, cowellpowered and have a separator  $s$ . Then  $\mathcal{C}$  is cocomplete if and only if for each set  $I$ , there exists an  $I$ -th copower of  $S$  in  $\mathcal{C}$ .*

```

import Mathlib

open CategoryTheory Limits

variable {C : Type u} [Category.{v} C]

theorem hasColimits_iff_hasCoprod_of_separator
  [HasLimits C]
  [WellPowered C]
  [WellPowered Cop]
  (S : C) (hS : IsSeparator S) :
  HasColimits C ↔ ∀ (I : Type v), HasColimit (Discrete.functor (fun (_ : I)
    ↵ ⇒ S)) := by
  sorry

```

**Problem 66** (Limit - High). — *Definition: Let  $\mathcal{C}$  be a category. Let  $S$  be a family of subobjects  $(s_n, i_n)$  of an object  $c \in \mathcal{C}$ , indexed by a class  $I$ . A subobject  $(x, i : x \rightarrow c)$  of  $c$  is called an **intersection** of  $S$  provided that the following two conditions are satisfied:*

(1)  *$i$  factors through each  $i_n$  i.e., for each  $n$  there exists an  $f_n : x \rightarrow s_n$  with  $i = i_n \circ f_n$ ,*

(2) *if a morphism  $f : z \rightarrow c$  factors through each  $i_n$ , then it factors through  $i$ .*

*Definition: A category  $\mathcal{C}$  is said to have **intersections** if for each object  $c \in \mathcal{C}$  and every family of subobjects of  $c$ , there exists an intersection.*

*Definition: A category is said to be **strongly complete** if it is complete and has intersections.*

*Definition: A category  $\mathcal{C}$  is strongly cocomplete if  $\mathcal{C}^{\text{op}}$  is strongly complete.*

*Theorem: There is a strongly cocomplete category with a separator that is neither wellpowered nor co-wellpowered.*

```
import Mathlib
open CategoryTheory Limits

namespace CAT_statement_S_0066

universe u v
variable {C : Type u} [Category.{v} C]

def IsIntersectionOf {B : C} (A : Subobject B) (S : Set (Subobject B)) : Prop
  := (V Ai, Ai ∈ S → A ≤ Ai) ∧
    (V A' : Subobject B, (V Ai, Ai ∈ S → A' ≤ Ai) → A' ≤ A)

def HasIntersections (C : Type u) [Category.{v} C] : Prop :=
  V (B : C) (S : Set (Subobject B)),
  ∃ A : Subobject B, IsIntersectionOf (C := C) (B := B) A S

class StronglyComplete (C : Type u) [Category.{v} C] : Prop where
  complete: HasLimits C
  hasinter: HasIntersections C

class StronglyCocomplete (C : Type u) [Category.{v} C] : Prop where
  dual: StronglyComplete (C := Cop)

theorem exists_cocomplete_separator_not_wellPowered_not_coWellPowered :
  ∃ (C : Type u) (_ : Category.{v} C),
  StronglyCocomplete C ∧ HasSeparator C ∧
  ¬ WellPowered.{v} C ∧ ¬ WellPowered.{v} Cop := by
  sorry

end CAT_statement_S_0066
```

**Problem 67** (Limit - High). — *Theorem: Let  $\omega$  be the ordinal of natural numbers. Consider  $F : \omega^{\text{op}} \rightarrow \mathcal{R}\text{ing}$  with  $F_n := k[x]/(x^n)$  and  $f_n : k[x]/(x^{n+1}) \rightarrow k[x]/(x^n)$ . Then the limit exists and is isomorphic to  $k[[x]]$ .*

```
import Mathlib
open CategoryTheory Polynomial Limits

universe u
```

```

namespace CAT_statement_S_0067

variable (k : Type u) [Field k]

noncomputable def F : Natop → RingCat :=
{
  obj := fun ⟨n⟩ => RingCat.of ((k[X] / Ideal.span {⟨X ^ n : k[X]⟩}))
  map := fun {A B} f => match A, B with
    | ⟨n⟩, ⟨m⟩ => match f with
      | <<<(f : m ≤ n)>>> =>
        RingCat.ofHom (Ideal.quotient.factor
          ↳ (Ideal.span_singleton_le_span_singleton.mpr (pow_dvd_pow X f)))
  }
}

lemma quotCommTrunc {n : ℕ} (p : k[X]) : (PowerSeries.trunc n p : k[X]) = (p
  ↵ : k[X] / Ideal.span {⟨X ^ n : k[X]⟩}) := by
  rw [Ideal.quotient.eq, Ideal.mem_span_singleton, X_pow_dvd_iff]
  intro d hd
  simp [PowerSeries.coeff_trunc, hd]

noncomputable def truncQuot (n : ℕ) : PowerSeries k →+* RingCat.of ((k[X] /
  ↵ Ideal.span {⟨X ^ n : k[X]⟩})) where
  toFun := fun x => PowerSeries.trunc n x
  map_zero' := by simp
  map_one' := by
    match n with
    | 0 => rw [show X^0 = 1 by simp, Ideal.span_singleton_one]
      simp [Ideal.quotient.zero_eq_one_iff]
    | n + 1 => simp
  map_add' := by simp
  map_mul' := fun x y => by
    rw [← PowerSeries.trunc_trunc_mul_trunc, ← coe_mul, ← (Ideal.quotient.mk
      ↳ _).map_mul, quotCommTrunc k _]

noncomputable def cone_F : Cone (F k) :=
{
  pt := RingCat.of (PowerSeries k)
  π := {
    app := fun ⟨n⟩ => RingCat.ofHom (truncQuot k n)
    naturality := by
      rintro ⟨n⟩ ⟨m⟩ <<<(l : m ≤ n)>>>
      ext (x : PowerSeries k)
      simp [F, truncQuot, ← PowerSeries.trunc_trunc_of_le x l,
        ↳ quotCommTrunc k]
  }
}

theorem power_series_islimit : Nonempty (IsLimit (cone_F k)) := by
  sorry

end CAT_statement_S_0067

```

**Problem 68** (Limit - High). — *Theorem: There is a category  $\mathcal{C}$  such that there exists two regular epimorphisms  $f : c \rightarrow d$  and  $g : c' \rightarrow d'$  in which the product of  $f$  and  $g$  is not regularly epic.*

```

import Mathlib
open CategoryTheory Limits

universe u

theorem regular_epimorphism_not_product_regular_epimorphism : ∃ (C : Type
  ↵ (u+1)) (inst : Category C) (c d c' d' : C) (f : c → d) (g : c' → d')
  ↵ (inst1 : RegularEpi f) (inst2 : RegularEpi g) (hasProd1 :
  ↵ HasBinaryProduct c c') (hasProd2 : HasBinaryProduct d d'), IsEmpty
  ↵ (RegularEpi (prod.map f g)) := by
  sorry

