Article

Sheaf Mereology: Parts and Wholes in a Topos-Theoretic Setting

Firstname Lastname [1], Firstname Lastname [2] and Firstname Lastname [2,*]

[1] Affiliation 1; e-mail@e-mail.com
[2] Affiliation 2; e-mail@e-mail.com
* Correspondence: e-mail@e-mail.com; Tel.: (optional; include country code; if there are multiple corresponding authors, add author initials) +xx-xxxx-xxx-xxxx (F.L.)

Abstract

A single paragraph of about 200 words maximum. For research articles, abstracts should give a pertinent overview of the work. We strongly encourage authors to use the following style of structured abstracts, but without headings: (1) Background: place the question addressed in a broad context and highlight the purpose of the study; (2) Methods: describe briefly the main methods or treatments applied; (3) Results: summarize the article's main findings; (4) Conclusions: indicate the main conclusions or interpretations. The abstract should be an objective representation of the article, it must not contain results which are not presented and substantiated in the main text and should not exaggerate the main conclusions.

Keywords: mereology; fusions and integral wholes; sheaves; point-free topology; frames and locales; toposes; modality; merology logic; categorical logic

1. Category Theory

In this section, we cover the parts of category theory that we will use in the remainder of the paper. Readers familiar with category theory can skip this section.

Since at least Aristotle's time, philosophers have been interested in placing objects of the same kind into categories. Modern category theorists do this too: a category is a collection of objects that are all the same in kind. But category theory has a further requirement: not only must you tell us what the objects are in your category, you must also tell us about the maps between them, so that we know how to relate/compare those objects. Further, those maps must compose in well-behaved ways.

Definition 1 (Categories)**.** *A category $\mathbb{C}$ consists of the following data:*

1. *A collection of objects X, Y, Z, and so on, denoted $Objs(\mathbb{C})$*
2. *A collection of morphisms $f : X \to Y$, $g : Y \to Z$, and so on, denoted $Morphs(\mathbb{C})$, each with a designated domain (an object) and codomain (an object), such that:*
   - (a) *There is an identity morphism $id_X : X \to X$ for each object X.*
   - (b) *Morphisms compose, i.e., if $f : X \to Y$ and $g : Y \to Z$ are morphisms in $\mathbb{C}$ such that f's codomain matches g's domain, then their composite $g \circ f : X \to Z$ (pronounced "g after f") is a morphism in $\mathbb{C}$ too.*

The objects and morphisms of $\mathbb{C}$ must satisfy the following conditions:

(K1) *Composing with an identity has no effect, i.e. for any morphism $f : X \to Y$,*

$$id_Y \circ f = f \text{ and } f = f \circ id_X.$$

Received:
Revised:
Accepted:
Published:

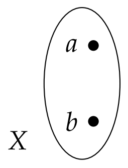
Citation: Lastname, F.; Lastname, F.; Lastname, F. Title. *Philosophies* **2025**, *1*, 0. https://doi.org/

(K2) Composition is associative, i.e., for any morphisms $f : X \to Y$, $g : Y \to Z$, and $h : Z \to W$,

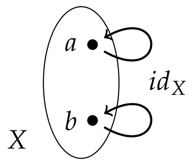
$$(h \circ g) \circ f = h \circ (g \circ f).$$

Example 1. *The category $\mathbb{S}et$ of sets and total maps has all sets for its objects and all total function between them for its morphisms. To check that it is a category, we need to confirm that it satisfies all the requirements.*

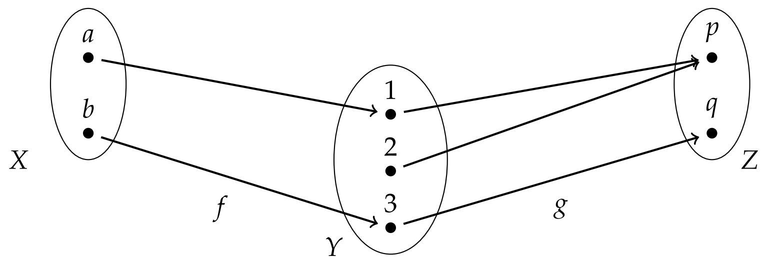
First, does every object have an identity? Yes, because for every set X, there is an identity function $id_X : X \to X$ given by $id_X(x) = x$. For instance, take the following set X:



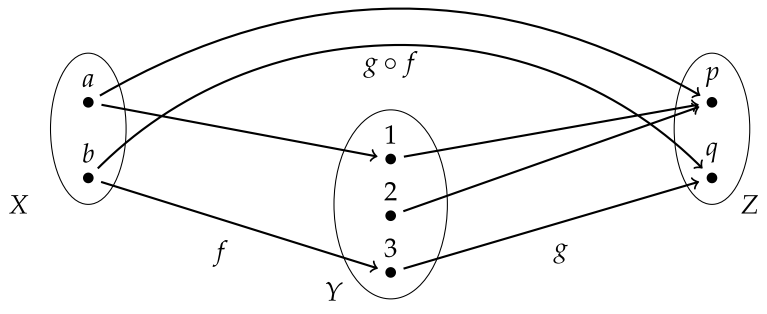
There is an identity function that sends each point of X to itself:



Do morphisms compose? Yes: the morphisms of this category are functions, and functions compose. For instance, take the following $f : X \to Y$ and $g : Y \to Z$:



Their composite $g \circ f : X \to Z$ is a function too:



Further, it is clear that composing with an identity has no effect, since identity functions do not shuffle around any points. It is also clear that function composition is associative: it does not matter if I take f first and then $h \circ g$, or $g \circ f$ first and then h. I get to the same results either way. So $\mathbb{S}et$ is a genuine category.

Example 2. *The category $\mathbb{T}op$ of topological spaces and continuous maps is the category whose objects are all topological spaces and whose morphisms are the continuous maps between them.*

Example 3. *The category $\mathbb{F}rm$ of frames and frame homomorphisms is the category whose objects are frames and whose morphisms are frame homomorphisms, i.e., maps that preserve arbitrary joins and finite meets.*

Example 4. *A poset can be seen as a category. For example, consider the powerset of $\{a, b\}$, ordered by inclusion. The Hasse Diagram looks like this:*

$$\begin{tikzcd}
 & \{a,b\} & \\
\{a\} \arrow[ur, no head] & & \{b\} \arrow[ul, no head] \\
 & \varnothing \arrow[ul, no head] \arrow[ur, no head] &
\end{tikzcd}$$

The fact that the elements of this diagram are sets is inessential. We could relabel them with arbitrary names to make the structure of the underlying poset obvious:

$$\begin{tikzcd}
 & s & \\
p \arrow[ur, no head] & & q \arrow[ul, no head] \\
 & r \arrow[ul, no head] \arrow[ur, no head] &
\end{tikzcd}$$

Hasse diagrams only show the minimal information needed to understand the poset. A poset is reflexive and transitive, but we don't draw lines for the reflexive and transitive steps, to avoid clutter. Thus, for instance, in this case, although $p \leqslant p$, we don't draw a line to represent it in the Hasse diagram. However, if we were to draw the full poset, we would get something that looks like this:

$$\begin{tikzcd}
 & s \arrow[loop above] & \\
p \arrow[ur] \arrow[loop left] & & q \arrow[ul] \arrow[loop right] \\
 & r \arrow[ul] \arrow[ur] \arrow[uu, bend left] \arrow[uu, bend right] \arrow[loop below] &
\end{tikzcd}$$

This is category. It's objects are the points (r, p, q, and s), and its morphisms are the arrows in the picture. It has identities (each reflexive loop), morphisms compose (because a poset is transitive), and so on.

This illustrates that the morphisms in a category need not be functions, nor do they need to be function-like. Here, we put a morphism between two objects just to state a fact about the poset: a morphism from p to s simply means $p \leqslant s$, i.e., that "p is lower than s in the ordering."

Definition 2 (Opposite categories)**.** *For any category $\mathbb{C}$, define the opposite category $\mathbb{C}^{op}$ as the category with the same objects as $\mathbb{C}$, but whose morphisms and composition are turned around. That is:*

1. *For any morphism $f : X \to Y$ in $\mathbb{C}$, the corresponding morphism in $\mathbb{C}^{op}$ is $f^{op} : Y \to X$.*
2. *For any morphisms $f : X \to Y$, $g : Y \to Z$, and their composite $g \circ f : X \to Z$ in $\mathbb{C}$, the correspond morphisms and composite in $\mathbb{C}^{op}$ are $f^{op} : Y \to X$, $g^{op} : Z \to Y$, and $(g \circ f)^{op} = f^{op} \circ g^{op} : Z \to X$.*

Example 5. *The category $\mathbb{L}oc$ of locales is defined as the opposite category of $\mathbb{F}rm$. Thus, the objects of $\mathbb{L}oc$ are the same objects as $\mathbb{F}rm$, which is why we can call a frame a locale or vice versa.*

1.1. Functors

Functors are maps between categories that preserve categorical structure. That is, a functor maps one category to another in such a way that it preserves the identities and composition of the original category, so that you end up picking out a kind of "image" of the first category in the second category.

