Category Theory & Functional Data Abstraction

Brandon Shapiro

Math 100b
A category $\mathbf{C}$ is a collection of objects with arrows (often called morphisms) pointing between them.

$\text{Hom}_\mathbf{C}(X, Y)$ is the set of morphisms in $\mathbf{C}$ from $X$ to $Y$.

If $f \in \text{Hom}_\mathbf{C}(X, Y)$ and $g \in \text{Hom}_\mathbf{C}(Y, Z)$, then there exists a morphism $f \circ g$ in $\text{Hom}_\mathbf{C}(X, Z)$ (composition is associative).

For every object $X$ in $\mathbf{C}$, there is an identity morphism $1_X \in \text{Hom}_\mathbf{C}(X, X)$ ($f \circ 1_X = f$ and $1_X \circ g = g$).
A category $\mathbf{C}$ is a collection of objects with arrows (often called morphisms) pointing between them

$\text{Hom}_C(X, Y)$ is the set of morphisms in $\mathbf{C}$ from $X$ to $Y$

If $f \in \text{Hom}_C(X, Y)$ and $g \in \text{Hom}_C(Y, Z)$, then there exists a morphism $f \circ g$ in $\text{Hom}_C(X, Z)$ (composition is associative)

For every object $X$ in $\mathbf{C}$, there is an identity morphism $1_X \in \text{Hom}_C(X, X)$ ($f \circ 1_X = f$ and $1_X \circ g = g$)
Examples

- **Set** is the category of all sets, with functions between sets as the morphisms.
- All groups also form a category, **Grp**, with group homomorphisms as its morphisms.
- **Ring** and **R-mod** for some ring $R$ can be formed with ring and module homomorphisms as morphisms.
- A subcategory of category $\mathbf{C}$ is a category with all of its objects and morphisms contained in $\mathbf{C}$.
- Finite sets and the functions between them form a subcategory of **Set**, and abelian groups are a subcategory of **Grp**. Fields form a subcategory of the category of commutative rings, which is itself a subcategory of **Ring**.
Examples

- **Set** is the category of all sets, with functions between sets as the morphisms.
- All groups also form a category, **Grp**, with group homomorphisms as its morphisms.
- **Ring** and **$R$-mod** for some ring $R$ can be formed with ring and module homomorphisms as morphisms.
- A subcategory of category $\mathbf{C}$ is a category with all of its objects and morphisms contained in $\mathbf{C}$.
- Finite sets and the functions between them form a subcategory of **Set**, and abelian groups are a subcategory of **Grp**. Fields form a subcategory of the category of commutative rings, which is itself a subcategory of **Ring**.
A functor is a structure preserving map between categories.

For categories $\mathbf{C}$ and $\mathbf{D}$, a covariant functor $\mathcal{F} : \mathbf{C} \to \mathbf{D}$ sends the objects of $\mathbf{C}$ to objects in $\mathbf{D}$, and sends the morphisms in $\mathbf{C}$ to morphisms in $\mathbf{D}$.

If $f \in \text{Hom}_\mathbf{C}(X, Y)$, $\mathcal{F}(f) \in \text{Hom}_\mathbf{D}(\mathcal{F}(X), \mathcal{F}(Y))$.

$\mathcal{F}(1_X) = 1_{\mathcal{F}(X)}$, $\mathcal{F}(f \circ g) = \mathcal{F}(f) \circ \mathcal{F}(g)$.
A functor is a structure preserving map between categories.

For categories $\mathbf{C}$ and $\mathbf{D}$, a covariant functor $F : \mathbf{C} \to \mathbf{D}$ sends the objects of $\mathbf{C}$ to objects in $\mathbf{D}$, and sends the morphisms in $\mathbf{C}$ to morphisms in $\mathbf{D}$.

- If $f \in \text{Hom}_\mathbf{C}(X, Y)$, $F(f) \in \text{Hom}_\mathbf{D}(F(X), F(Y))$.
- $F(1_X) = 1_{F(X)}$, $F(f \circ g) = F(f) \circ F(g)$. 

The identity functor from $\mathbf{C}$ to $\mathbf{C}$ sends every object and morphism in $\mathbf{C}$ to itself.