```

**Problem 69** (Limit - High). — *Theorem: An abelian group is torsion free if and only if it is a directed colimit in  $\mathcal{A}b$  of free abelian groups.*

```

import Mathlib
open CategoryTheory Limits

theorem torsionFree_iff_isFilteredColimit_free
  (A : ModuleCat ℤ) :
  NoZeroSMulDivisors ℤ A ↔
  ∃ (J : Type) (_ : SmallCategory J) (_ : IsFiltered J)
    (F : J ⇒ ModuleCat ℤ),
    (∀ j : J, Module.Free ℤ (F.obj j)) ∧
    Nonempty (A ≅ colimit F) := by
  sorry

```

**Problem 70** (Limit - High). — *Definition: A concrete category  $(\mathcal{C}, U)$  over  $\mathcal{B}$  is said to have (small) concrete limits if  $\mathcal{C}$  has all small limits and  $U$  preserves them.*

*Theorem: Let  $(\mathcal{C}, U)$  have small concrete limits. Then  $U$  reflects small limits if and only if  $U$  reflects isomorphisms.*

```

import Mathlib
open CategoryTheory Limits

variable {C : Type u} [Category.{v} C]
variable {D : Type u'} [Category.{v'} D]
variable (U : C ⇒ D)

theorem reflects_limits_iff_reflects_isomorphisms_preserves_limits
  [HasLimitsOfSize.{v, v} C]
  [PreservesLimitsOfSize.{v, v} U]
  [CategoryTheory.Functor.Faithful U]
  :
  ReflectsLimitsOfSize.{v, v} U ↔ U.ReflectsIsomorphisms := by
  sorry

```

**Problem 71** (Limit - High). — *Definition: A functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  is said to **lift limits** if for every diagram  $D : \mathcal{S} \rightarrow \mathcal{C}$  and every limit  $L$  of  $F \circ D$ , there exists a limit  $L' \in \mathcal{D}$  such that  $F(L') \cong L$ .*

*Theorem: A functor that lifts equalizers is faithful if and only if it reflects epimorphisms.*

```
import Mathlib

open CategoryTheory Limits

namespace CAT_statement_S_0071

universe uC vC uD vD w w'

variable {C : Type uC} [Category.{vC} C]
variable {D : Type uD} [Category.{vD} D]
variable (F : C → D)
variable {J : Type w} [Category.{w'} J]

class LiftsLimit (K : J ≈ C) (F : C ≈ D) : Prop where
  lifts {c : Cone (K ≫ F)} (hc : IsLimit c) :
    ∃ c' : Cone K, Nonempty (IsLimit c') ∧ Nonempty (F.mapCone c' ≅ c)

class LiftsLimitsOfShape (J : Type w) [Category.{w'} J] (F : C ≈ D) : Prop
  ↵ where
  liftsLimit : ∀ {K : J ≈ C}, LiftsLimit K F := by infer_instance

theorem functor_faithful_iff_reflectsEpimorphisms_of_liftsEqualizers
  [LiftsLimitsOfShape Limits.WalkingParallelPair F] :
  F.Faithful ↔ F.ReflectsEpimorphisms := by
  sorry

end CAT_statement_S_0071
```

**Problem 72** (Limit - High). — *Theorem: A full subcategory of  $\mathcal{T}\text{op}^{CH}$  is reflective in  $\mathcal{T}\text{op}^{CH}$  if and only if it is cocomplete.*

```
import Mathlib

open CategoryTheory Limits Topology

universe u

variable {D : Type (u+1)} [Category.{u} D]
variable (i : D ≈ Comphaus.{u})
variable [CategoryTheory.Functor.Full i] [CategoryTheory.Functor.Faithful i]

theorem
  ↵ reflective_iff_cocomplete_and_contains_nonempty_of_full_subcategory_Comphaus
  ↵ :
  Nonempty (CategoryTheory.Reflective i) ↔
  (Nonempty (HasColimits D) ∧ ∃ X : D, Nonempty (i.obj X)) := by
  sorry
```

**Problem 73** (Limit - High). — *Definition: A functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  is said to **lift limits** if for every diagram  $D : \mathcal{S} \rightarrow \mathcal{C}$  and every limit  $L$  of  $F \circ D$ , there exists a limit  $L' \in \mathcal{D}$  such that  $F(L') \cong L$ .*

*Theorem: The forgetful functor  $U : \mathcal{T}\text{op} \rightarrow \mathcal{S}\text{et}$  lifts limits, but not reflects limits.*

```

import Mathlib
open CategoryTheory Limits

namespace CAT_statement_S_0073

universe w' w₂' w w₂ v₁ v₂ v₃ u₁ u₂ u₃

variable {C : Type u₁} [Category.{v₁} C]
variable {D : Type u₂} [Category.{v₂} D]
variable {J : Type w} [Category.{w'} J] {K : J ⇒ C}

class LiftsLimit (K : J ⇒ C) (F : C ⇒ D) : Prop where
  lifts {c : Cone (K ≫ F)} (hc : IsLimit c) :
    ∃ c' : Cone K, Nonempty (IsLimit c') ∧ Nonempty (F.mapCone c' ≈ c)

class LiftsLimitsOfShape (J : Type w) [Category.{w'} J] (F : C ⇒ D) : Prop
  ↵ where
  liftsLimit : ∀ {K : J ⇒ C}, LiftsLimit K F := by infer_instance

@[nolint checkUnivs, pp_with_univ]
class LiftsLimitsOfSize (F : C ⇒ D) : Prop where
  liftsLimitsOfSize : ∀ {J : Type w} [Category.{w'} J], LiftsLimitsOfShape J
  ↵ F := by
  infer_instance

abbrev LiftsLimits (F : C ⇒ D) :=
  LiftsLimitsOfSize.{v₂, v₂} F

theorem TopCat_forget_lifts_and_not_reflects_limits :
  LiftsLimits (forget TopCat) ∧ IsEmpty (ReflectsLimits (forget TopCat)) :=
  ↵ by
  sorry

end CAT_statement_S_0073

```

**Problem 74** (Cocompletion - Medium). — *Theorem: Let  $\mathcal{C}$  be a small category. A category  $\mathcal{L}$  containing  $\mathcal{C}$  as a full subcategory is an pro-completion of  $\mathcal{C}$  if and only if the following conditions hold:*

- (1)  $\mathcal{L}$  has cofiltered colimits,
- (2) every object of  $\mathcal{L}$  is the colimit of a cofiltered diagram in  $\mathcal{C}$ , and
- (3) every object of  $\mathcal{C}$  is finitely copresentable in  $\mathcal{L}$ .