Definition 3 (Functors). *A functor $F : \mathbb{J} \to \mathbb{C}$ is comprised of the following data:*

1. *A mapping of objects to objects, i.e., for each object $X \in \mathbb{J}$, $F(X)$ is an object in $\mathbb{C}$.*
2. *A mapping of morphisms to morphisms, i.e., for each morphism $f : X \to Y$ in $\mathbb{J}$, $F(f) : F(X) \to F(Y)$ is a morphism in $\mathbb{C}$.*

Further, these mappings must satisfy the following conditions:

(F1) *F preserves identities, i.e.,*

$$F(id_X) = id_{F(X)}.$$

(F2) *F preserves composition, i.e. for any $f : X \to Y$ and $g : Y \to Z$ in $\mathbb{C}$,*

$$F(g \circ f) = F(f) \circ F(g).$$

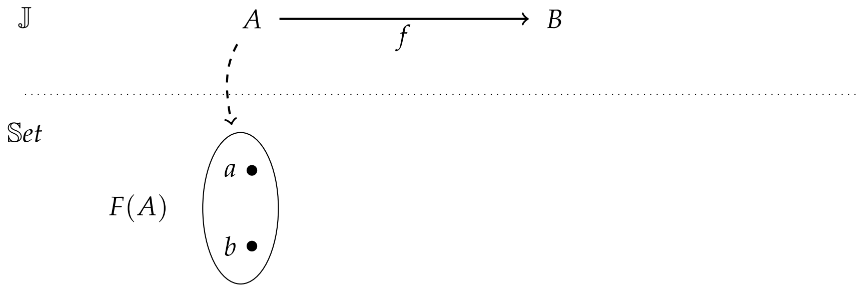
A functor $F : \mathbb{J} \to \mathbb{C}$ is also called a diagram, because it picks out $\mathbb{J}$-shaped figures of $\mathbb{C}$. $\mathbb{J}$ is then called the indexing category, because the objects and morphisms of F's image are indexed by the objects and morphisms of J.

A functor to or from an opposite category (e.g., $F : \mathbb{J}^{op} \to \mathbb{C}$ or $F : \mathbb{J} \to \mathbb{C}^{op}$) is called a contravariant functor because the morphisms go in opposite directions on either side of the functor.

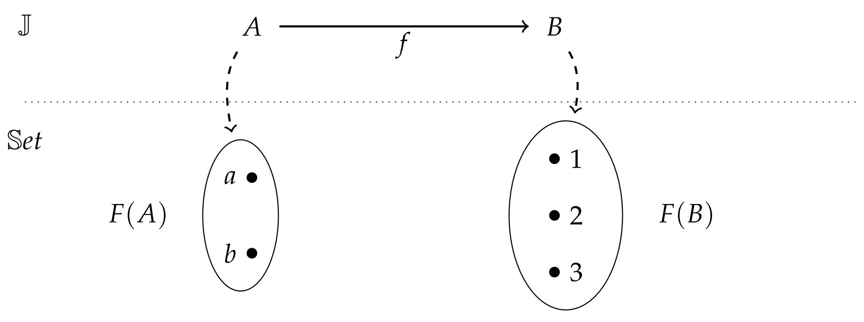
Example 6. *Suppose $\mathbb{J}$ consists of two objects and one non-trival morphism, something like this (we do not draw identity morphisms):*

$$\begin{tikzcd}\mathbb{J} & A \arrow[r, "f"'] & B\end{tikzcd}$$

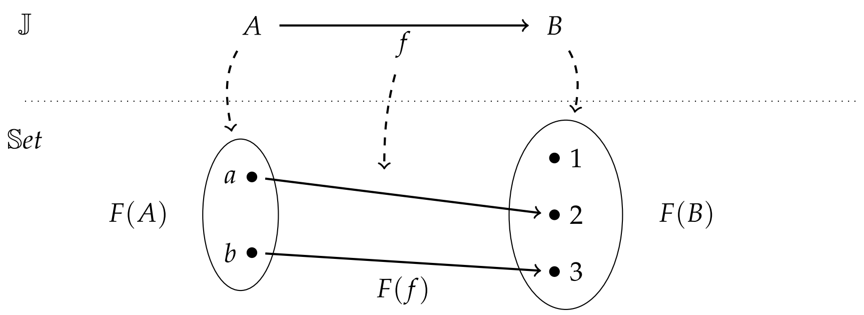
Then a functor F from $\mathbb{J}$ to $\mathbb{S}et$ uses $\mathbb{J}$ as a kind of template: it picks out a set $F(A)$ for A, a set $F(B)$ for B, and a function $F(f) : F(A) \to F(B)$ for f. For instance, to build such an F, for A we might pick the set $\{a, b\}$:



Then for B we might pick the set $\{1, 2, 3\}$:



Finally, for f we might pick the function that sends a to 2 and b to 3:



Or, to draw the same picture without the internal details:

$$\begin{tikzcd}
\mathbb{J} & A \arrow[rr, "f"'] \arrow[d, dashed] \arrow[drr, phantom, ""] & & B \arrow[d, dashed, "F"] \\
\mathbb{S}et & F(A) \arrow[rr, "F(f)"'] & & F(B)
\end{tikzcd}$$

This makes it clear that F picks out a $\mathbb{J}$-shaped piece of $\mathbb{S}et$: namely, a piece of $\mathbb{S}et$ that consists of two objects with one non-trivial morphism between them.

We could construct a different functor $G : \mathbb{J} \to \mathbb{S}et$ by picking different sets with a different morphism between them, in which case we would thereby pick out a different $\mathbb{J}$-shaped piece of $\mathbb{S}et$.

Remark 1. *Functors do not always pick out* isomorphic *$\mathbb{J}$-figures. A functor can collapse parts of $\mathbb{J}$ and still preserve its identities and composition. For instance, a functor can send all objects of $\mathbb{J}$ to the one-element set $\{*\}$ in $\mathbb{S}et$ and all morphisms of $\mathbb{J}$ to its identity $* \mapsto *$.*

Remark 2. *A contravariant functor from an indexing category $\mathbb{J}$ to $\mathbb{S}et$ is a presheaf. The signature of such a functor is $F : \mathbb{J}^{op} \to \mathbb{S}et$. To see why this is a presheaf, consider the following. $\mathbb{J}^{op}$ is the base category, and F assigns to each object $U \in \mathbb{J}^{op}$ a set. This is the data over the fiber of U. For each morphism $f : V \to U$ in $\mathbb{J}^{op}$, F sends f to a morphism going the other direction: $F(f) : F(U) \to F(V)$. These are the restriction maps. The functor laws F1 and F2 ensure that the restriction maps are unital and transitive as required.*

1.2. Natural transformations

A natural transformation maps one $\mathbb{J}$-figure to another. It does this by connecting up all of the objects of the first figure with those of the second in lock-step.

Visually, we can think of constructing a natural transformation as follows. Suppose we have an indexing category $\mathbb{J}$ that looks something like this:

$$\begin{tikzcd} & X \arrow[d, "f"'] \\ & Y \arrow[d, "g"'] \\ \mathbb{J} & Y \end{tikzcd}$$

Next, suppose we have a diagram F of $\mathbb{J}$ over on the left:

$$\begin{tikzcd} & & & X \arrow[d, "f"'] \arrow[dddl, dotted, bend right] \\ & & & Y \arrow[d, "g"'] \arrow[ddddll, dotted, bend right] \\ \mathbb{J} & F & & Y \arrow[dddl, dotted, bend right] \\ \mathbb{C} & & F(X) \arrow[dl, "F(f)"'] & \\ & F(Y) \arrow[dr, "F(g)"'] & & \\ & & F(Z) & \end{tikzcd}$$

Then suppose we have another diagram G of $\mathbb{J}$ over on the right:

$$\begin{tikzcd} & & & X \arrow[d, "f"'] \arrow[dddr, dotted, bend left] & & \\ & & & Y \arrow[d, "g"'] \arrow[dddrr, dotted, bend left] & & \\ \mathbb{J} & & & Y \arrow[dddr, dotted, bend left] & & G \\ \mathbb{C} & & F(X) \arrow[dl, "F(f)"'] & & G(X) \arrow[dr, "G(f)"] & \\ & F(Y) \arrow[dr, "F(g)"'] & & & & G(Y) \arrow[dl, "G(g)"] \\ & & F(Z) & & G(Z) & \end{tikzcd}$$

in other words, we have two diagrams of $\mathbb{J}$ side by side in $\mathbb{C}$:

$$\begin{tikzcd} & F(X) \arrow[dl, "F(f)"'] & & G(X) \arrow[dr, "G(f)"] & \\ F(Y) \arrow[dr, "F(g)"'] & & & & G(Y) \arrow[dl, "G(g)"] \\ & F(Z) & & G(Z) & \end{tikzcd}$$

We can map the one diagram (F) to the other (G) as follows. First, for each object in the left-hand $\mathbb{J}$-figure, pick a morphism that goes over to the corresponding object in the right-hand $\mathbb{J}$-figure. For instance, for the X-component, pick a morphism that goes from $F(X)$ to $G(X)$ (call it α_X):

$$\begin{tikzcd}
& F(X) \arrow[r, dashed, "\alpha_X"'] \arrow[ld, "F(f)"'] & G(X) \arrow[rd, "G(f)"] & \\
F(Y) \arrow[rd, "F(g)"'] & & & G(Y) \arrow[ld, "G(g)"] \\
& F(Z) & G(Z) &
\end{tikzcd}$$

