Lecture 19: Equivalence of Categories

Lecturer: Max S. New
Scribe: Daiwen Zhang

March 22, 2023

Contents

0 Preface [00:01:05]-[00:04:23] **1**

1 Isomorphism of Categories [00:04:24]-[00:07:45] **1**

2 Examples: Par and $\mathbf{Set}_*$ [00:07:46]-[00:25:36] **2**

3 Equivalence of Categories [00:25:37]-[00:30:36] **5**

4 Examples: $\mathrm{Un}_1 \simeq \mathcal{C}$ [00:30-37]-[00:36:50] **5**

5 ESO., Full and Faithful [00:36:51]-[00:49:02] **7**

6 Proof of Par $\simeq$ $\mathbf{Set}_*$ [00:49:03]-[00:54:12] **8**

7 $\mathbf{Pred}(X) \simeq \mathbf{Mono}(X)$ [00:54:13]-[01:07:21] **9**

8 $\mathbf{Set}^X \simeq \mathbf{Set}/X$: [01:07:22]-[01:17:05] **11**

0 Preface [00:01:05]-[00:04:23]

[00:01:05]

Do you know where Category Theory was invented? It was right here at the University of Michigan! (by Saunders Mac Lane and Samuel Eilenberg in 1941) [Page 1, Emily Riehl's *Category Theory in Context*]

Instead of talking about adjunction, today we are going to discuss a big and important topic that we have not covered yet: *when is one category $\mathcal{C}$ "the same" as another?*

1 Isomorphism of Categories [00:04:24]-[00:07:45]

[00:04:24]

What we might mean by saying two categories are the same? Literally, that might mean they have the same definition, but we really like something weaker than that.

Maybe I have some category $\mathcal{C}$, and I have another category $\mathcal{D}$ that's very, very similar to it. But maybe I've alerady proved a bunch of things about this other category $\mathcal{D}$ and I would like to use them. Maybe I know that $\mathcal{D}$ has products, or coproducts, and I know it's essentially the same category as another category $\mathcal{C}$ and I want to have that $\mathcal{C}$ has coproducts. So, there already is a notion that we have for when two categories should be the same, and that is isomorphism of categories.

What does that mean? Remember that we have the (large) category **Cat**, whose objects are small categories, and its morphisms are functors. So, for any categories, we have a notion of when two objects of the category are the same, by probing them with other morphism of the category, and that is isomorphism.

In any category, we have the notion of an isomorphism. We might say that between $\mathcal{C}$ and $\mathcal{D}$ I want some functor $F : \mathcal{C} \to \mathcal{D}$ and some functor $F^{-1} : \mathcal{D} \to \mathcal{C}$ going backwards,

$$\mathcal{C} \underset{F^{-1}}{\overset{F}{\rightleftarrows}} \mathcal{D}$$

such that

$$F^{-1}F = \mathrm{id}_{\mathcal{C}},$$
$$FF^{-1} = \mathrm{id}_{\mathcal{D}}.$$

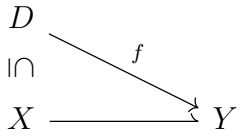
Just like there's no problem with saying $\mathcal{C}$ is the same category as $\mathcal{D}$ when they're exactly the same, we can say $\mathcal{C}$ and $\mathcal{D}$ are the same when there is an isomorphism between them. The problem is this definition is too strict to be useful.

2 Examples: Par and Set$_*$ [00:07:46]-[00:25:36]

[00:07:46] Let's look at an example from the homework and is related to programs that might crash. Let **Par** be the category,

$$\mathbf{Par}_0 := \text{(small) sets},$$
$$\mathbf{Par}(X, Y) := X \rightharpoonup Y,$$

where the objects were small sets and the morphism from X to Y was a partial function from X to Y, with a subset $D \subseteq X$ and a total function $D \to Y$.



We call D the domain of the function with a notion of composition as well. In addition, we say that the function f is undefined on any points outside of the domain.