Reference: Corollary A.5. JIÍ ADÁMEK, LIANG-TING CHEN, STEFAN MILIUS and HENNINGURBAT, Reiterman's Theorem on Finite Algebras for a Monad, <https://arxiv.org/pdf/2101.00942>

```

import Mathlib
open CategoryTheory Limits

universe u v w u₁ v₁

namespace CAT_statement_S_0074

noncomputable section

```

```

abbrev Pro (C : Type u) [SmallCategory C] := (CategoryTheory.Ind (Cop))op

abbrev proYoneda (C : Type u) [SmallCategory C] : C  $\Rightarrow$  Pro C := 
CategoryTheory.opOp C » (CategoryTheory.Ind.yoneda (C := Cop)).op

def HasCofilteredColimits (L : Type u1) [Category.{v1} L] : Prop := 
 $\forall$  (J : Type w) [SmallCategory J] [IsCofiltered J], HasColimitsOfShape J L

def IsFinitelyCopresentable {L : Type u1} [Category.{v1} L] (X : L) : Prop := 
CategoryTheory.IsFinitelyPresentable.{w} (C := Lop) (Opposite.op X)

def IsCofilteredColimitOf
{C : Type u} [SmallCategory C] {L : Type u1} [Category.{v1} L]
( $\iota$  : C  $\Rightarrow$  L) (X : L) : Prop :=
 $\exists$  (J : Type w) (hJ : SmallCategory J) (hC : IsCofiltered J), by
  let _ := hJ
  let _ := hC
  exact  $\exists$  (F : J  $\Rightarrow$  C) (t : Cocone (F »  $\iota$ )),
  Nonempty (IsColimit t)  $\wedge$  Nonempty (t.pt  $\cong$  X)

def IsProCompletion
{C : Type u} [SmallCategory C] {L : Type u1} [Category.{v1} L]
( $\iota$  : C  $\Rightarrow$  L) : Prop :=
 $\exists$  (e : L  $\cong$  Pro C), Nonempty ( $\iota$  » e.functor  $\cong$  proYoneda C)

def ProCompletionConditions
{C : Type u} [SmallCategory C] {L : Type u1} [Category.{v1} L]
( $\iota$  : C  $\Rightarrow$  L) : Prop :=
HasCofilteredColimits.{w} L  $\wedge$ 
( $\forall$  X : L, IsCofilteredColimitOf.{u, w}  $\iota$  X)  $\wedge$ 
( $\forall$  c : C, IsFinitelyCopresentable.{w} ( $\iota$ .obj c))

theorem isProCompletion_iff_intrinsic_conditions
{C : Type u} [SmallCategory C] {L : Type u1} [Category.{v1} L]
( $\iota$  : C  $\Rightarrow$  L) [CategoryTheory.Functor.Full  $\iota$ ]
 $\hookrightarrow$  [CategoryTheory.Functor.Faithful  $\iota$ ] :
IsProCompletion ( $\iota$  :=  $\iota$ )  $\leftrightarrow$  ProCompletionConditions ( $\iota$  :=  $\iota$ ) := by
sorry

end

end CAT_statement_S_0074

```

**Problem 75** (Cocompletion - High). — *Definition: A category is called sifted if the category of cocones over any finite discrete family of objects in it is connected.*

*Notation:*  $\text{Rec}(\mathcal{C}) :=$  free cocompletion of  $\mathcal{C}$  under reflexive coequalizers.

*Theorem:* For a sifted category with pullbacks  $\mathcal{C}$ ,  $\text{Rec}(\mathcal{C})$  is filtered.

Reference: Proposition 3.2, Chen Ruiyuan 2021, On sifted colimits in the presence of pullbacks, arXiv:2109.12708

```

import Mathlib

namespace CAT_statement_S_0075

open CategoryTheory Limits

universe u v

namespace CategoryTheory.Limits
open Limits Functor
variable {C : Type u} [Category.{v} C]

variable (C) in

abbrev Psh (C : Type u) [Category.{v} C] : Type (max u (v + 1)) :=
Cᵒᵖ → Type v

inductive RecObjectPresentation : Psh C → Type (max u (v + 1))
| ofYoneda (X : C) :
  RecObjectPresentation ((yoneda : C ≈ Psh C).obj X)
| iso {A B : Psh C} (P : RecObjectPresentation A) (i : A ≃ B) :
  RecObjectPresentation B
| reflexiveCoeq {A B : Psh C}
  (PA : RecObjectPresentation A) (PB : RecObjectPresentation B)
  (f g : A → B) [IsReflexivePair f g] [HasCoequalizer f g] :
  RecObjectPresentation (coequalizer f g)

structure IsRecObject (A : Psh C) : Prop where
  mk' :: nonempty_presentation : Nonempty (RecObjectPresentation A)

theorem IsRecObject.mk (A : Psh C) (P : RecObjectPresentation A) :
  IsRecObject A :=
⟨⟨P⟩⟩

theorem isRecObject_yoneda (X : C) :
  IsRecObject (C := C) ((yoneda : C ≈ Psh C).obj X) :=
⟨⟨RecObjectPresentation.ofYoneda (C := C) X⟩⟩

theorem isRecObject_coequalizer
  {A B : Psh C} (hA : IsRecObject (C := C) A) (hB : IsRecObject (C := C) B)
  (f g : A → B) [IsReflexivePair f g] :
  IsRecObject (C := C) (coequalizer f g) := by
  classical
  rcases hA.nonempty_presentation with ⟨PA⟩
  rcases hB.nonempty_presentation with ⟨PB⟩
  letI : HasCoequalizer f g := by infer_instance
  exact ⟨⟨RecObjectPresentation.reflexiveCoeq (C := C) PA PB f g⟩⟩

end CategoryTheory.Limits

```

```

namespace CategoryTheory

open Limits

variable {C : Type u} [Category.{v} C]

variable (C) [LocallySmall C] in

def Rec : Type (max u (v + 1)) :=
  ShrinkHoms (ObjectProperty.FullSubcategory (IsRecObject (C := C)))

noncomputable instance : Category.{max u v} (Rec C) :=
  inferInstanceAs <| Category.{max u v}
    (ShrinkHoms (ObjectProperty.FullSubcategory (IsRecObject (C := C)))))

noncomputable def Rec.equivalence :
  Rec C ≈ ObjectProperty.FullSubcategory (IsRecObject (C := C)) :=
  (ShrinkHoms.equivalence _).symm

theorem sifted_with_pullbacks_Rec_is_filtered {C : Type u} [Category.{v} C]
  [IsSifted C] [HasPullbacks C] :
  IsFiltered (Rec C) := by
  sorry

end CategoryTheory

end CAT_statement_S_0075

```

**Problem 76** (Cocompletion - High). — *Theorem: Let  $\mathcal{S}et^{fin}$  be the category of finite sets and functions. Its pro-completion is the category*

$$\text{Pro}(\mathcal{S}et^{fin}) = \mathcal{S}tone$$

*of Stone spaces, i.e. compact topological spaces in which distinct elements can be separated by clopen subsets. Morphisms are the continuous functions.*

*Reference: JIÍ ADÁMEK, LIANG-TING CHEN, STEFAN MILIUS and HENNINGURBAT, Reiterman's Theorem on Finite Algebras for a Monad, <https://arxiv.org/pdf/2101.00942>*

```

import Mathlib

open CategoryTheory

universe v u

abbrev Pro (C : Type u) [Category.{v, u} C] : Type (max u (v + 1)) := (Ind
  ↳ (Cop))op

theorem pro_fintypecat_equiv_profinite : Nonempty ((Pro (FintypeCat)) ≈
  ↳ Profinite) := sorry

```

**Problem 77** (Cocompletion - High). — *Notation:*

$\text{Sind}(\mathcal{C}) :=$  free cocompletion of  $\mathcal{C}$  under small sifted colimits;  
 $\text{Ind}(\mathcal{C}) :=$  free cocompletion of  $\mathcal{C}$  under small filtered colimits;  
 $\text{Rec}(\mathcal{C}) :=$  free cocompletion of  $\mathcal{C}$  under reflexive coequalizers.