Then, for the Y-component, pick a morphism that connects $F(Y)$ to $G(Y)$:

$$\begin{tikzcd}
& F(X) \arrow[r, dashed, "\alpha_X"'] \arrow[ld, "F(f)"'] & G(X) \arrow[rd, "G(f)"] & \\
F(Y) \arrow[rrr, dashed, "\alpha_Y"'] \arrow[rd, "F(g)"'] & & & G(Y) \arrow[ld, "G(g)"] \\
& F(Z) & G(Z) &
\end{tikzcd}$$

Finally, for the Z-component, pick a morphism that connects $F(Z)$ to $G(Z)$:

$$\begin{tikzcd}
& F(X) \arrow[r, dashed, "\alpha_X"'] \arrow[ld, "F(f)"'] & G(X) \arrow[rd, "G(f)"] & \\
F(Y) \arrow[rrr, dashed, "\alpha_Y"'] \arrow[rd, "F(g)"'] & & & G(Y) \arrow[ld, "G(g)"] \\
& F(Z) \arrow[r, dashed, "\alpha_Z"'] & G(Z) &
\end{tikzcd}$$

By doing this, we connect each object from the left-hand figure with an object on the right-hand figure.

A natural transformation is just such a family of connecting wires. It is a family because there are many of them: there is one such connecting wire for each "component" (object) of the figure. Thus, it is a family $\{\alpha_i\}_{i \in Objs(\mathbb{J})}$. We can think of it as a stack of bridges: each "bridge" (morphism) lets us travel from a component in the left-hand figure to the corresponding component in the right-hand figure.

However, we cannot pick just any set of "bridges" and call it a natural transformation. The choice of bridges has to be "natural," which means that our choice of bridges has to keep paths through the two figures in lock-step. In other words, we must be able to travel along a morphism in either figure, and it won't matter if we go over the bridge before or after we go down. We'll get to the same place either way.

For instance, in our picture, if we selected our connecting bridges correctly, then it won't matter if we go down $F(f)$ in the left figure and then go over the bridge α_Y, or whether we go over the bridge α_X first and then go down $G(f)$. Either way, we'll get the same results. In other words, this square of the picture must commute (i.e., both paths through the diagram must be equal):

$$\begin{tikzcd}
& F(X) \arrow[r, dashed, "\alpha_X"'] \arrow[ld, "F(f)"'] & G(X) \arrow[rd, "G(f)"] & \\
F(Y) \arrow[rrr, dashed, "\alpha_Y"'] & & & G(Y)
\end{tikzcd}$$

In order to qualify as a natural transformation, our choice of bridges has to be such that *all* such squares in the stack of bridges commute. This requirement is called the naturality condition. It is a fairly strict requirement. It is not always possible to find a natural transformation from one diagram to another.

Definition 4 (Natural transformations)**.** *Given two diagrams $F, G : \mathbb{J} \to \mathbb{C}$, a natural transformation α is a family of morphisms $\{\alpha_i : F(i) \to G(i)\}_{i \in Objs(\mathbb{J})}$ in $\mathbb{C}$. Each such α_i is called a "component" of α, or the "i-component" of α.*

Further, the components of α must be chosen in such a way that they satisfy the following naturality condition. For any morphism $f : X \to Y$ in $\mathbb{J}$, $\alpha_Y \circ F(f) = G(f) \circ \alpha_X$. In other words, for every morphism $f : X \to Y$ in $\mathbb{J}$, the following must commute:

$$\begin{tikzcd}
F(X) \arrow[r, dashed, "\alpha_X"'] \arrow[d, "F(f)"'] & G(X) \arrow[d, "G(f)"] \\
F(Y) \arrow[r, dashed, "\alpha_Y"'] & G(Y)
\end{tikzcd}$$

Remark 3. *Given an indexing category $\mathbb{J}$ and another category $\mathbb{C}$, the diagrams (functors) from $\mathbb{J}$ to $\mathbb{C}$ form a category, denoted $\mathbb{C}^{\mathbb{J}}$ or $[\mathbb{J}, \mathbb{C}]$. The objects of the category are the diagrams, and the morphisms are natural transformations. Since functors of the shape $F : \mathbb{J}^{op} \to \mathbb{S}et$ are presheaves, a category of $\mathbb{S}et$-valued diagrams $\mathbb{S}et^{\mathbb{J}^{op}}$ (alternatively written $[\mathbb{J}^{op}, \mathbb{S}et]$) is called a presheaf category, or a category of presheaves.*

1.3. Limits

TODO.

1.4. (Co)Terminals, (Co)Products, (Co)Pullbacks

TODO.

1.5. Exponentials

TODO? I think the fibrational setting is easier to describe. Maybe do that instead.

2. Topos Theory

In this section, we cover the parts of topos theory that we will utilize in the rest of the paper. Readers familiar with topos theory can skip this section.

A topos is often described as a category that has enough internal structure to it that you can do "sets"-like reasoning in it. In other words, it has an internal logic that very much resembles the kind of first-order logic that we often model in universes of sets.

However, it is important to recognize that the key feature of first-order logic is its ability to speak about *subsets* of sets. Indeed, in first-order logic, a predicate $P(x)$ is typically said to hold when x belongs to the subset of things that satisfy P. This is why one can say that first-order logic really boils down to a system for reasoning about the "parts" of sets (their subsets).

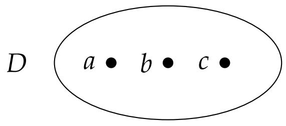
Topos theory generalizes this considerably. Topos theory characterizes categories in which we can reason not just about sets and their "parts" (subsets), but about a great variety of other kinds of objects and their "parts" (subobjects). In the category $\mathbb{S}et$, the objects are sets, and the "parts" or subobjects of a set are its subsets. But in other categories the appropriate "part" or subobject might be different. For instance, in the category of directed multi-graphs, the objects are directed multi-graphs, and the "parts" or subobjects of such a graph are its subgraphs.

Whatever sort of objects they may be, if they are the objects of a topos, then the topos has an internal logic that provides a consistent way to reason about those objects and their "parts" (subobjects). So, it is perhaps more appropriate to say that a topos is a category that has enough internal structure that we can do "parts"-like reasoning in it. A topos is a category that has an internal "parts"-like reasoning system built-in for free.

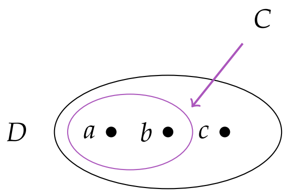
We will not give a definition of a topos, since the details are not important for our purposes. There are a number of different equivalent definitions of a topos, and all are easy to find in the literature (see, for instance, [1], [2], or [3]). All we need to know is that a topos has a rich structure: in particular, it has all limits and colimits (hence a terminal object, an initial object, products, coproducts, pullbacks, etc), it has all exponentials, and it has a subobject classifier (which we will introduce and discuss below). In what follows, we will focus on explaining how notions like predicates and logical connectives are realized in the internal logic of a topos.

2.1. Subobjects

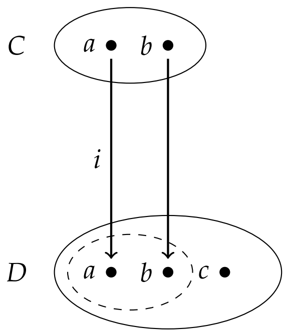
In the category $\mathbb{S}et$, it is natural to think of a "part" of a set as a subset. For example, suppose $D = \{a, b, c\}$:



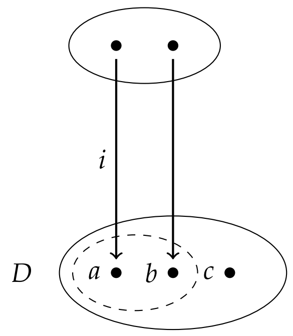
Now consider the "parts" of D. For instance, take $C = \{a, b\}$. We can picture the fact that C is a "part" of D by drawing a circle around C's elements to make it clear that the elements of C live "inside" D:



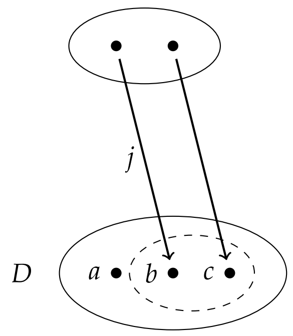
Thinking of C as a "part" of D in this way is fine, but it is very specific to the way that sets work. We can generalize by thinking about C as an insertion map $i : C \to D$ that injects C into D:



Once we think of a subset as an insertion map, it becomes clear that the names of the elements in C are irrelevant. In order to pick out the subset $\{a, b\}$, a function from any two-element set that picks out a and b will do:



Further, we can pick another subset of D with a different insertion map. For instance, we can pick out $\{b, c\}$ as follows:



The essential feature of an insertion map is that it does not collapse any information. In other words, it is a mere pass-through: i.e., it keeps things distinct and doesn't equate things that aren't already equated. Or, to put it the other way around, if two things aren't already equal, an insertion map won't make them equal. A morphism that has this property is called a monomorphism.