Let $\mathcal{F}$ be a map from $\mathbf{Grp}$ to $\mathbf{Set}$ sending groups and homomorphisms in $\mathbf{Grp}$ to themselves in $\mathbf{Set}$. $\mathcal{F}$ is a functor from $\mathbf{Grp}$ to $\mathbf{Set}$ called the ‘forgetful functor’

Similarly, forgetful functors exist from $\mathbf{Ring}$ and $\mathbf{R}$-mod to $\mathbf{Grp}$ and to $\mathbf{Set}$

A functor from a category to itself is called an endofunctor

The identity functor is an endofunctor
Examples

- The identity functor from $\mathbf{C}$ to $\mathbf{C}$ sends every object and morphism in $\mathbf{C}$ to itself.
- Let $\mathcal{F}$ be a map from $\mathbf{Grp}$ to $\mathbf{Set}$ sending groups and homomorphisms in $\mathbf{Grp}$ to themselves in $\mathbf{Set}$. $\mathcal{F}$ is a functor from $\mathbf{Grp}$ to $\mathbf{Set}$ called the ‘forgetful functor’
- Similarly, forgetful functors exist from $\mathbf{Ring}$ and $\mathbf{R}$-$\mathbf{mod}$ to $\mathbf{Grp}$ and to $\mathbf{Set}$
- A functor from a category to itself is called an endofunctor
- The identity functor is an endofunctor
Date Types

- In computer programming languages, a data type is a set of elements that can be represented by a computer (finitely in binary) in the same way.
- Two of the most common data types are \( \mathbb{Z} \) and \( \mathbb{R} \).
- Real-world computing has constraints on memory, etc.
- Mathematically, a data type can be treated just as a set.

Types of Data
Set has sets as objects and functions as morphisms

\[ \text{Maybe} : \text{Set} \rightarrow \text{Set} \]
\[ \text{Maybe}(A) = A \cup \{\text{Nothing}\} \]

Maybe lets us define ‘safe’ versions of partial functions

\[ f : \mathbb{R} \rightarrow \text{Maybe}(\mathbb{R}) \]
\[ f(0) = \text{Nothing} \]
\[ f(x) = 1/x \ (x \neq 0) \]
- **Set** has sets as objects and functions as morphisms

  \[ \text{Maybe} : \text{Set} \rightarrow \text{Set} \]

  \[ \text{Maybe}(A) = A \cup \{\text{Nothing}\} \]

- **Maybe** lets us define ‘safe’ versions of partial functions

  \[ f : \mathbb{R} \rightarrow \text{Maybe}(\mathbb{R}) \]

  \[ f(0) = \text{Nothing} \]

  \[ f(x) = \frac{1}{x} \ (x \neq 0) \]
Maybe is a functor from **Set** to **Set** (endofunctor)

Needs a mapping for the morphisms (functions)

\[
M\text{map} : \text{Hom}(A, B) \to \text{Hom}(Maybe(A), Maybe(B))
\]

\[
M\text{map}(f)(\text{Nothing}) = \text{Nothing}
\]

\[
M\text{map}(f)(x) = f(x) \ (x \neq \text{Nothing})
\]

\[
M\text{map}(1_A) = 1_{Maybe(A)}
\]

\[
M\text{map}(f \circ g) = M\text{map}(f) \circ M\text{map}(g)
\]
- `Maybe` is a functor from `Set` to `Set` (endofunctor)
- Needs a mapping for the morphisms (functions)

\[ M \text{map} : \text{Hom}(A, B) \rightarrow \text{Hom}(M \text{aybe}(A), M \text{aybe}(B)) \]

\[ M \text{map}(f)(\text{Nothing}) = \text{Nothing} \]
\[ M \text{map}(f)(x) = f(x) (x \neq \text{Nothing}) \]

- \[ M \text{map}(1_A) = 1_{M \text{aybe}(A)} \]
- \[ M \text{map}(f \circ g) = M \text{map}(f) \circ M \text{map}(g) \]
List sends a set $A$ to the set of ‘lists’ of elements in $A$

$$\text{List} : \text{Set} \to \text{Set}$$

$$\text{List}(A) = \{()\} \cup \{(x, x\text{list}) \mid x \in A, x\text{list} \in \text{List}(A)\}$$

() is called the empty list

$$(1, (2, (3, (4, ())))) \in \text{List}(\mathbb{Z})$$
$$(1/2, (\text{Nothing}, (1/4, ())))) \in \text{List}(\text{Maybe}(\mathbb{Q}))$$
$$(1, 2, 3, 4) \in \text{List}(\mathbb{Z})$$
\( \text{List} \) sends a set \( A \) to the set of ‘lists’ of elements in \( A \)

\[
\text{List} : \text{Set} \rightarrow \text{Set} \\
\text{List}(A) = \{()\} \cup \{(x, x\text{list}) | x \in A, x\text{list} \in \text{List}(A)\}
\]

() is called the empty list

\[(1, (2, (3, (4, ())))) \in \text{List}(\mathbb{Z})\]
\[(1/2, (\text{Nothing}, (1/4, ()))) \in \text{List}(\text{Maybe}(\mathbb{Q}))\]
\[(1, 2, 3, 4) \in \text{List}(\mathbb{Z})\]
\( \mathcal{L} \text{ist} \) sends a set \( A \) to the set of ‘lists’ of elements in \( A \)

\[
\mathcal{L} \text{ist} : \text{Set} \rightarrow \text{Set}
\]

\[\mathcal{L} \text{ist}(A) = \{()\} \cup \{(x, x\text{list}) | x \in A, x\text{list} \in \mathcal{L} \text{ist}(A)\}\]