Theorem: Let  $\mathcal{C}$  be a category with pullbacks. Then  $\text{Sind}(\mathcal{C}) = \text{Ind}(\text{Rec}(\mathcal{C}))$

Reference: Theorem 5.1, Chen Ruiyuan 2021, On sifted colimits in the presence of pullbacks, arXiv:2109.12708

```
import Mathlib

namespace CAT_statement_S_0077

open CategoryTheory Limits

universe u v

namespace CategoryTheory.Limits

open Limits Functor
variable {C : Type u} [Category.{v} C]

abbrev Psh (C : Type u) [Category.{v} C] : Type (max u (v + 1)) :=
  Cᵒᵖ → Type v

inductive RecObjectPresentation : Psh C → Type (max u (v + 1))
| ofYoneda (X : C) :
  RecObjectPresentation ((yoneda : C → Psh C).obj X)
| iso {A B : Psh C} (P : RecObjectPresentation A) (i : A ≈ B) :
  RecObjectPresentation B
| reflexiveCoeq {A B : Psh C}
  (PA : RecObjectPresentation A) (PB : RecObjectPresentation B)
  (f g : A → B) [IsReflexivePair f g] [HasCoequalizer f g] :
  RecObjectPresentation (coequalizer f g)

structure IsRecObject (A : Psh C) : Prop where
  mk' :: nonempty_presentation : Nonempty (RecObjectPresentation A)

theorem IsRecObject.mk (A : Psh C) (P : RecObjectPresentation A) :
  IsRecObject A :=
  ⟨⟨P⟩⟩

theorem isRecObject_yoneda (X : C) :
  IsRecObject (C := C) ((yoneda : C → Psh C).obj X) :=
  ⟨⟨RecObjectPresentation.ofYoneda (C := C) X⟩⟩

theorem isRecObject_coequalizer
  {A B : Psh C} (hA : IsRecObject (C := C) A) (hB : IsRecObject (C := C) B)
  (f g : A → B) [IsReflexivePair f g] :
  IsRecObject (C := C) (coequalizer f g) := by
  classical
  rcases hA.nonempty_presentation with ⟨PA⟩
  rcases hB.nonempty_presentation with ⟨PB⟩
  letI : HasCoequalizer f g := by infer_instance
  exact ⟨⟨RecObjectPresentation.reflexiveCoeq (C := C) PA PB f g⟩⟩

end CategoryTheory.Limits
```

```

namespace CategoryTheory

open Limits

variable {C : Type u} [Category.{v} C]

variable (C) in

def Rec : Type (max u (v + 1)) :=
ShrinkHoms (ObjectProperty.FullSubcategory (IsRecObject (C := C)))

noncomputable instance : Category.{max u v} (Rec C) :=
inferInstanceAs <| Category.{max u v}
(ShrinkHoms (ObjectProperty.FullSubcategory (IsRecObject (C := C)))))

noncomputable def Rec.equivalence :
Rec C ≈ ObjectProperty.FullSubcategory (IsRecObject (C := C)) :=
(ShrinkHoms.equivalence _).symm

end CategoryTheory

namespace CategoryTheory.Limits
open Limits Functor
variable {C : Type u} [Category.{v} C]

structure SindObjectPresentation (A : Cop → Type v) where
I : Type v
[ $\mathcal{I}$  : SmallCategory I]
[hI : IsSifted I]

F : I → C
ι : F ≫ yoneda → (Functor.const I).obj A
isColimit : IsColimit (Cocone.mk A ι)

namespace SindObjectPresentation

@[simp]
def yoneda (X : C) : SindObjectPresentation (yoneda.obj X) where
I := Discrete PUnit.{v + 1}
F := Functor.fromPUnit X
ι := { app := fun _ => 1 _ }
isColimit :=
{ desc := fun s => s.ι.app ⟨PUnit.unit⟩
uniq := fun _ _ h => h ⟨PUnit.unit⟩ }

end SindObjectPresentation

```

```

structure IsSindObject (A : Cop → Type v) : Prop where
  mk' :: nonempty_presentation : Nonempty (SindObjectPresentation A)

theorem IsSindObject.mk {A : Cop → Type v} (P : SindObjectPresentation A) :
  ↵ IsSindObject A := ⟨⟨P⟩⟩

theorem isSindObject_yoneda (X : C) : IsSindObject (yoneda.obj X) :=
  .mk <| SindObjectPresentation.yoneda X

end CategoryTheory.Limits

namespace CategoryTheory

open Limits

variable {C : Type u} [Category.{v} C]
variable (C) in

def Sind : Type (max u (v + 1)) :=
  ShrinkHoms (ObjectProperty.FullSubcategory (IsSindObject (C := C)))

noncomputable instance : Category.{max u v} (Sind C) :=
  inferInstanceAs <| Category.{max u v}
    (ShrinkHoms (ObjectProperty.FullSubcategory (IsSindObject (C := C)))))

variable (C) in

noncomputable def Sind.equivalence :
  Sind C ≃ ObjectProperty.FullSubcategory (IsSindObject (C := C)) :=
  (ShrinkHoms.equivalence _).symm

end CategoryTheory

open CategoryTheory

theorem SindC_is_Ind_of_RecC {C : Type u} [SmallCategory C] :
  Nonempty (Sind C ≃ Ind (Rec C)) := by
  sorry

end CAT_statement_S_0077

```

**Problem 78** (Cocompletion - High). — *Def:* For  $F : \mathcal{C} \rightarrow \mathcal{D}$ , we define the induced cocontinuous functor  $\text{Lan}_{F^{\text{op}}} : \mathcal{Psh}(\mathcal{C}) \rightarrow \mathcal{Psh}(\mathcal{D})$ , by  $\phi \mapsto \phi * yF$ , where  $\phi * yF$  is the  $\phi$ -weighted colimit of the diagram  $yF$  and  $y$  is the Yoneda embedding.