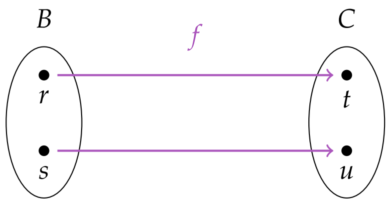
Definition 5 (Monomorphism)**.** *A morphism $i : C \to D$ in $\mathbb{C}$ is a monomorphism iff:*

$$\forall B \in \mathbb{C} \text{ and } \forall f, g : B \to C \in \mathbb{C}, f \neq g \text{ implies } i \circ f \neq i \circ g.$$

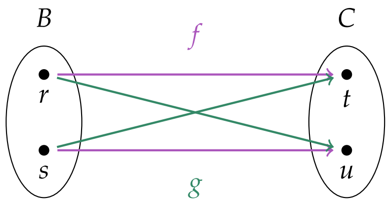
Equivalently:

$$\forall B \in \mathbb{C} \text{ and } \forall f, g : B \to C \in \mathbb{C}, i \circ f = i \circ g \text{ implies } f = g.$$

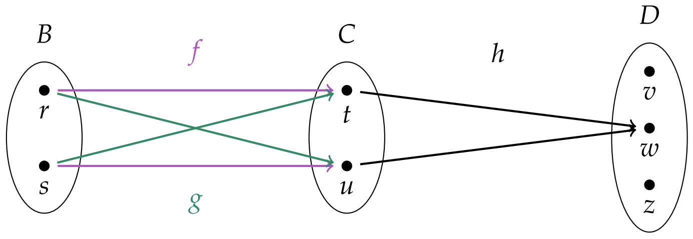
Example 7. *Consider a function $f : B \to C$ that looks like this:*



Suppose we also have another, non-equal function $g : B \to C$:

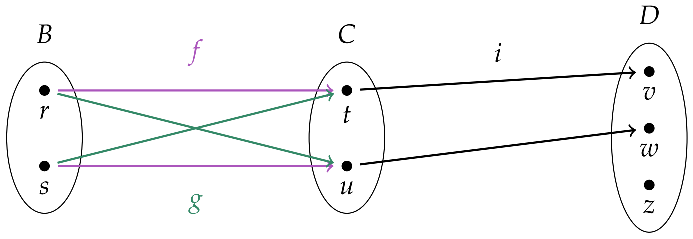


Since f and g are not equal, they send their inputs to different places. However, if we compose them with a function that collapses points, we can make them send their inputs to the same place. For instance, suppose we follow f and g with the following function $h : C \to D$:



Since, h collapses information — that is to say, since it sends both t and u to the same output w — by composing f and g with h, we can make them send their inputs to the same place too (namely, w). Since we can make a non-equal f and g yield equal results by composing them with h, we know that h therefore collapses information, and so h is not *a monomorphism.*

Now consider what happens when we compose f and g with a monomorphism $i : C \to D$:



Here, we can see that i is just a pass-through: it does not collapse any points but rather simply passes on whatever f and g give it, keeping the inputs it receives distinct. So unlike h, i cannot ever help f and g send their inputs to the same place.

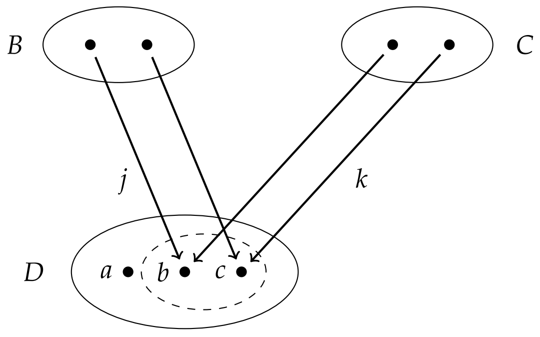
Further, we can see that i will behave this way with any *pair of non-equal morphisms f and g: if they already result in distinct outputs, i will keep them distinct. That is what it means for i to be a monomorphism.*

The previous examples were taken from the category $\mathbb{S}et$, but the definition of a monomorphism does not depend on the nature of the sets and functions involved. It works in any category.

Monomorphisms therefore give us a general way to talk about the subobjects of objects, in any category whatever. We can thus identify the subobjects of any object with the monomorphisms into it.

Definition 6 (Subobject)**.** *Given an object D in a category $\mathbb{C}$, a subobject i of D is a monomorphism $i : C \to D$.*

Different monomorphisms can pick out the same subobject. For instance, in the following picture, both j and k pick out the same subobject of D:

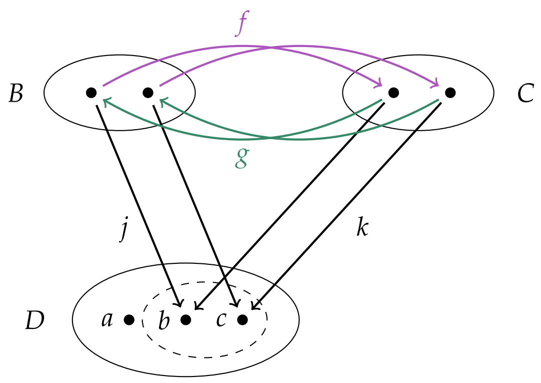


However, we can treat these two monomorphisms as equivalent subobjects if there is a way to translate back and forth between them. In particular, if there are morphisms between B and C that let us go back and forth, so that it doesn't matter which monomorphism we use, then we say $j \equiv k$.

Definition 7 (Equivalent Subojbects)**.** *Given subobjects $j : B \to D$ and $k : C \to D$, we say j and k are equivalent, denoted $j \equiv k$, iff there are morphisms $f : B \to C$ and $g : C \to B$ such that:*

$$k \circ f = j \quad \textit{and} \quad k = j \circ g.$$

Example 8. *Consider the pair of functions f and g that "translate" between B and C:*



Here, you can go over f and then down k and that yields the same result as just going down j. Similarly, you can go over g and then down j, and that yields the same result as just going down k. Hence, j and k are equivalent subobjects. t

Remark 4. *Definition 7 identifies an equivalence relation on the subobjects of any given object. Hence, we can partition all of the subobjects of that object into equivalence classes. Then we can pick any monomorphism from any such class and we can speak of it as "the" subobject, even though there are actually many equivalent such monomorphisms that pick out the same subobject. For convenience, in what follows we will usually speak of individual monomorphisms in this way, even though we often mean the equivalence class of such monomorphisms.*

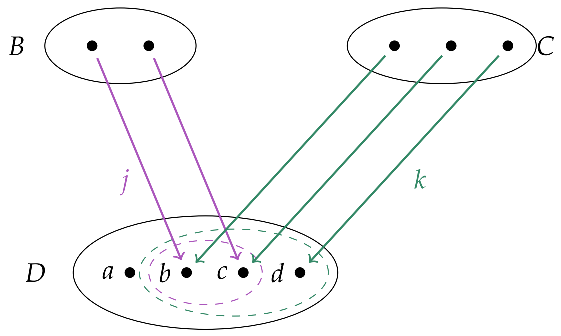
Some subobjects are smaller parts of bigger subobjects. If we think of sets, this is natural: some subsets are subsets of other subsets, e.g., $\{a\}$ is a subset of $\{a, b\}$, which in turn is a subset of $\{a, b, c\}$. So, $\{a\}$ and $\{a, b\}$ are both subobjects of $\{a, b, c\}$, but $\{a\}$ is in a certain sense "smaller" than $\{a, b\}$ and thus contained in it.

This generalizes to the monomorphism-theoretic approach to subobjects too. One subobject $j : B \to D$ is contained in another one $k : C \to D$ if the smaller one can be routed through the bigger one. If you can route the smaller one through the bigger one, then the smaller one must be a part of the larger one, since whatever else the bigger one inserts into D, it at least inserts the smaller one's contents.

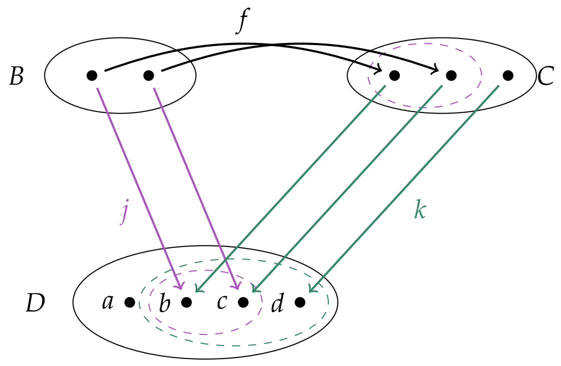
Definition 8 (Subobject Containment)**.** *Given monomorphisms $j : B \to D$ and $k : C \to D$, we say that j is contained in k (or j is "in" k for short), denoted $j \subseteq k$, iff there is a morphism $f : B \to C$ such that*

$$j = k \circ f.$$

Example 9. *Consider the following monomorphisms j and k that pick out two different subobjects of D:*



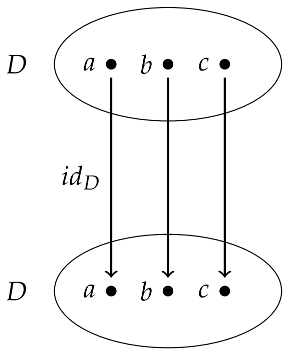
Now suppose we have a morphism $f : B \to C$ as follows:



Here, f routes j through k: if you go over f and then down k, you pick out the same subobject as j. So, whatever else k might inject into D, it at least injects the contents of j. Hence, we can say that j is contained in k, i.e. $j \subseteq k$.