On the other hand, there is another very natural definition, especially when you're looking at the syntax of programs that might crash, which is that we don't really have any partial functions if we add a crashing term to our language. Instead, we

have a slightly different structure. We have that the crashing program is a legitimate program, but it represents undefinedness. And that is captured by a different category called the category of pointed sets, $\mathbf{Set}_*$, where a pointed set is a pair of: X, a set, and an element $x \in X$.

$$\begin{aligned}(\mathbf{Set}_*)_0 &:= (X \text{ set}) \quad \text{and} \quad x \in X,\\ \mathbf{Set}_*\,(x \in X, y \in Y) &:= f : X \to Y \quad \text{s.t.} \quad fx = y.\end{aligned}$$

The way we're going to think about this x, the distinguished point, is that this point represents "undefined". So in our set of programs, this X would be the closed programs of a given type, and x would be our crash program, our exit.

Then, a morphism between $x \in X$ and $y \in Y$, is going to be a function from X to Y that preserves this undefined point or base point. Notice that there is no way to give the empty set a pointed set structure because it doesn't have any points. Also, this idea is basically the same with the notion of a pointed topological space, where we have a distinguished base point that we need to take into account some calculations. In fact, there is a more general definition, but we will stick to this concrete one.

We can verify that both of **Par** and $\mathbf{Set}_*$ are actually categories. The composition on **Par** was described in the homework, and we can see that the morphisms f of $\mathbf{Set}_*$ preserves the base point property.

The idea is that these two categories are two seemingly equivalent ways to represent the idea of a potentially failing function. In partial functions, the way we capture failure is through undefinedness, where when given an x and it's not in the domain, the computation fails to run or execute; whereas with the pointed set view, we would think of when applying the function, if it results in the base point, then it fails or crashes. In fact, they're really equivalent in some way.

Notice that the one element set, which is terminal in $\mathbf{Set}_*$, is impossible to "succeed" with that only one base point.

00:13:56

These two categories seem kind of similar, but are they actually isomorphic as categories? We will start by defining functors between them.

First, consider the functor $F : \mathbf{Par} \to \mathbf{Set}_*$. Since the objects $X \in \mathbf{Par}$ might be the empty set, the only thing we can generically do is adding a new base point.

$$\begin{aligned}F &: \mathbf{Par} \to \mathbf{Set}_*\\ F(X) &:= X \uplus \{\texttt{err}\}\\ F(D, f) &:= X \uplus \{\texttt{err}\} \to Y \uplus \{\texttt{err}\},\\ F(D, f)(\texttt{err}) &:= \texttt{err},\\ F(D, f)(x \in D) &:= f(x),\\ F(D, f)(x \notin D) &:= \texttt{err}.\end{aligned}$$

The idea is that since all elements of X represents successful computations, we can then add in a new element that represents an error in computation. Then, given a partial function f defined on the domain D, we need to produce a total function that preserves the base point. The idea is that, if there's error, it gets sent to an error

since we have to preserve it; if it's $x' \in X$ and in the domain, then we can send to $f(x')$, if it's not in the domain, we send it to error. It's straightforward to verify that
[00:17:10] this indeed is a functor that preserves composition.

Now let's try to define functor going the other direction, $G : \mathbf{Set}_* \to \mathbf{Par}$. Here we want to take the representation of a partial function as a base point-preserving morphism of pointed sets, to the notion of a partial function in **Par**. In particular, this means that we have to preserve undefinedness.

$$\begin{aligned} G : \mathbf{Set}_* &\to \mathbf{Par} \\ G(X \ni x) &:= X - \{x\} \\ G(f : {}_{x\in}X \to {}_{y\in}Y) &:= (D := \{x' \in X | f(x') \neq y\}, f|_D) \end{aligned}$$

So we will remove the base point which represents an undefined computation and only return totally defined elements. Then, given such a function f, the domain will be all the elements in X that not get mapped to y, and the function is essentially f
[00:20:30] restricted to this domain.