*Notation:*  $\text{Sind}(\mathcal{C}) :=$  free cocompletion of  $\mathcal{C}$  under small sifted colimits;

*Theorem:* For any full and faithful  $I : \mathcal{C} \rightarrow \mathcal{D}$  between small categories,  $\phi \in [\mathcal{C}^{\text{op}}, \mathcal{S}\text{et}]$  is in  $\text{Sind}(\mathcal{C})$  iff  $\text{Lan}_{I^{\text{op}}} \text{ is in } \text{Sind}(\mathcal{D})$ .

*Reference:* Lemma 6.2, Chen Ruiyuan 2021, On sifted colimits in the presence of pullbacks, arXiv:2109.12708

```

import Mathlib

namespace CAT_statement_S_0078

open CategoryTheory Limits Functor

universe u v

namespace CategoryTheory

namespace Limits

variable {C : Type u} [Category.{v} C]

structure SindObjectPresentation (A : Cop → Type v) where
  I : Type v
  [I : SmallCategory I]
  [hI : IsSifted I]
  F : I → C
  i : F ≫ yoneda → (Functor.const I).obj A
  isColimit : IsColimit (Cocone.mk A i)

structure IsSindObject (A : Cop → Type v) : Prop where
  mk' :: nonempty_presentation : Nonempty (SindObjectPresentation A)

theorem IsSindObject.mk {A : Cop → Type v} (P : SindObjectPresentation A) :
  IsSindObject A :=
  ⟨⟨P⟩⟩

end Limits

namespace Functor

def weightedColimitFunctor {J : Type v} [SmallCategory J] {E : Type u}
  [Category.{v} E]
  (W : Jop → Type v) (G : J → E) : E → Type v where
    obj X := W ≫ G.op ≫ (yoneda.obj X)
    map f h := h ≫ (NatTrans.id G.op □ yoneda.map f)

abbrev WeightedColimitData {J : Type v} [SmallCategory J] {E : Type u}
  [Category.{v} E]
  (W : Jop → Type v) (G : J → E) (colim : E) :=
  (weightedColimitFunctor W G).CorepresentableBy colim

abbrev HasWeightedColimit {J : Type v} [SmallCategory J] {E : Type u}
  [Category.{v} E]
  (W : Jop → Type v) (G : J → E) :=
  (weightedColimitFunctor W G).IsCorepresentable

```

```

noncomputable def weightedColimit {J : Type v} [SmallCategory J] {E : Type u}
  ↵ [Category.{v} E]
    (W : Jop → Type v) (G : J ⇒ E) [h : HasWeightedColimit W G] : E :=
    h.has_corepresentation.choose

noncomputable def weightedColimitData {J : Type v} [SmallCategory J] {E :
  Type u} [Category.{v} E]
  ↵ (W : Jop → Type v) (G : J ⇒ E) [h : HasWeightedColimit W G] :
    WeightedColimitData W G (weightedColimit W G) :=
    h.has_corepresentation.choose_spec.some

end Functor

end CategoryTheory

open CategoryTheory Limits Functor

variable {C D : Type u} [SmallCategory C] [SmallCategory D]

def lanDiagram (F : C ⇒ D) : C ⇒ (Dop → Type u) := F ≫ yoneda

noncomputable def lanPresheaf (F : C ⇒ D) (φ : Cop → Type u)
  [HasWeightedColimit φ (lanDiagram F)] : Dop → Type u :=
  weightedColimit φ (lanDiagram F)

theorem isSindObject_iff_isSindObject_lanPresheaf
  (I : C ⇒ D) [Full I] [Faithful I] (φ : Cop → Type u)
  [HasWeightedColimit φ (lanDiagram I)] :
  IsSindObject φ ↔ IsSindObject (lanPresheaf I φ) := by
  sorry

end CAT_statement_S_0078

```

**Problem 79** (Abelian - Easy). — *Theorem: Let  $\mathcal{A}$  be an additive category. Let  $x, y, z$  be objects in  $\mathcal{A}$ . Then the composition  $\text{hom}_{\mathcal{A}}(y, z) \times \text{hom}_{\mathcal{A}}(x, y) \rightarrow \text{hom}_{\mathcal{A}}(x, z)$  is bilinear map.*

```

import Mathlib

open CategoryTheory

variable {C : Type u} [Category.{v} C] [Preadditive C]

structure IsBilinear {X Y Z : C} (f : (Y → Z) → ((X → Y) → (X → Z))) : Prop
  ↵ where
  map_add_left : ∀ (a b : Y → Z) (g : X → Y),
    f (a + b) g = f a g + f b g

  map_add_right : ∀ (a : Y → Z) (g h : X → Y), f a (g + h) = f a g + f a h

  map_zero_left : ∀ (g : X → Y), f 0 g = 0

  map_zero_right : ∀ (a : Y → Z), f a 0 = 0

```

```
theorem compIsBilinear {X Y Z : C} :
  IsBilinear (fun (g : Y → Z) => (fun (f : (X → Y)) => f ≫ g)) := sorry
```

**Problem 80** (Abelian - Easy). — *Theorem: Let  $\mathcal{A}$  be an abelian category and let  $f$  be a morphism in  $\mathcal{A}$ . Then  $f$  is an isomorphism if and only if  $f$  is monic and epic.*

```
import Mathlib

open CategoryTheory

variable {C : Type*} [Category C] [Abelian C]

theorem isIso_iff_mono_and_epi {X Y : C} (f : X → Y) :
  IsIso f ↔ (Mono f ∧ Epi f) := by
  sorry
```

**Problem 81** (Abelian - Easy). — *Theorem: Let  $\mathcal{A}$  be an abelian category and let  $f$  be a morphism in  $\mathcal{A}$ . Then  $f$  is monic if and only if  $\ker(f) = 0$ .*

```
import Mathlib

open CategoryTheory Limits Category

variable {C : Type*} [Category C] [Abelian C]

theorem mono_iff_isZero_kernel {X Y : C} (f : X → Y) :
  Mono f ↔ IsZero (kernel f) := by
  sorry
```

**Problem 82** (Abelian - Easy). — *Theorem:  $\mathbb{k}$  is the unique (up to isomorphism) simple object in  $\text{Vect}_{\mathbb{k}}$ .*

```
import Mathlib

open Module

variable (k : Type u) [Field k]

instance isSimpleModule_self : IsSimpleModule k k := by
  constructor
  intro N
  have : IsSimpleOrder (Submodule k k) := by infer_instance
  exact eq_bot_or_eq_top N

theorem unique_simple_object (M : Type v) [AddCommGroup M] [Module k M]
  ← [IsSimpleModule k M] :
  Nonempty (M ≃[k] k) := by
  sorry
```

**Problem 83** (Abelian - Easy). — *Theorem:  $\mathbb{Z}_p$  is simple object in  $\mathcal{A}\mathbf{b}$  when  $p$  is prime number.*

```

import Mathlib
open CategoryTheory
variable (p : ℕ) [Fact p.Prime]
theorem ZMod_simple : CategoryTheory.Simple (ModuleCat.of ℤ (ZMod p)) := by
  sorry