- () is called the empty list

\[
(1, (2, (3, (4, ()))))) \in \mathcal{L} \text{ist}(\mathbb{Z})
\]

\[
(1/2, (\text{Nothing}, (1/4, ()))) \in \mathcal{L} \text{ist}(\text{Maybe}(\mathbb{Q}))
\]

\[
(1, 2, 3, 4) \in \mathcal{L} \text{ist}(\mathbb{Z})
\]
- $\mathcal{L}ist$ is an endofunctor on $\textbf{Set}$
- Needs a mapping for the morphisms (functions)

\[ \mathcal{L}\text{map} : \text{Hom}(A, B) \rightarrow \text{Hom}(\text{List}(A), \text{List}(B)) \]

\[ \mathcal{L}\text{map}(f)(()) = () \]

\[ \mathcal{L}\text{map}(f)((x, xlist)) = (f(x), \mathcal{L}\text{map}(f)(xlist)) \]

For $f(x) = x^2$, $\mathcal{L}\text{map}(f)((1, 2, 3, 4)) = (1, 4, 9, 16)$

- Clearly satisfies functor laws (identity and composition)
\( \mathcal{L} \text{List} \) is an endofunctor on \( \text{Set} \)

- Needs a mapping for the morphisms (functions)

\[
\mathcal{L}\text{map} : \text{Hom}(A, B) \to \text{Hom}(\mathcal{L}\text{List}(A), \mathcal{L}\text{List}(B))
\]

\[
\mathcal{L}\text{map}(f)(()) = ()
\]

\[
\mathcal{L}\text{map}(f)((x, x\text{list})) = (f(x), \mathcal{L}\text{map}(f)(x\text{list}))
\]

For \( f(x) = x^2 \), \( \mathcal{L}\text{map}(f)(((1, 2, 3, 4))) = (1, 4, 9, 16) \)

- Clearly satisfies functor laws (identity and composition)
- $\mathcal{L}ist$ is an endofunctor on $\textbf{Set}$
- Needs a mapping for the morphisms (functions)

\[
\mathcal{L} \text{map} : \text{Hom}(A, B) \rightarrow \text{Hom}(\mathcal{L}ist(A),\mathcal{L}ist(B)) \\
\mathcal{L} \text{map}(f)(()) = () \\
\mathcal{L} \text{map}(f)((x, xlist)) = (f(x), \mathcal{L} \text{map}(f)(xlist))
\]

For $f(x) = x^2$, $\mathcal{L} \text{map}(f)((1, 2, 3, 4)) = (1, 4, 9, 16)$

- Clearly satisfies functor laws (identity and composition)
- \( \mathcal{L} \text{List} \) is an endofunctor on \( \textbf{Set} \)
- Needs a mapping for the morphisms (functions)

\[
\mathcal{L}\text{map} : \text{Hom}(A, B) \to \text{Hom}(\mathcal{L}\text{List}(A), \mathcal{L}\text{List}(B))
\]

\[
\mathcal{L}\text{map}(f)(()) = ()
\]

\[
\mathcal{L}\text{map}(f)((x, x\text{list})) = (f(x), \mathcal{L}\text{map}(f)(x\text{list}))
\]

For \( f(x) = x^2 \), \( \mathcal{L}\text{map}(f)((1, 2, 3, 4)) = (1, 4, 9, 16) \)

- Clearly satisfies functor laws (identity and composition)
Applicative Functors

- What does an endofunctor on \textbf{Set} to do a set of functions?
- An applicative functor is a functor with a ‘splat’ function
  \[ \mathcal{F} \text{splat} : \mathcal{F}(\text{Hom}(A, B)) \to \text{Hom}(\mathcal{F}(A), \mathcal{F}(B)) \]
- \( \mathcal{F} \text{splat} \) can also be defined as a binary function
  \[ \mathcal{F} \text{splat} : \mathcal{F}(\text{Hom}(A, B)) \times \mathcal{F}(A) \to \mathcal{F}(B) \]
- There are rules applicative functors must follow
Applicative Functors

- What does an endofunctor on \textbf{Set} to do a set of functions?
- An applicative functor is a functor with a ‘splat’ function

\[ \mathcal{F}\text{\_splat} : \mathcal{F}(\text{Hom}(A, B)) \rightarrow \text{Hom}(\mathcal{F}(A), \mathcal{F}(B)) \]