Now let's check do we actually get an isomorphism of categories.

$$\mathrm{Par} \underset{G}{\overset{F}{\rightleftarrows}} \mathrm{Set}_*$$

On the one hand, it's easier to check that $G \circ F$ is the identity as we added an error element and then dropped it. For the other direction, start with some pointed set $x \in X$, we mapped it to an object $X - \{x\}$ of **Par**, and F throws in a new element `err`.

$$x \in X \xrightarrow{\quad G \quad} X - \{x\} \xrightarrow{\quad F \quad} (X - \{x\}) \uplus \{\texttt{err}\} \ni \texttt{err}$$
$$x \in X \xrightarrow{\quad \neq \quad} \mathrm{Id}$$

Then when we get it back, it's not what we started with, so it's not possible for $F \circ G$ to be equal to the identity function, as it's simply not the identity on objects.

However, the codomain set is in a base point-preserving bijection with the set X, so we can define a function mapping x to `err` and vice versa. So even it's not the case that $F \circ G$ is equal to the identity, we do get there is an isomorphism between them, which is almost as good.

$$\begin{aligned} F \circ G &\neq \mathrm{Id}, \\ F \circ G &\cong \mathrm{Id}. \end{aligned}$$

And in fact, this isomorphism is natural in this pointed set X.

Note that this is actually the common case, not the one where the isomorphism actually works. For the essentially same reasons, when we have something like a group structure, a monoid, a topological space, it's very unlikely that you're going to do a

bunch of constructions and get the exact same set back. But, you can reconstruct something that is equivalent to what you started with. So this notion of a "weak" isomorphism, occurs far more often.

It seems hard to prove that there's no isomorphism between categories. But, in this case, consider the initial object in $\mathbf{Set}_*$ which is the one-element set. One of these categories has one initial object and the other one has multiple isomorphic initial objects, so they cannot be isomorphic.

3 Equivalence of Categories [00:25:37]-[00:30:36]

[00:25:37] It's not worth asking whether we have an isomorphism between two categories, as previously mentioned, it's "evil". Because we're trying to demand that two sets end up being exactly the same, i.e., two objects of a category are the same, or, equivalently, we are requiring two functors to be the same. But remember, functors are objects of a category, we don't like to consider functors to be just elements of a set, we like to consider functors to be objects of a functor category. And the right notion of two objects of a category being the same is isomorphism in that category, so the right notion of two functors being equivalent is natural isomorphism in functor category.

We will consider two categories to be *essentially the same* when we have functors going back and forth that are each (naturally) isomorphic to the identity, and this is call an **equivalence of categories**. In a lot of ways it acts like a notion of isomorphism in that there are nice things such as we can compose two equivalences of categories together and the composite functor will be an equivalence of categories. And the identity functor will be an equivalence of categories. So, in a sensible notion, equivalence of categories is an equivalence relation on categories.

If you've done some like Homotopy Theory, one way to think about that is that we have an isomorphism up to homotopy being this natural isomorphism.

So, a great deal of category theory comes from weakening equations like this that we have from set theory to more complex concepts involving natural isomorphism.

Notice that I'm not saying equality is "evil", what I'm saying is equality of functors is "evil". Equality of natural transformations or equality of morphisms in a category is not "evil". Equality of morphisms in a 2-category is "evil" again. In fact, one way to look at this is there's something called 2-categories which abstract over the structure of the category of categories that there's the notion of a two-dimensional morphism between one-dimensional morphisms. So there's a whole area, higher category theory, goes all the way to infinite category theory, that is extremely technical and hard. We're going to do very low dimensional in two or one dimension, and this is genuinely first 2-categorical concept.

4 Examples: $\mathrm{Un}_1 \simeq \mathcal{C}$ [00:30-37]-[00:36:50]

[00:30:37] If we have a Cartesian category $\mathcal{C}$, then we could turn in into a CT structure self$\mathcal{C}$, and from the CT structure, we can take the unary categories of it, Un_Γ for each $\Gamma \in \mathcal{C}$.

Here we have the objects and morphisms from the definition:

$$\forall \Gamma \in \mathcal{C},\ (\mathrm{Un}_\Gamma)_0 := \mathcal{C}_0,$$
$$\mathrm{Un}_\Gamma(A, B) := \Gamma \times A \to B.$$

The idea is the composition of morphisms threads this Γ everywhere.

Now, what happens if we take Γ to be the terminal object $1 \in \mathcal{C}$? We get

$$\mathrm{Un}_1(A, B) := 1 \times A \to B,$$

and from previously homework and the soundness of STT, we have that $1 \times A$ is always naturally isomorphic to A, i.e. $1 \times A \cong A$.