```

**Problem 84** (Abelian - Easy). — *Theorem:  $\mathcal{G}rp$  is not an additive category.*

```

import Mathlib
open CategoryTheory Limits

def IsAdditiveCategory (C : Type u) [Category.{v} C] : Prop :=
  ∃ (_ : Preadditive C), HasZeroObject C ∧ HasFiniteBiproducts C

theorem Grp_not_is_additive : IsEmpty (IsAdditiveCategory Grp.{u}) := by
  sorry

```

**Problem 85** (Abelian - Medium). — *Definition: A functor is called **left exact** if it preserves all finite limits.*

*Theorem: Let  $\mathcal{A}$  and  $\mathcal{B}$  be abelian categories and let  $F : \mathcal{A} \rightarrow \mathcal{B}$  be a functor. Then  $F$  is left exact if and only if  $F$  is additive and  $F$  maps exact sequence  $0 \rightarrow x \rightarrow y \rightarrow z$  to  $0 \rightarrow F(x) \rightarrow F(y) \rightarrow F(z)$ .*

```

import Mathlib
open CategoryTheory Functor Limits ShortComplex

variable {C D : Type*} [Category C] [Category D]
variable [Abelian C] [Abelian D]

theorem preservesFiniteLimits_tfae
  (F : C ≈ D) [F.Additive] : List.TFAE
  [
    ∀ (S : ShortComplex C), S.ShortExact → (S.map F).Exact ∧ Mono (F.map
      ← S.f),
    ∀ (S : ShortComplex C), S.Exact ∧ Mono S.f → (S.map F).Exact ∧ Mono
      ← (F.map S.f),
    ∀ {X Y : C} (f : X → Y), PreservesLimit (parallelPair f 0) F,
    PreservesFiniteLimits F
  ] := by
  sorry

```

**Problem 86** (Abelian - Medium). — *Theorem: Let  $\mathcal{A}$  be an abelian category and let  $P \in \mathcal{A}$ . Then  $\hom_{\mathcal{A}}(P, -) : \mathcal{A} \rightarrow \mathcal{A}\text{-}\mathbf{op}$  is right exact if and only if  $\hom_{\mathcal{A}}(P, -) : \mathcal{A} \rightarrow \mathcal{A}\text{-}\mathbf{op}$  preserves epimorphism.*

```

import Mathlib
open CategoryTheory Limits Opposite

```

```

variable {A : Type u} [Category.{v} A] [Abelian A]

theorem hom_rightExact_iff_preserves_epi (P : A) :
  PreservesFiniteColimits (preadditiveCoyoneda.obj (op P)) ↔
  Functor.PreservesEpimorphisms (preadditiveCoyoneda.obj (op P)) := by
  sorry

```

**Problem 87** (Abelian - Medium). — *Definition: An Abelian category  $\mathcal{A}$  is called **semisimple** if any short exact sequence in  $\mathcal{A}$  is splittable.*

*Theorem: Let  $\mathcal{A}$  be an abelian category. Then the followings are equivalent:*

1.  $\mathcal{A}$  is semisimple;
2. any object in  $\mathcal{A}$  is injective;
3. any object in  $\mathcal{A}$  is projective.

```

import Mathlib

open CategoryTheory.Limits

variable {A : Type u} [Category.{v} A] [Abelian A]

def IsSemisimple (A : Type u) [Category.{v} A] [Abelian A] : Prop :=
  ∀ (S : ShortComplex A), S.ShortExact → Nonempty S.Splitting

theorem isSemisimple_iff_injective_iff_projective :
  (IsSemisimple A ↔ ∀ (X : A), Injective X) ∧
  (IsSemisimple A ↔ ∀ (X : A), Projective X) := by
  sorry

```

**Problem 88** (Abelian - Medium). — *Theorem: Let  $\mathcal{A}$  be an abelian category. If  $x, y$  are simple objects in  $\mathcal{A}$ . Then each non-zero  $f : x \rightarrow y$  are isomorphism. In particular, if  $x$  is simple, then  $\text{hom}_{\mathcal{A}}(x, x)$  is a division ring; if  $x \neq y$ , then  $\text{hom}_{\mathcal{A}}(x, y) = 0$ .*

```

import Mathlib

open CategoryTheory

variable {A : Type*} [Category A] [Abelian A]

theorem simple_objects_nonzero_morphisms_iso
  {x y : A} [Simple x] [Simple y] (f : x → y) (h : f ≠ 0) :
  IsIso f := by
  sorry

theorem simple_object_end_is_division_ring
  (x : A) [Simple x] :
  Nonempty (DivisionRing (CategoryTheory.End x)) := by
  sorry

```

```

theorem simple_objects_hom_zero_of_ne
  {x y : A} [Simple x] [Simple y] (hxy : x ≠ y) :
  ∀ f : x → y, f = 0 := by
  sorry

```

**Problem 89** (Abelian - Medium). — *Theorem: Let  $\mathcal{A}$  be an additive category. Let  $x$  be a Schurian simple object, then it is both monosimple and episimple.*

```

import Mathlib

open CategoryTheory

class IsSplitMonoCategory (A : Type*) [Category A] where
  splitMonoOfMono {X Y : A} (f : X → Y) [Mono f] : Nonempty (SplitMono f)

class IsSplitEpiCategory (A : Type*) [Category A] where
  splitEpiOfEpi {X Y : A} (f : X → Y) [Epi f] : Nonempty (SplitEpi f)

variable {A : Type*} [Category A] [Preadditive A] [IsSplitMonoCategory A]
  ← [IsSplitEpiCategory A]

theorem schur_simple_monosimple_and_episimple
  (x : A) [NoZeroDivisors (End x)] :
  (∀ (y : A) (f : y → x) [Mono f], f = 0 ∨ IsIso f) ∧
  (∀ (y : A) (g : x → y) [Epi g], g = 0 ∨ IsIso g) := by
  sorry

```

**Problem 90** (Abelian - High). — *Definition: A category is called **normal** if each monomorphism is a kernel.*

*Definition: A category is called **conormal** if each epimorphism is a cokernel.*

*Definition: A category is called **binormal** if it is both normal and conormal.*

*Definition: Let  $\mathcal{C}$  be a category. An object  $c \in \mathcal{C}$  is called **mono-simple** if it has no proper subobjects. An object  $c \in \mathcal{C}$  is called **epi-simple** if it has no proper quotient objects.*

*Theorem: Let  $\mathcal{A}$  be a binormal category. Then an object is mono-simple if and only if it is epi-simple.*

```

import Mathlib

open CategoryTheory

variable {A : Type*} [Category A] [Limits.HasZeroMorphisms A]
  [IsNormalMonoCategory A]
  [IsNormalEpiCategory A]

  [Limits.HasKernels A]
  [Limits.HasCokernels A]

theorem binormal_mono_simple_iff_epi_simple (x : A) :
  (∀ (y : A) (f : y → x) [Mono f], f = 0 ∨ IsIso f) ↔
  (∀ (y : A) (g : x → y) [Epi g], g = 0 ∨ IsIso g) := by
  sorry