- \(\mathcal{F}\text{\_splat}\) can also be defined as a binary function

\[ \mathcal{F}\text{\_splat} : \mathcal{F}(\text{Hom}(A, B)) \times \mathcal{F}(A) \rightarrow \mathcal{F}(B) \]

- There are rules applicative functors must follow
Applicative Functors

- What does an endofunctor on $\textbf{Set}$ to do a set of functions?
- An applicative functor is a functor with a ‘splat’ function

\[ F\text{ splat} : F(Hom(A, B)) \rightarrow Hom(F(A), F(B)) \]

- $F\text{ splat}$ can also be defined as a binary function

\[ F\text{ splat} : F(Hom(A, B)) \times F(A) \rightarrow F(B) \]

- There are rules applicative functors must follow
Applicative Functors

- What does an endofunctor on \textbf{Set} to do a set of functions?
- An applicative functor is a functor with a ‘splat’ function

\[ \mathcal{F}\text{splat} : \mathcal{F}(\text{Hom}(A, B)) \to \text{Hom}(\mathcal{F}(A), \mathcal{F}(B)) \]

- \( \mathcal{F}\text{splat} \) can also be defined as a binary function

\[ \mathcal{F}\text{splat} : \mathcal{F}(\text{Hom}(A, B)) \times \mathcal{F}(A) \to \mathcal{F}(B) \]

- There are rules applicative functors must follow
Applicative Functors

- *Maybe* is an applicative functor

\[ \text{Msplat} : \text{Maybe}(\text{Hom}(A, B)) \times \text{Maybe}(A) \to \text{Maybe}(B) \]
\[ \text{Msplat}(\text{Nothing})(\_\_) = \text{Msplat}(\_\_)(\text{Nothing}) = \text{Nothing} \]
\[ \text{Msplat}(f)(x) = f(x) \]

- *List* is an applicative functor

\[ \text{Lsplat} : \text{List}(\text{Hom}(A, B)) \times \text{List}(A) \to \text{List}(B) \]
\[ \text{Lsplat}_1((\_\_))(\_\_) = \text{Lsplat}_1(\_\_)((\_\_)) = () \]
\[ \text{Lsplat}_1((f, \text{flist}))(\langle x, \text{xlist} \rangle) = (f(x), \text{Lsplat}_1(\text{flist})(\text{xlist})) \]

- Could *List* be an applicative functor in any other ways?
Maybe is an applicative functor

\[ M \text{splat} : M\text{aybe}(\text{Hom}(A, B)) \times M\text{aybe}(A) \rightarrow M\text{aybe}(B) \]
\[ M \text{splat}(\text{Nothing})(\_ \_ ) = M \text{splat}(\_ \_ )(\text{Nothing}) = \text{Nothing} \]
\[ M \text{splat}(f)(x) = f(x) \]

List is an applicative functor

\[ L\text{splat} : L\text{ist}(\text{Hom}(A, B)) \times L\text{ist}(A) \rightarrow L\text{ist}(B) \]
\[ L\text{splat}_1((\_))(\_ \_ ) = L\text{splat}_1(\_ \_ )(\_ \_ ) = () \]
\[ L\text{splat}_1((f, f\text{liss}))(\text{x, x\text{list}})) = (f(x), L\text{splat}_1(f\text{liss})(x\text{list})) \]

Could List be an applicative functor in any other ways?
Applicative Functors

- Maybe is an applicative functor

\[ Msplat : \text{Maybe}(\text{Hom}(A, B)) \times \text{Maybe}(A) \rightarrow \text{Maybe}(B) \]
\[ Msplat(\text{Nothing})(\_ \_ ) = Msplat(\_ \_ )(\text{Nothing}) = \text{Nothing} \]
\[ Msplat(f)(x) = f(x) \]

- List is an applicative functor

\[ Lsplat : \text{List}(\text{Hom}(A, B)) \times \text{List}(A) \rightarrow \text{List}(B) \]
\[ Lsplat_1(((\_ \_ ))(\_ \_ )) = Lsplat_1(\_ \_ )(() \_ ) = () \]
\[ Lsplat_1(\left((f, \text{flist}))((x, xlist)) = (f(x), Lsplat_1(\text{flist})(xlist)) \]

- Could List be an applicative functor in any other ways?
Sources

- Abstract Algebra by Dummit and Foote
- Lectures by and conversations with Kenny Foner

Images

- https://bartoszmilewski.files.wordpress.com/2014/10/img_1330.jpg
- http://shuklan.com/haskell/L12_files/category.png
- https://lh3.googleusercontent.com/proxy/W-kz6vWx9ntZrS2FCduApSQ0E-YsddspOrfnWyKP2J-49Uu8_5ahu-IOEfHLmT7w2IZMvQ_vhDGxCkqHIMo1C_0VCrCFeSzfvtW4PjD