So, in a sense, this unary category looks almost exactly like the original category $\mathcal{C}$. And it would be nice for example if given that Un_1 had coproduct, we would be able to conclude that $\mathcal{C}$ had coproduct. In fact, we can, because we get that there's an equivalence of categories between Un_1 and $\mathcal{C}$, denoted as

$$\mathrm{Un}_1 \simeq \mathcal{C}.$$

[00:33:40] *Remark.* If we say two morphisms in a category f and g are equal, we write

$$f = g. \qquad \text{(equality)}$$

If we have two objects of a category and we want to say they're isomorphic, we write

$$A \cong B. \qquad \text{(isomorphism)}$$

If we have two categories, and we want to say that they're equivalent, we write

$$\mathcal{C} \simeq \mathcal{D}. \qquad \text{(equivalence)}$$

Note that when we have equality, there's not other information that you could possibly want; but when we have an isomorphism, there is actually data involved and that there could be more than one isomorphism between two sets. For example, ther are two bijections between any two 2-element sets. Similarly, with equivalences of categories, we can have multiple different equivalences of categories, because the categories can have automorphisms in some sense.

It turns out that we can prove that Un_1 and $\mathcal{C}$ are equivalent categories, and in fact here we do get an isomorphism of categories, as on objects this is actually the identity. The construction is given by

$$\frac{f : 1 \times A \to B}{f \circ i : A \to B} \qquad \frac{g : A \to B}{g \circ \pi_2 : 1 \times A \to A \to B},$$

where $i : 1 \times A \to A$ is the isomorphism.

5 ESO., Full and Faithful [00:36:51]-[00:49:02]

[00:36:51] To show that this actually is an equivalence of categories, we could just construct the two functors and define the natural isomorphism. But there turns out to be a slightly easier method to do this. To understand this, we will first go back to basics and look at a simpler version of the same problem, which is a bijection of sets.

Let's now say we have a function f from set X to set Y:

$$f : X \to Y.$$

There are different definitions for f to be a bijection:

1. - It's surjective: $\forall y \in Y.\ \exists x \in X.\ fx = y$;
   - and it's injective: $\forall x_1, x_2 \in X.\ (x_1 = x_2) \iff (fx_1 = fx_2)$.
2. It has an inverse, i.e., it's an isomorphism in the category of sets: $\exists f^{-1} : Y \to X.\ ff^{-1} = \mathrm{id},\ f^{-1}f = \mathrm{id}$.
3. $\forall y \in Y.\ \exists! x \in X.\ fx = y$.

Note that often it's easier to establish for a given function that it's injective and surjective than concretely constructing the actual inverse. Now we have a generalization of this characterization of surjectivity and injectivity to categories and functors. It becomes more complex because we're "one dimension higher". We typically split it up into three properties rather than two.

Let $F : \mathcal{C} \to \mathcal{D}$ be a functor.

[00:41:15] **Definition 1.** *F is essentially surjective on* **objects** *(eso), if and only if*

$$\forall d \in \mathcal{D}.\ \exists c \in \mathcal{C}.\ Fc \cong d.$$

This is saying that for every object of the codomain category there exists an object of the domain category that maps to it. We don't want to require that these are two equal functors as it's too strict.

[00:42:37] Next,

Definition 2. *F is faithful, if and only if*

$$\forall c, c' \in \mathcal{C}.\ \forall f_1, f_2 \in \mathcal{C}(c, c').\ Ff_1 = Ff_2 \implies f_1 = f_2.$$

This is saying that a functor is faithful when the action on **morphisms** (hom-sets) is injective for each fixed domain and codomain. This is a more typical definition than the one for terms in the CT structure.

[00:43:42] And the third one is also in analog to part of the injective property:

Definition 3. *F is full, if and only if*

$$\forall c, c' \in \mathcal{C}.\ \forall g \in \mathcal{D}(Fc, Fc').\ \exists . g' \in \mathcal{C}(c, c').\ Fg' = g.$$

This is saying that a functor is full when the action on **morphisms** (hom-sets) is surjective for each fixed domain and codomain.