```

**Problem 91** (Monad - Easy). — *Theorem: For any monad  $(T, \mu, \eta)$  on a category  $\mathcal{C}$  and let  $\mathcal{C}^T$  be its Eilenberg-Moore category. Let  $U : \mathcal{C}^T \rightarrow \mathcal{C}$  be the forgetful functor, then it admits a left adjoint.*

```

import Mathlib
open CategoryTheory
variable {C : Type u₁} [Category.{v₁} C]

theorem monad_forget_has_left_adjoint (T : Monad C) :
  T.forget.IsRightAdjoint := by
  sorry

```

**Problem 92** (Monad - Easy). — *Theorem: The forgetful functor  $U : \mathcal{A}\mathbf{b}_R \rightarrow \mathcal{A}\mathbf{b}$  creates all colimits that  $\mathcal{A}\mathbf{b}$  admits.*

```

import Mathlib
open CategoryTheory Limits
variable {R : Type u} [CommRing R]

theorem ModuleCat.forgetReflectsColimits :
  Nonempty (ReflectsColimits (forget₂ (ModuleCat R) AddCommGrp)) := 
  sorry

```

**Problem 93** (Monad - Medium). — *Theorem: Suppose  $\mathcal{C}$  is cocomplete and  $G : \mathcal{D} \rightarrow \mathcal{C}$  is monadic. Then  $\mathcal{D}$  is cocomplete if and only if  $\mathcal{D}$  has coequalizers.*

```

import Mathlib
open CategoryTheory Limits
universe uC uD vC vD w w'
variable {C : Type uC} [Category.{vC} C]
variable {D : Type uD} [Category.{vD} D]
variable (G : D → C)

theorem cocomplete_iff_hasCoequalizers_of_monadic
  [HasColimitsOfSize.{w, w'} C] [MonadicRightAdjoint G] :
  HasColimitsOfSize.{w, w'} D ↔ HasCoequalizers D := by
  sorry

```

**Problem 94** (Monad - Medium). — *Theorem: If  $U : \mathcal{C} \rightarrow \mathcal{B}$  is an isomorphism-closed full reflective embedding, then the associated monad is idempotent.*

```

import Mathlib
open CategoryTheory Functor
namespace CAT_statement_S_0094

```

```

variable {C : Type*} [Category C]
variable {B : Type*} [Category B]

noncomputable def monadOfRightAdjoint (U : Functor C B) [IsRightAdjoint U] :
  ↳ Monad B := 
    (Adjunction.ofIsRightAdjoint U).toMonad

def IsIsoClosed (U : Functor C B) := ∀ (x : C) (y : B) (f : U.obj x → y)
  ↳ [IsIso f], ∃ (z : C), y = U.obj z

variable {U : Functor C B} [Full U] [Faithful U] [IsRightAdjoint U]
  {h_inj : Function.Injective U.obj}
  {h_iso_closed : IsIsoClosed U}

theorem monad_idempotent_of_full_reflective_embedding :
  let T : Monad B := monadOfRightAdjoint U
  IsIso T.μ := by
  sorry

end CAT_statement_S_0094

```

**Problem 95** (Monad - Medium). — *Theorem: If  $\mathcal{D}$  admits coequalizers, a functor  $G : \mathcal{D} \rightarrow \mathcal{C}$  is monadic if  $G$  has a left adjoint, conservative and preserves coequalizers.*

```

import Mathlib

open CategoryTheory Limits

universe u₁ u₂ v₁

variable {C : Type u₁} {D : Type u₂} [Category.{v₁} C] [Category.{v₁} D]
variable {G : D ≈ C} {F : C ≈ D} (adjFG : F ⊢ G)
variable [HasCoequalizers D]
variable [G.ReflectsIsomorphisms]
variable [PreservesColimitsOfShape WalkingParallelPair G]

theorem monadicOfConservativePreservesCoequalizers :
  Nonempty (MonadicRightAdjoint G) := by
  sorry

```

**Problem 96** (Monad - Medium). — *Theorem: Consider the adjunction  $-\otimes_{\mathbb{Z}} R : \mathcal{A}\mathbf{b} \rightarrow \mathcal{A}\mathbf{b}_R$  and  $U : \mathcal{A}\mathbf{b}_R \rightarrow \mathcal{A}\mathbf{b}$ . We obtain a monad  $T$ . The  $T$ -modules are right  $R$ -modules.*

```

import Mathlib

open CategoryTheory

namespace CAT_statement_S_0096

universe u v

```

```

variable (R : Type u) [CommRing R]

abbrev intToR : ℤ →+* R := Int.castRingHom R

noncomputable abbrev U : ModuleCat.{max u v} R ⇒ ModuleCat.{max u v} ℤ := 
ModuleCat.restrictScalars (intToR R)

noncomputable abbrev F : ModuleCat.{max u v} ℤ ⇒ ModuleCat.{max u v} R := 
ModuleCat.extendScalars (intToR R)

noncomputable abbrev adj : F (R := R) ⊢ U (R := R) := 
ModuleCat.extendRestrictScalarsAdj (intToR R)

noncomputable abbrev T : Monad (ModuleCat.{max u v} ℤ) := 
(adj (R := R)).toMonad

theorem t_algebra_equiv_modulecat : 
Nonempty (Monad.Algebra (T (R := R)) ≃ ModuleCat.{max u v} R) := by
sorry

end CAT_statement_S_0096

```

**Problem 97** (Monad - Medium). — *Theorem: Let  $\mathcal{C}, \mathcal{D}$  be categories and  $F : \mathcal{C} \rightarrow \mathcal{D}$  be a left adjoint functor to  $G : \mathcal{D} \rightarrow \mathcal{C}$ . Denote the induced monad of the adjunction  $F \dashv G$  by  $T := GF$ . Let  $K : \mathcal{D} \rightarrow \mathcal{C}^T$  be the comparison functor. If  $\mathcal{D}$  admits coequalizers, then  $K$  has a left adjoint.*

```

import Mathlib

open CategoryTheory Monad

universe u₁ u₂ v₁

variable {C : Type u₁} [Category.{v₁} C] {D : Type u₂} [Category.{v₁} D]
variable (F : C ⇛ D) (G : D ⇛ C) (adj : F ⊢ G)

theorem comparison_adjunction
[∀ (A : adj.toMonad.Algebra), Limits.HasCoequalizer (F.map A.a)
→ (adj.counit.app (F.obj A.A))] :
∃ K : adj.toMonad.Algebra ⇒ D, Nonempty (K ⊢ comparison adj) := by
sorry