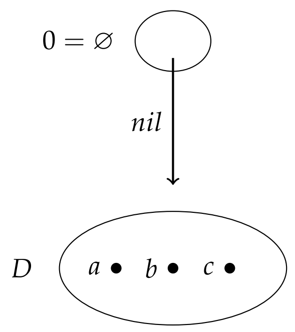
If you take all of the subobjects of an object and order them by inclusion, the result is a lattice. Thus, in any given topos, each object has an associated lattice of subobjects, ordered by inclusion. The meets and joins are as expected, namely the intersection and union of subobjects. There are even largest and smallest subobjects at the top and bottom ends of the lattice.

The largest subobject of an object is of course the whole object itself. In other words, it is just the identity (which is a monomorphism). For instance, in the following picture, id_D picks out the largest subobject of D, namely all of D:



By contrast, the smallest subobject of D is the monomorphism that inserts the least amount of information into D, namely no information at all. In other words, the smallest subobject of D is the empty one. This corresponds to the morphism from 0, the initial (empty) object of the topos, to D.

For example, in $\mathbb{S}et$, the initial/empty object is the empty set $\varnothing$, and there is exactly one morphism from $\varnothing$ to any other object D, namely the empty function *nil* that inserts nothing:

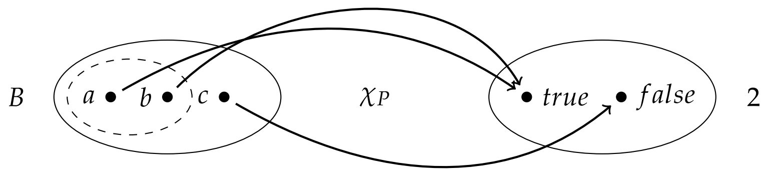


Definition 9 (Largest and smallest subobjects)**.** *For any given object D in a topos, the smallest subobject of D is the empty one, namely $nil : 0 \to D$, while the largest subobject is the identity $id_D : D \to D$.*

Remark 5. *Given an object D, the lattice of its subobjects is typically external to the topos that D lives in. For instance, suppose D is an object in $\mathbb{S}et$. The lattice of D's subobjects is not an object of $\mathbb{S}et$, since it is a lattice. It lives in a category of lattices, not the category of sets. It also counts as a category in its own right, since a lattice is a poset and every poset can be seen as a category (see Example 4). However, the lattice of D's subobjects is isomorphic to the exponential object Ω^D. So, although the lattice itself is external to the category, an internal version of it lives inside the topos as the exponential.*

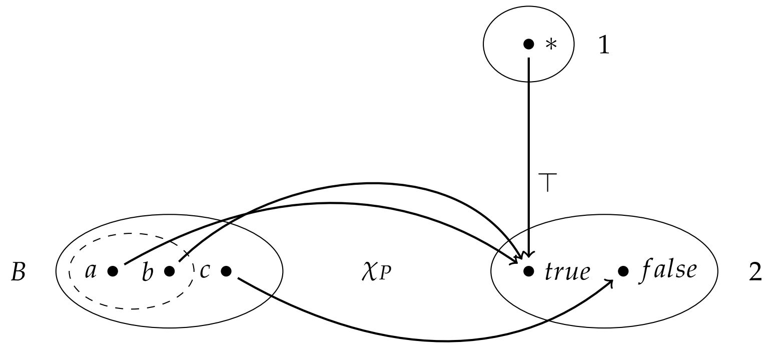
2.2. Subobject classifiers

A predicate P on a domain B is often defined as a function, let's call it χ_P, from B to $2 = \{true, false\}$, where elements of B that satisfy the predicate P are sent to "$true$" and those that don't are sent to "$false$." For instance, suppose we have a set $B = \{a, b, c\}$, and a predicate P with $P(a) = true$, $P(b) = true$, and $P(c) = false$. For concreteness, suppose B is the set of students in a class, and $P(x)$ is the predicate expressing that a student is passing the class. As a picture:



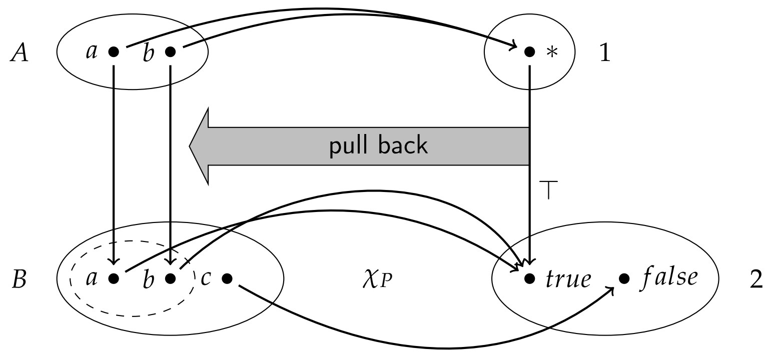
In essence, χ_P is a characteristic function: it "classifies" a subset A of B by telling us which elements of B belong in that subset A and which ones do not. In this case, the subset that gets classified by χ_P is $A = \{a, b\}$, i.e., the subset of students who are passing the class. This subset is precisely the extension of the predicate P — it is the set of elements in B that satisfy the predicate P, i.e., the set of students in the class who are passing.

Categorically, we can find A through a pullback. The key feature of a characteristic function like χ_P is that it sends the elements that it classifies to $true$. So, let's use a morphism from 1 (call this morphism $\top$ for "true") to pick out $true$ in 2:



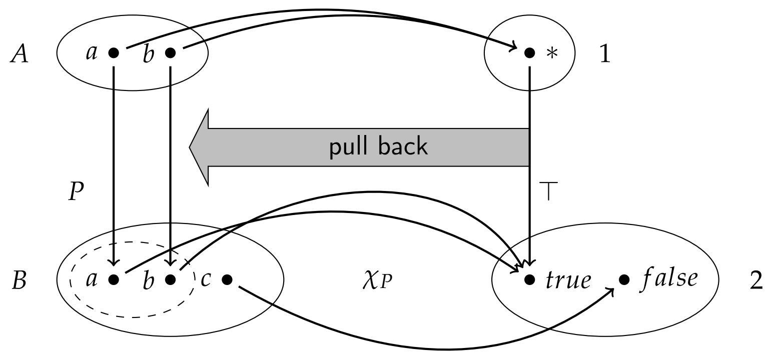
349

Next, pull back $\top : 1 \to 2$ along χ_P to get the preimage of χ_P on *true*: 350



351

This gives us another way to think about the predicate P. It is the map on the left-hand side of the pullback square that inserts A into B. In other words, it is the subobject (monomorphism) on the left-hand side of the pullback. Let's call it $P : A \to B$: 352 353 354



355

P and χ_P give us two perspectives on the same thing. Taken in itself, P identifies a subset of B (e.g., a subset of students in the class). By contrast, χ_P has more of a logical interpretation: given a student x in B, it tells you if they satisfy the predicate P (i.e., it tells you if it is true that they are passing or not). The pullback makes it clear how these two maps — P and χ_P — are essentially related to truth values. 356 357 358 359 360

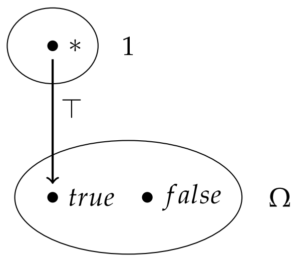
What is crucial about this pullback construction is the object 2 of truth-values on the right, along with the monomorphism $\top$ that picks out the "true" part of 2. In $\mathbb{S}et$, we can pull back $\top$ along any morphism into 2 to identify the corresponding predicate on 2. 361 362 363

This construction generalizes to other categories beyond $\mathbb{S}et$. In other categories too, it often happens that we have an object of truth values, call it Ω, and a monomorphism $\top : 1 \to \Omega$ that picks out the "true" part of Ω, which is such that we can pull back along any morphism into Ω to get a predicate on it. 364 365 366 367

Definition 10 (Subobject Classifier)**.** *In a topos $\mathbb{E}$, a subobject classifier is an object, denoted Ω, along with a morphism $\top : 1 \to \Omega$ which is such that, for any monomorphism $P : A \to B$, there is a unique morphism $\chi_P : B \to \Omega$ that makes the following a pullback:*

$$\begin{tikzcd}
A \arrow[r] \arrow[d, "P"'] & 1 \arrow[d, "\top"] \\
B \arrow[r, "\chi_P"'] & \Omega
\end{tikzcd}$$

Example 10. *In $\mathbb{S}et$, as we saw, Ω is the set of boolean truth values $\{true, false\}$, with $\top : 1 \to \Omega$ picking out true:*



Note that $\top$ is a monomorphism. It picks out a genuine subset of Ω: namely $\{true\} \subseteq \Omega$.