[00:44:52] Combining the last two, we often say that

Definition 4. *F is fully faithful (f.f.), if and only if*

$$\forall c, c' \in \mathcal{C}.\ F_1^{c,c'} : \mathcal{C}(c, c') \to \mathcal{D}(Fc, Fc') \text{ is a bijection.}$$

We say that full and faithful together are the analog of injectivity, because we are generalizing from the if-and-only-if between $x_1 = x_2$ and $fx_1 = fx_2$, to the bijection from the morphisms from $\mathcal{C}(c, c')$ to $\mathcal{D}(Fc, Fc')$. So the reason that we split from two concepts to three is because this notion of injectivity splits into full and faithful.

[00:46:10] Now, the main theorem that we have, is that

Theorem 1. *F is Fully Faithful, essentially surjective if and only if F is an equivalence of categories.*

Proof. Section 1.5 (Theorem 1.5.9, Page 31) in Emily Riehl's *Category Theory in Context*. □

Note that the "essential injectivity" on objects is implied, in that it's injective on isomorphism classes of objects, though it not actual injective on objects as we saw from the previous examples where any one-element pointed set gets sent to the empty set.

6 Proof of Par $\simeq$ Set$_*$ [00:49:03]-[00:54:12]

[00:49:03] Now we can use this theorem to prove that **Par** $\simeq$ **Set**$_*$.

Proof. Consider the functor,

$$- \uplus \texttt{err} : \mathbf{Par} \to \mathbf{Set}_*.$$

It's sufficient to show that it's full, faithful, and essentially surjective.

1) $- \uplus \texttt{err}$ is essentially surjective

Given $X \ni x$, we need to come up with a set that gets mapped to something isomorphic to it. As we did before, it's pretty easy to see that

$$X \ni x \longleftrightarrow (X - \{x\}) \uplus \texttt{err} \ni \texttt{err}$$

is a base point-preseving bijection, so it's an isomorphism in the category of pointed sets **Set**$_*$.

2) $- \uplus \texttt{err}$ is fully faithful

We need to show that there is a bijection between the hom-sets

$$\mathbf{Par}(X, Y) \cong \mathbf{Set}_*(X \uplus \texttt{err} \ni \texttt{err}, Y \uplus \texttt{err} \ni \texttt{err}).$$

We will show that the mapping we defined is injective and surjective.

For injectivity, if we took two partial functions (D_1, f_1) and (D_2, f_2), and they were equal functions, we got equal functions between these pointed sets. Then if we got equal functions between these pointed sets, we need to confirm that they have the same domain and then on those domains they're defined to be the same elements. This is pretty easy to see because we can see that the domain will be all of the elements of X that don't get sent to `err`. And so if they're equal functions, they will send the same points to the `err`, and then they'll also send the other points to the same value in Y.

For surjectivity, for any function $g \in \mathbf{Set}_*$, we need to construct (D_g, g'), a partial function from X to Y, that will give us back the g. We can just say that the domain of g is all the elements of X that don't get sent to `err`, and g' is defind to be the same as g on the remaining elements, which is essentially the same as our other functor.

$$\frac{g : X \uplus \texttt{err} \ni \texttt{err} \to Y \uplus \texttt{err} \ni \texttt{err}}{(D_g, g') : X \rightharpoonup Y} .$$

□

7 $\mathbf{Pred}(X) \simeq \mathbf{Mono}(X)$ [00:54:13]-[01:07:21]

[00:54:13] For the rest of class, I wanted to go over two other examples of equivalence of categories that are very common and come up in a lot of applications, and one of them is the generalization of the other.

Let's again start with the category of sets. On the one hand, for any set X, we can consider all the maps from X into the Booleans:

$$X \xrightarrow{P} \{\texttt{true}, \texttt{false}\}$$

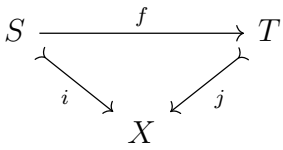
We can think of this as a predicate P on X. We can define a category $\mathbf{Pred}(X)$of all predicates on a fixed set X.