```

**Problem 98** (Monad - Medium). — *Definition: For any monad  $T$  on  $\mathcal{C}$ , we define a category  $\text{Adj}_T$  whose objects are adjunctions  $(F : \mathcal{C} \rightarrow \mathcal{D}, G, \eta, \epsilon)$  which induce the same monad  $T$ , and a morphism between  $(F : \mathcal{C} \rightarrow \mathcal{D}, G, \eta, \epsilon)$  and  $(F' : \mathcal{C} \rightarrow \mathcal{D}', G', \eta', \epsilon')$  in  $\text{Adj}_T$  is given by a functor  $K : \mathcal{D} \rightarrow \mathcal{D}'$  such that  $KF = F'$  and  $K'G = G'$ .*

*Theorem: Let  $(T, \mu, \eta)$  be a monad on a category  $\mathcal{C}$ . The Kleisli category  $\mathcal{C}_T$  is initial in  $\text{Adj}_T$  and the Eilenberg-Moore category  $\mathcal{C}^T$  is terminal,*

```
import Mathlib
```

```

open CategoryTheory Monad

namespace CAT_statement_S_0098

variable {C : Type*} [Category C]

structure AdjCat (T : Monad C) where
  D : Type*
  [category : Category D]
  F : Functor C D
  U : Functor D C
  adj : F ≤ U
  monad_eq : T ≈ Adjunction.toMonad adj

namespace AdjCat

variable {T : Monad C}

instance (X : AdjCat T) : Category X.D := X.category

structure Hom (X Y : AdjCat T) where
  K : Functor (X.D) (Y.D)
  comm_left : X.F ≫ K = Y.F
  comm_right : K ≫ Y.U = X.U

instance : Category (AdjCat T) where
  Hom X Y := Hom X Y
  id X :=
    { K := Functor.id X.D
      comm_left := Functor.comp_id X.F
      comm_right := Functor.id_comp X.U }
  comp f g :=
    { K := f.K ≫ g.K
      comm_left := by
        rewrite [←Functor.assoc, f.comm_left]
        exact g.comm_left
      comm_right := by
        rewrite [Functor.assoc, g.comm_right]
        exact f.comm_right }

end AdjCat

variable (T : Monad C)

def kleisli_adj_obj : AdjCat T :=
{ D := Kleisli T
  F := Kleisli.Adjunction.toKleisli T
  U := Kleisli.Adjunction.fromKleisli T
  adj := Kleisli.Adjunction.adj T
  monad_eq :=
    { hom :=
      { app := fun X => 1 (T.obj X)
        app_μ (X : C) := by
          simp
          rewrite [Kleisli.Adjunction.adj] }
    }
}

```

```

        simp
        rewrite [Equiv.refl]
        simp }

inv :=
{ app := fun X => 1 (T.obj X)
app_μ (X : C) := by
simp
rewrite [Kleisli.Adjunction.adj]
simp
rewrite [Equiv.refl]
simp } } }

theorem kleisli_initial : Nonempty (Limits.IsInitial (kleisli_adj_obj T)) :=
by
sorry

def eilenberg_moore_adj_obj : AdjCat T :=
{ D := T.Algebra
F := Monad.free T
U := Monad.forget T
adj := Monad.adj T
monad_eq :=
{ hom := { app := fun X => 1 (T.obj X) }
inv := { app := fun X => 1 (T.obj X) } } }

theorem eilenberg_moore_terminal : Nonempty (Limits.IsTerminal
(eilenberg_moore_adj_obj T)) := by
sorry

end CAT_statement_S_0098

```

**Problem 99** (Monad - Medium). — *Theorem: The monad associated with the forgetful functor  $\mathcal{T}\text{op} \rightarrow \mathcal{S}\text{et}$  is idempotent.*

```

import Mathlib

open CategoryTheory

theorem monad_Top_idempotent : IsIso TopCat.adj₁.toMonad.μ := by
sorry

```

**Problem 100** (Monad - High). — *Theorem: Let  $\mathcal{C}, \mathcal{D}$  be categories and  $F : \mathcal{C} \rightarrow \mathcal{D}$  be a left adjoint functor to  $G : \mathcal{D} \rightarrow \mathcal{C}$ . Denote the induced monad of the adjunction  $F \dashv G$  by  $T$ . The following statements are equivalent:*

1. *The comparison functor  $K : \mathcal{D} \rightarrow \mathcal{C}^T$  is fully faithful.*
2. *For every  $d \in \mathcal{D}$ , the counit  $\epsilon_d : FG(d) \rightarrow d$  is a coequalizer of*

$$FGFG(d) \begin{array}{c} \xrightarrow{\epsilon_{FG(d)}} \\[-1ex] \xrightarrow{FG(\epsilon_d)} \end{array} FG(d)$$

3. *The functor  $G$  reflects split epimorphisms to regular epimorphisms*

```

import Mathlib
open CategoryTheory Limits

universe v u u'

namespace CAT_statement_S0100

variable {C : Type u} [Category.{v} C]
variable {D : Type u'} [Category.{v} D]

variable (F : C ⇒ D) (G : D ⇒ C)
variable (adj : F ⊢ G)

abbrev FG : D ⇒ D := G ≫ F

abbrev K : D ⇒ (adj.toMonad).Algebra :=
  Monad.comparison adj

abbrev epsFG (d : D) :
  (FG (F := F) (G := G)).obj ((FG (F := F) (G := G)).obj d)
  → (FG (F := F) (G := G)).obj d :=
  adj.counit.app ((FG (F := F) (G := G)).obj d)

abbrev FGeps (d : D) :
  (FG (F := F) (G := G)).obj ((FG (F := F) (G := G)).obj d)
  → (FG (F := F) (G := G)).obj d :=
  (FG (F := F) (G := G)).map (adj.counit.app d)

def counitCofork (d : D) :
  Cofork (epsFG (F := F) (G := G) adj d) (FGeps (F := F) (G := G) adj d) :=
  Cofork.ofπ (adj.counit.app d) (by
    simp [epsFG, FGeps]
  )

def cond1 : Prop :=
  (K (F := F) (G := G) adj).Full ∧ (K (F := F) (G := G) adj).Faithful

def cond2 : Prop :=
  ∀ d : D, Nonempty (IsColimit (counitCofork (F := F) (G := G) adj d))

def IsRegularEpi' {X Y : D} (f : X → Y) : Prop :=
  Nonempty (RegularEpi f)

def ReflectsSplitEpiToRegularEpi' (G : D ⇒ C) : Prop :=
  ∀ {X Y : D} (f : X → Y), IsSplitEpi (G.map f) → IsRegularEpi' (D := D) f

def cond3 : Prop :=
  ReflectsSplitEpiToRegularEpi' (G := G)

```

```
theorem K_fullyFaithful_tfae :  
  List.TFAE  
  [cond1 (F := F) (G := G) adj,  
   cond2 (F := F) (G := G) adj,  
   cond3 (G := G)] := by  
  sorry  
end CAT_statement_S0100
```