2.3. An Extended Example

In more complicated categories, the subobject classifier works just as it does in $\mathbb{S}et$, but the set of truth values might be different, and the morphisms may be more complex. This is particularly useful when it is so binary a matter whether something satisfies a predicate.

In this section, we will consider how the subobject classifier works in a category of diagrams. We will omit proofs, and proceed in an expositional style. Readers familiar with subobject classifiers can skip this section.

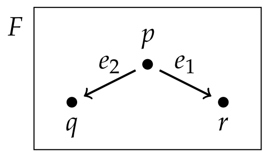
The diagrams we are interested in here are the ones whose indexing category $\mathbb{J}$ has the following shape:

$$\begin{tikzcd}
E \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & V
\end{tikzcd}$$

$\mathbb{S}et$-valued diagrams of this shape (i.e., functors from $\mathbb{J}$ to $\mathbb{S}et$) correspond to directed multi-graphs. Each such functor sends E to a set of edges, it sends V to a set of vertices, and what it picks for s and t send edges to their respective source and target vertices. For instance, let $F : \mathbb{J} \to \mathbb{S}et$ be given as follows:

- $F(E) = \{e_1, e_2\}$
- $F(V) = \{p, q, r\}$
- $F(s)(e_1) = p$, $F(s)(e_2) = p$
- $F(t)(e_1) = r$, $F(t)(e_2) = q$

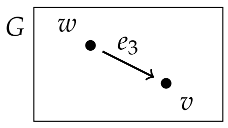
If we draw that graph, we get this:



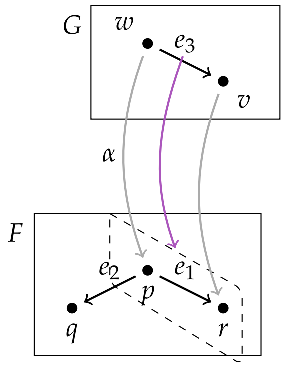
Now suppose we have another functor $G : \mathbb{J} \to \mathbb{S}et$, given as follows:

- $G(E) = \{e_3\}$
- $G(V) = \{w, v\}$
- $G(s)(e_3) = w$
- $G(t)(e_3) = v$

If we draw G as a graph, we get this:



G can be inserted into F as a subgraph. Since F and G are diagrams in the category $\mathbb{S}et^{\mathbb{J}}$, morphisms between them are natural transformations. Hence, an insertion map $\alpha : G \to F$ will be a natural transformation that inserts G directly into F without collapsing any of G. There are two such insertions here, but let us pick the one that sends w to p, v to r, and e_3 to e_1:



Now that we have picked out a subobject G of F, suppose next that we want to define a predicate P on F whose extension is precisely the subobject G. For instance, suppose F describes a fork p on a bike path, with e_1 going to the park r and e_2 going to the museum q. Given this interpretation of F, let us introduce a predicate P that means something like "is on the route to the park." The extension of this predicate is exactly the subgraph G.

But how exactly do we define such a predicate, formally? With sets, a predicate takes a point (an element of a set) as input and then it tells you in response whether that point satisfies the predicate or not. More exactly, it takes a point x and returns "$true$" or "$false$," depending on whether x is in the extension (subobject) of the predicate.

To do something similar for graphs, we have to generalize:

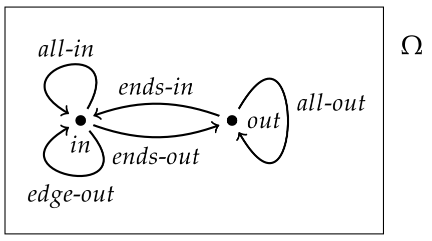
- A predicate on sets takes only *one* kind of input (a point in a set), but a predicate on graphs needs to take *two* kinds of inputs (vertices and edges).
- A predicate on sets tells you if a point is in the *subset* of the predicate's domain, but a predicate on graphs needs to tell you if a vertex or edge is in the *subgraph* of the predicate's domain.
- A predicate on sets gives you a binary "in" or "out" answer (either the point is *in* the predicate's extension or it is *out*), but a predicate on graphs needs to give a more variegated answer:
  - With respect to vertices, a given vertex is either inside or outside of a subgraph.
  - With respect to edges, we have five options:
    - The edge lives entirely outside of the subgraph (its source and target vertices are not part of the subgraph at all).

- The edge starts outside the subgraph and ends up inside the subgraph (its source is outside the subgraph, and its target is inside the subgraph).
- The edge starts inside the subgraph and ends up outside the subgraph (its source is inside the subgraph, but its target is outside the subgraph).
- The edge travels outside the subgraph and then comes back in (its source and target are inside the subgraph, but the edge itself is outside the subgraph).
- The edge lives entirely inside the subgraph (its source and target vertices are inside the subgraph, and the edge itself stays inside the subgraph and does not travel outside the subgraph).

This makes it clear that the logic of graphs is more discriminating than the logic of sets. With a predicate on sets, if you ask it whether a given input satisfies a predicate, it gives you a simple yes or no answer, because the given element either is or is not inside the given subset. This makes sense, of course. There is no more structure to sets than the points, so there is no more to say about the matter beyond stating whether the given point is or is not in the extension of the predicate.

A graph is different. There is much more structure to a directed graph, and so if you ask a predicate whether a given input (a vertex or an edge) satisfies that predicate, it cannot give you such a simple answer. It can tell you whether a given vertex falls inside the extension of the predicate with a yes/no answer, but if you ask it about an edge, it cannot give you a simple yes/no answer. It must instead tell you the degree to which the edge falls inside the extension of the predicate, since an edge can be partly in, partly out, etc.

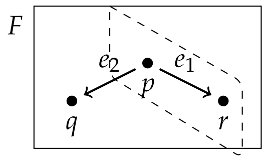
A simple binary set $\{true, false\}$ is thus not sufficient to serve as the object of truth values in this category of directed multi-graphs. Instead, Ω in this category has to look like the following:



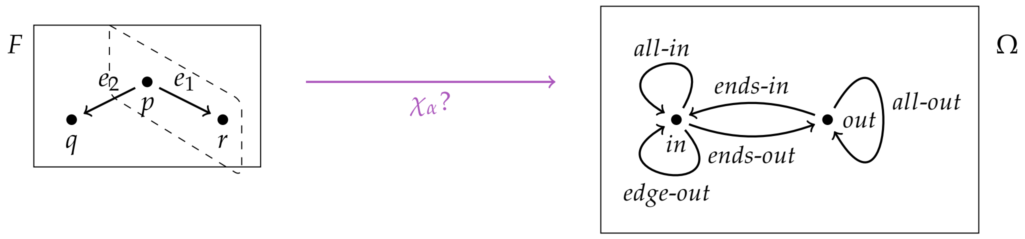
There are two vertices in this graph, labeled *"in"* and *"out,"* and they represent whether a given vertex is inside or outside of the given subgraph. There are five edges, each of which represents one of the five options described above:

- The edge labeled *"all-out"* represents an edge that is outside the given subgraph and whose vertices are outside it too. Notice that both its source and target vertices are *"out,"* i.e., they are vertices that live outside the given subgraph.
- The edge labeled *"ends-in"* represents an edge that starts outside the given subgraph and ends inside the subgraph. Notice that its source vertex is *"out"* and its target vertex is *"in."*
- The edge labeled *"ends-out"* represents an edge that starts inside the given subgraph and ends outside the subgraph. Notice that its source vertex is *"in"* and its target vertex is *"out."*
- The edge labeled *"edge-out"* represents an edge that starts inside the given subgraph, travels outside the subgraph, and then travels back in. Notice that its source and target vertices are both *"in."*
- The edge labeled *"all-in"* represents an edge that lives inside the given subgraph and whose vertices are also inside the subgraph.

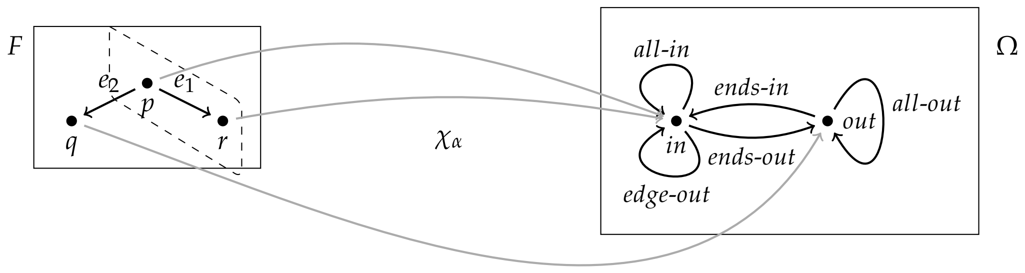
With this version of Ω, we can classify all of the different ways a vertex or edge in a predicate's domain can be part of a a given subgraph. For instance, consider F and the subgraph G:



With sets, we can define a characteristic morphism into Ω that "classifies" which points in the predicate's domain belong in the given subset. But in this setting, how do we define a characteristic morphism χ_α into Ω that "classifies" which parts of F belong in G?