- $\mathbf{Pred}(X)$:
  - Objects P: $X \xrightarrow{P} \{\texttt{true}, \texttt{false}\}$
  - Morphisms (P, Q): $P \leq Q := \forall x \in X.\ (P(x) = \texttt{true}) \implies (Q(x) = \texttt{true})$

We have the objects are functions from X into Booleans, and the morphisms are poset relations.

On the other hand, we can look at another preorder, which is the all of the monomorphisms into the fixed set X, $\mathbf{Mono}(X)$.

- $\mathbf{Mono}(X)$:
  - Objects (S, i): $\begin{array}{ccc} S & & \\ & \searrow^{i} & \\ & & X \end{array}$

– Morphisms $((S,i),(T,j))$: 

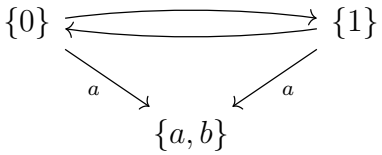
, $j \circ f = i$

The objects will be the pair (S,i), with the injective function i from S to X; a morphism from S to T is going to be a function f that makes the above diagram commutes.

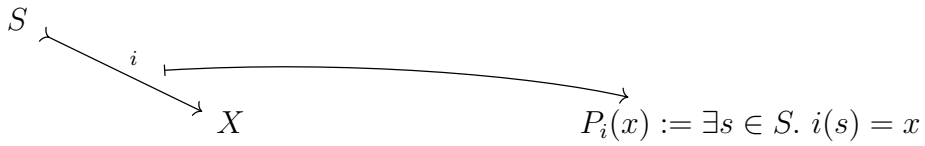
[01:00:35] Now claim that there is at most one morphism between any two objects of the category $\mathbf{Mono}(X)$. Recall that $i : S \rightarrowtail X$ is a monomorphism if $\forall f, f' : V \to S$, we have $(i \circ f = i \circ f') \implies (f = f')$. A monomorphism is kind of like a generalization of an injective function, and in the category of set, this is exactly the same as injective functions. Then since j is a monomorphism, if I have $f : S \to T$ and $g : S \to T$ that are equal when composing with j, then f and g will be equal.

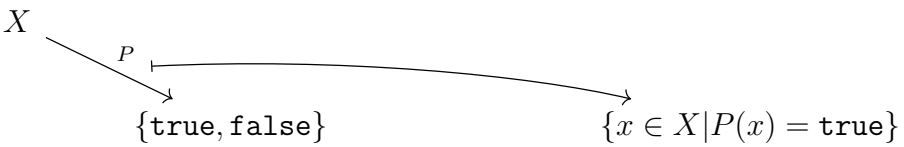
Notice that $\mathbf{Pred}(X)$ is a poset. Given two predicates P and Q, if for all $x \in X$, $P(x) = \texttt{true}$ if and only if $Q(x) = \texttt{true}$, then P and Q have to be the exactly same on every element of X, thus they're the same predicate.

On the other hand, $\mathbf{Mono}(X)$ is a only preorder, but not a poset. Because in the category of sets, I could have many different S that are equivalent by mapping into X. For example, given a set $\{a,b\}$, and I can have two one-element set $\{0\}$ and $\{1\}$ that both send to a, they are isomorphic but not equal.



We can prove that this preorder $\mathbf{Mono}(X)$ is equivalent as a category to this poset $\mathbf{Pred}(X)$. The construction is as follows. Given an injective function $i : S \rightarrowtail X$, we map that to a predicate on X which is $P_i(x) := \exists s \in S.\ i(s) = x$.





On the other hand, given a predicate $P : X \to \{\texttt{true}, \texttt{false}\}$, I can turn that into the subset of all elements $x \in X$ that satisfies the predicate.

Again, we get this nice property that one direction is actually equal to the identity because if you start with a predicate, you turn it into a subset, and then you get the

same predicate that you started with. However, if you start with a monomorphism $i : S \rightarrowtail X$, you turn it into a predicate, then the monomorphism into X you get is the map of S to its image of i. So those are equivalent notions.

In fact, this is a very common pattern. We can see that here the poset is the equivalence classes of the preorder up to order equivalence, as from a category theory perspective there is an equivalence of categories betwen them. And, what we also get is a concrete description in which we could replace the set of all monomorphisms into X by the poset of all subsets of X.