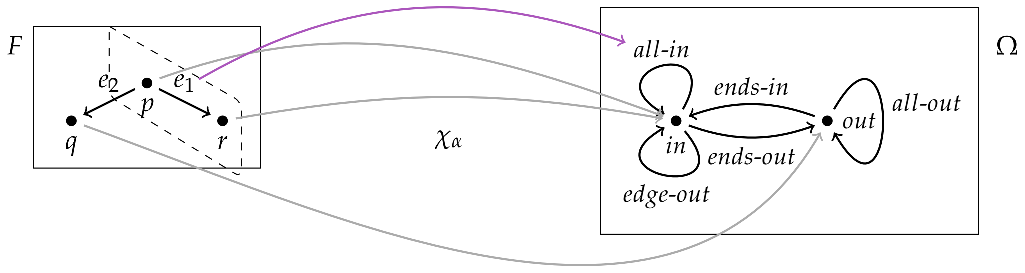


First, we send the vertices p and r to "*in*" (since they are inside G) and we send the vertex q to "*out*," since it is outside G:



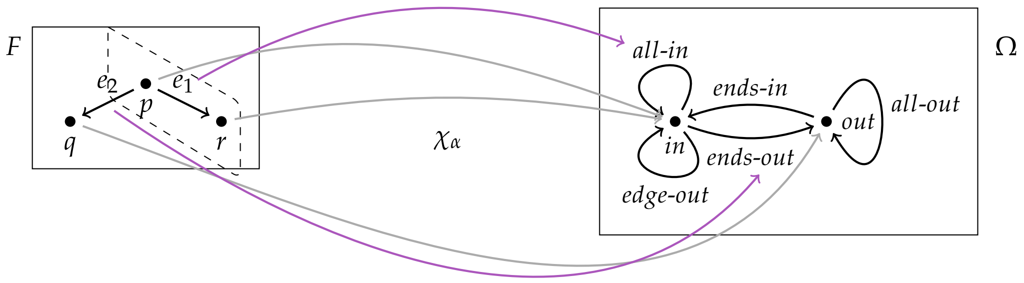
Hence, by this classification, p and r fall under the extension of the predicate, while q does not. This matches what we expect. Is the fork p on the route to the park (i.e., does $P(p)$ hold)? Yes, and the same goes for the park q ($P(q)$ holds). But the museum q is not on the route to the park, so $P(q)$ does not hold.

Next, we send e_1 to "*all-in*" because it lives inside G:



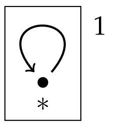
This matches our intuitions about the meaning of the predicate "is on the route to the park" too. Since e_1 is the path from the fork p to the park r, it makes sense that $P(e_1)$ holds.

Finally, we send e_2 to "*ends-out*" because although it begins *inside* of G (at p), it leaves G and ends up outside of G (at q):



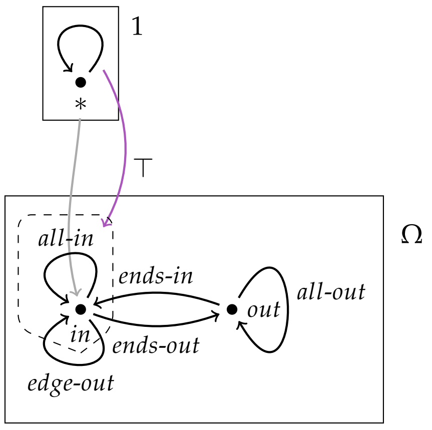
This also classifies e_2 appropriately. Does the path to the museum fall on the route to the park? Well, the beginning part of it does, at the fork p. But after that, it does not. So $P(e_2)$ does not hold unequivocally. But nor does it fail to hold unequivocally. The edge e_2 is classified as being partly in and partly out of the extension of the predicate.

What is the unequivocal "true" part of Ω? That is to say, what is the subobject of Ω that represents "true" unequivocally? With sets, we pick out the "$\{true\}$" subobject of Ω with a monomorphism from the terminal object 1. In this category, the terminal object is the one-vertex graph with a single edge:

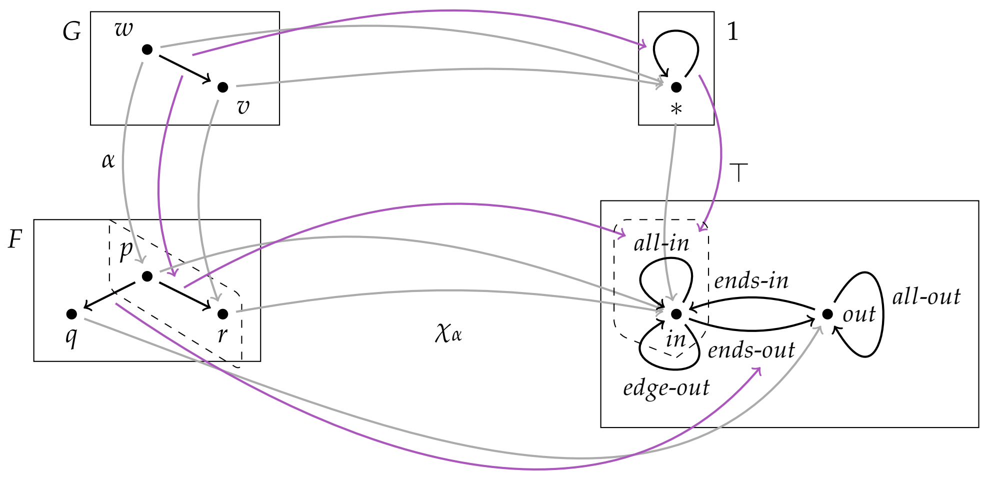


The $\top$ monomorphism from 1 into Ω has to pick out the "true" subobject of Ω, but in this case that means it will have to pick out the subgraph of Ω that represents when a given input (vertex or edge) is unequivocally inside the extension of the predicate.

Some of the possible cases represented by Ω are not entirely "in" the extension of the predicate, e.g. when an edge is partly in, and partly out. The only *unequivocal* case is when an edge and both of its vertices are *entirely* inside the given subgraph. Thus, $\top$ must pick out the subgraph consisting of the vertex labeled "*in*" and the edge labeled "*all-in*":



This is the subobject classifier for the category at hand, and it yields the correct characteristic morphisms when we pull $\top$ back. For instance, in the case of F and G:



2.4. Logical Connectives

Logical connectives like "*and*" and "*or*" operate on pairs of formulas. For instance, "*and*" takes two formulas and returns "*true*" if both are true. In a topos, formulas can be thought of as morphisms into Ω (in a sense we will make precise later). Accordingly, the topos-theoretic versions of "*and*" and "*or*" are morphisms from $\Omega \times \Omega$ to Ω. Similarly, negation (i.e., "$\neg$") operates on a single formula, and its topos-theoretic version is likewise a morphism from Ω to Ω.

In general, each of the connective operations can be defined as the characteristic morphisms of the expected combinatorial operations on Ω. For instance, "*and*" is the characteristic morphism of the subobject that picks both true parts of $\Omega \times \Omega$, while "*or*" is the characteristic morphism of the subobject that picks either of the true parts of $\Omega \times \Omega$.

These operations impose a Heyting algebra structure on Ω: there are operations that correspond to top, bottom, join ("*or*"), meet ("*and*"), complement ("$\neg$"), and so on. This of course is also the case for the operations with set-based logic, except that the algebraic structure of the logic is Heyting (intuitionistic) for toposes in general, and not Boolean (although there are toposes that are Boolean).

In the rest of this section, we will describe some of these operations. For further details, see for instance [1, pp. TODO: find the pages (chapter 6, I think)] or [3, pp. 349–352].

2.4.1. $\top$ and $\bot$ in Ω

One of the themes of category theory is that morphisms into objects serve as probes that we use to study those objects. For instance, a morphism $f : B \to C$ is a probe: f gives us one way to look at C through B's eyes, so to speak.

This also means that the probe you pick determines how discriminating your observations can be. A coarse probe will only reveal coarse information about the object under scrutiny, whereas a more finely variegated object will be able to make finer observations.

In a topos, the terminal object 1 is special because it is the coarsest non-trivial probe. 0 is the coarsest probe, since it provides no information at all, but 1 provides the next step up. 1 can tell you only if an object has something or not. Other objects will see information indexed by their own further distinctions. But 1 is entirely unrefined, so to speak.

Because of this, we can think of 1 as the "global" lens. Anything we can see with 1 is a "global" fact about the topos rather than something that holds only relative to some finer level of contextualization.

If we use 1 as a lens to probe Ω, we will only be able to see global facts about Ω. That is to say, 1 simply cannot see anything finer than that when it interacts with Ω.

In some categories, Ω is very simple. For instance, in $\mathbb{S}et$, it contains only two elements: "$true$" and "$false$." But in other categories, it is more complex. For example, as we saw with the category of directed multi-graphs, Ω carries a number of intermediate notions of partial truth.

However sophisticated Ω's handling of truth may be, 1 cannot perceive anything except for the extremes. It can only see the "entirely true" and "entirely false" cases, even if Ω itself can discriminate many other cases in between.

We can use the subobject classifier itself to identify which parts of itself 1 picks out as the "entirely true" and "entirely false." The subobject classifier classifies the subobjects of objects. In particular, for any object X, the morphisms $X \to \Omega$ correspond exactly to the subobjects of X because each such morphism is the characteristic arrow that classifies the corresponding suboject. If we substitute 1 in for X, it follows that the morphisms $1 \to \Omega$ correspond exactly to the subobjects of 1.

First, consider the largest subobject of 1. As noted in Definition 9, the largest subobject of an object is selected by its identity morphism. So, id_1 is the monomorphism that picks out the largest subobject of 1:

$$\begin{tikzcd}
1 \arrow[d, "id_1"'] \\
1
\end{tikzcd}$$

If we pull back from the subobject classifier, there will be a unique characteristic morphism that classifies that subobject:

$$\begin{tikzcd}
1 \arrow[r] \arrow[d, "id_1"'] & 1 \arrow[d, "\top"] \arrow[dl, phantom, "\text{pull back}"] \\
1 \arrow[r, "\chi_{id_1}?"'] & \Omega
\end{tikzcd}$$

What could χ_{id_1} be? In order to make the pullback square commute, it has to send all of 1 on the bottom left to the same thing that $\top$ picks out in Ω on the right. But that is exactly what $\top$ does. Hence, χ_{id_1} just *is* the morphism $\top$:

$$\begin{tikzcd}
1 \arrow[r] \arrow[d, "id_1"'] & 1 \arrow[d, "\top"] \arrow[dl, phantom, "\text{pull back}"] \\
1 \arrow[r, "\chi_{id_1} = \top"'] & \Omega
\end{tikzcd}$$

Now return to the idea that 1 can only discern the global elements of Ω. What this means is that $\top$ picks out exactly that part of Ω that represents being "entirely true."

What about the smallest subobject of 1? As noted in Definition 9, morphisms from 0 pick out the smallest subobjects. Hence, $nil : 0 \to 1$ will pick out the smallest subobject of 1:

$$\begin{tikzcd}
0 \arrow[d, "nil"'] \\
1
\end{tikzcd}$$

If we pull back from the subobject classifier, there will be a unique characteristic morphism that classifies this subobject:

$$\begin{tikzcd}
0 \arrow[r] \arrow[d, "nil"'] & 1 \arrow[d, "\top"] \arrow[l, phantom, "\text{pull back}"] \\
1 \arrow[r, "\chi_{nil}?"'] & \Omega
\end{tikzcd}$$

What could χ_{nil} be? A pullback is a preimage. This mean that the subobject on the left has to consist exactly of those parts of 1 that χ_{nil} sends to $\top$. But since *nil* selects nothing, it follows that χ_{nil} cannot send anything at all to $\top$, on pain of contradiction. So, χ_{nil} has to *avoid* the "entirely true" part of Ω entirely. When it sends pieces of 1 on the left over to Ω on the right, it cannot send anything into the part of Ω that $\top$ touches.

At this point, it might be tempting to think that although χ_{nil} cannot send pieces of 1 to $\top$, nothing prevents it from send pieces of 1 to other parts of Ω. For instance, if Ω can discriminate partial truths, then perhaps χ_{nil} sends pieces of 1 to any of those parts in Ω that represent partial truth.

But this cannot be, because 1 is too course to discriminate anything except the entirely true and entirely false. So, χ_{nil} has no choice but to send pieces of 1 to the part of Ω that represents being "entirely false." In other words, χ_{nil} is the $\perp$ arrow:

$$\begin{tikzcd}
0 \arrow[r] \arrow[d, "nil"'] & 1 \arrow[d, "\top"] \arrow[l, phantom, "\text{pull back}"] \\
1 \arrow[r, "\chi_{nil} = \perp"'] & \Omega
\end{tikzcd}$$

Thus, the two morphisms from 1 to Ω pick out the two "global" elements of Ω. Even if there are many more grades and variations of partial truth encoded inside Ω, $\top : 1 \to \Omega$ picks out the top element (being entirely true), and $\perp$: $1 \to \Omega$ picks out the bottom element (being entirely false).

2.4.2. Conjunction

Intuitively, conjunction is a binary operator: it takes two formulas and then it tells us whether they are both true. Thus, the characteristic morphism for conjunction will be a morphism from $\Omega \times \Omega$ to Ω.

The truth table for conjunction is of course well known: both conjuncts have to be true, otherwise the conjunction is false. To characterize conjunction in a topos, we want to do the same thing: we want to select the subobject of $\Omega \times \Omega$ that has both components true. This can be done with the morphism $\langle \top, \top \rangle$:

$$\begin{tikzcd}
1 \arrow[d, "\langle \top, \top \rangle"'] \\
1
\end{tikzcd}$$

If we pull back from the subobject classifier, there will be a unique characteristic morphism that classifies that subobject:

$$\begin{tikzcd}
1 \arrow[r] \arrow[d, "\langle \top, \top \rangle"'] & 1 \arrow[d, "\top"] \arrow[dl, phantom, "\Leftarrow \text{pull back}"] \\
1 \arrow[r, "\chi_{\langle \top, \top \rangle}?"'] & \Omega
\end{tikzcd}$$

This $\chi_{\langle \top, \top \rangle}$ classifies the subobject that has both components true, so it will send the parts of $\Omega \times \Omega$ that are both entirely true to $\top$, just like the truth table in classical logic. Hence, $\chi_{\langle \top, \top \rangle}$ characterizes the conjunction operation:

$$\begin{tikzcd}
1 \arrow[r] \arrow[d, "\langle \top, \top \rangle"'] & 1 \arrow[d, "\top"] \arrow[dl, phantom, "\Leftarrow \text{pull back}"] \\
1 \arrow[r, "\chi_{\langle \top, \top \rangle} = \mathit{and}"'] & \Omega
\end{tikzcd}$$

2.4.3. Negation

Negation $\neg : \Omega \to \Omega$ intuitively corresponds to the complement. If we have a predicate $P(x)$, say the students in the class who are passing, then $\neg P(x)$ should pick out the complement: those remaining students who fall outside the extension of $P(x)$. In other words, we want to send the "false" part of Ω to the "true" part of Ω, as if to say "those who fail to satisfy the predicate are the ones we want to pick out."

To do that, we simply need to classify $\bot \colon 1 \to \Omega$. Since $\bot$ picks out the subobject of Ω that corresponds to "entirely false," its characteristic morphism will send that part to $\top$. So, start with $\bot$, i.e., the monomorphism that picks out the false part of Ω:

$$\begin{tikzcd}
1 \arrow[d, "\bot"'] \\
\Omega
\end{tikzcd}$$

If we pull back from the subobject classifier, there will be a unique characteristic morphism that classifies this subobject:

$$\begin{tikzcd}
0 \arrow[r] \arrow[d, "\bot"'] & 1 \arrow[d, "\top"] \arrow[dl, phantom, "\xleftarrow{\text{pull back}}"] \\
\Omega \arrow[r, "\chi_\bot ?"'] & \Omega
\end{tikzcd}$$

What could $\chi_\bot$ be? Since $\bot$ picks out the false part of Ω, $\chi_\bot$ will send that part of Ω to $\top$. Thus, $\chi_\bot$ will select the part of Ω that corresponds to false and make it true, just as $\neg P(x)$ takes those who fail to satisfy $P(x)$ and it then selects them. In other words, $\chi_\bot$ characterizes logical negation (complement):

$$\begin{tikzcd}
0 \arrow[r] \arrow[d, "\bot"'] & 1 \arrow[d, "\top"] \\
\Omega \arrow[r, "\chi_{nil} = \neg"'] & \Omega
\end{tikzcd}$$

Note that taking the complement in a topos is not in general a Boolean operation. It sends a truth-value representing "this part is in P" to the truth-value representing "this part is in no part of P." But that amounts to the largest subobject disjoint from P, and that may not be the Boolean complement. It may be the interior of the Boolean complement, or a sub-part of the Boolean complement. In general, $\neg$ is the Heyting complement, although it is Boolean in Boolean toposes.

The graph example from earlier offers an example of a non-Boolean negation. There, the extension of $\neg P$ is the graph consisting only of the point q. Note in particular that the edge e_2 is not part of the complement. This is due to the fact that e_2 begins inside the extension of P but then leaves P and ends up outside of P's extension. When we take the complement of P, it is therefore not part of the largest part of F that is disjoint from P.)

2.5. *Quantifiers*

TODO

I still think I should be doing this from the fibrational perspective. Quantifiers are way easier to understand that way. Can we get away with not discussing quantifiers at all?

2.6. *The Internal Language of the Topos*

TODO

3. Mereological Reasoning in Toposes

TODO

3.1. Parts and Wholes, Logically

TODO: define wholes in the internal language, and define the parthood relation in the internal language. Formulate some definitions like overlap.

4. Mereological Logics For Free

TODO: prove some theorems like reflexivity, antisymmetry, and transitivity. discuss supplementation? extensionality? atoms?

References

1. Goldblatt, R. *Topoi: The Categorical Analysis of Logic*, 2nd ed.; Elsevier: Amsterdam, 1984.
2. Mac Lane, S.; Moerdijk, I. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*; Springer: New York, 1994.
3. Borceux, F. *Handbook of Categorical Algebra 3: Categories of Sheaves*; Cambridge University Press: Cambridge, UK, 1994.