8 $\mathbf{Set}^X \simeq \mathbf{Set}/X$: [01:07:22]-[01:17:05]

[01:07:22]

The definition of $\mathbf{Pred}(X)$ is actually a way of characterizing what is special about the Booleans in the category of sets. In fact, in Topos theory, which is kind of a subfield of category theory, where they study the categories that generalize the structure that you have on the category of sets. The way that they define the analog of the set of propositions $\{\texttt{true}, \texttt{false}\}$ is exactly that it satisfies this property, so that you can define a category of monomorphism in any category. And one of the axioms of topos says that you have this object of propositions, which is that for any X, the morphisms into the object of propositions are equivalent to the monomorphisms into X. So if you alerady have defined functions between your sets, you can use this to define what the propositions should be.

Next is an example that is a generalization of this. Now let's take the entire category of sets. Instead of monomorphisms, consider arbitrary functions. The category is going to be called $\mathbf{Set}/X$.

- $\mathbf{Set}/X$:
  - Objects (S, p):
  - Morphisms $((S, p), (T, q))$: , $q \circ f = p$

So the objects are pairs of a set S and a function p from S into X. And the morphisms in this category are going to be again the commuting squares. Now there can be many functions that satisfies this because p and q are not assumed to be monomorphisms or epimorphisms.

On the other hand, we can take the category of sets, and we can take the set X and view it as a discrete category, i.e., a category whose objects are the elements of X and the only morphisms is the identity morphism. And then we can consider the category of functors from that into the category of sets, $\mathbf{Set}^X$. This is just a family

of sets indexed by X. So the objects are essentially functions from X into the sort of large set of all small sets.

- $\mathbf{Set}^X$:
  - Objects: $S_- : X \to \mathbf{Set}$
  - Morphisms $\forall x \in X.\ f_x : S_x \to T_x$

And the morphisms are natural transformations, but in this case, it means that for all $x \in X$, you have a function $f_x : S_x \to T_x$, where S, T are objects here.

[01:11:48] We can actually view this as a generalization of our category of predicates on X, and this is exacly the kind of generalization from sets to predicates that we have been seeing all semester. We can take any set and think of it as the set of proofs that some proposition is true, so we can view this as a kind of proof relevant version of a predicate. And the morphisms here is exactly the generalization of this property of the ordering between propositions. We can also generalize the equivalence between the predicates and the monomorphisms to an equivalence between these indexed families of sets over X and a function into X.

$$\frac{S_- : X \to \mathbf{Set}}{\{(x, s) | x \in X, s \in S_x\}}\ .$$

So the way we do that is as follows. Given $S_- : X \to \mathbf{Set}$, we want to construct some set and a function into X. We will define the set of pairs (x, s) such that $x \in X$ and $s \in S_x$. In type theoretic notation, this is written as a Σ type $\sum_{x:X} S_x$. The projection back to X is easy to define, which is just the first projection:

$$\begin{array}{ccc} \sum_{x:X} S_x & & \\ & \searrow^{\pi_1} & \\ & & X \end{array}$$

Note that for the notation of Σ types, if X and each S_- are all finite sets, then the cardinality will be

$$\left| \sum_{x:X} S_x \right| = \sum_{x \in X} |S_x|\ .$$

Then given a function $p : S \to X$, we can turn that into a family of sets by taking the inverse images of the elements of X.

$$\frac{p : S \to X}{p^{-1}(\cdot) : X \to \mathbf{Set}, \qquad p^{-1}(x) := \{s \in S | ps = x\}}\ .$$

Also, just like the correspondence between $\mathbf{Par}(X, Y)$ and $\mathbf{Set}_*$ can be used to define a universal property for the Booleans, this is one way to describe the large collection of sets. So if we have a way of classifying some subsets of these morphisms, we can

then define a universal object to have this similar kind of correspondence. So this one specifically being the property where we've already have this large set of all small sets, but this can be used in the same way to give an abstract characterization of these universal objects analogous to objective propositions in the previous example. If you've done dependent type theory before, this is essentially the semantics of the Σ types, given by generalization of this to other categories.

[01:17:05]