Элементы теории категорий в функциональном программировании

Андрей Ляшин, FProg

SPb 26.05.2016

Категория

Категорией C называется следующая совокупность определений:

- Класс объектов Ob_c (|C|);
- Каждой паре объектов A,B из Ob_c ставится в соответствие множество морфизмов (или стрелок) Hom(A,B);
- Для каждой пары морфизмов $f \in Hom(A,B)$, $g \in Hom(B,C)$ определен морфизм, называемый композицией $f \circ g \in Hom(A,C)$;
- Для каждого объекта A существует тождественный морфизм $id_A \in Hom(A,A)$. Данный факт, в частности, позволяет обойтись без отдельного класса объектов, считая что они заданы своими тождественными морфизмами;
- Операция композиции ассоциативна $f \circ (g \circ h) = (f \circ g) \circ h$;
- Тождественная стрелка действует тривиально $f \circ id_A = id_B \circ f = f$, $f \in Hom(A,B)$.

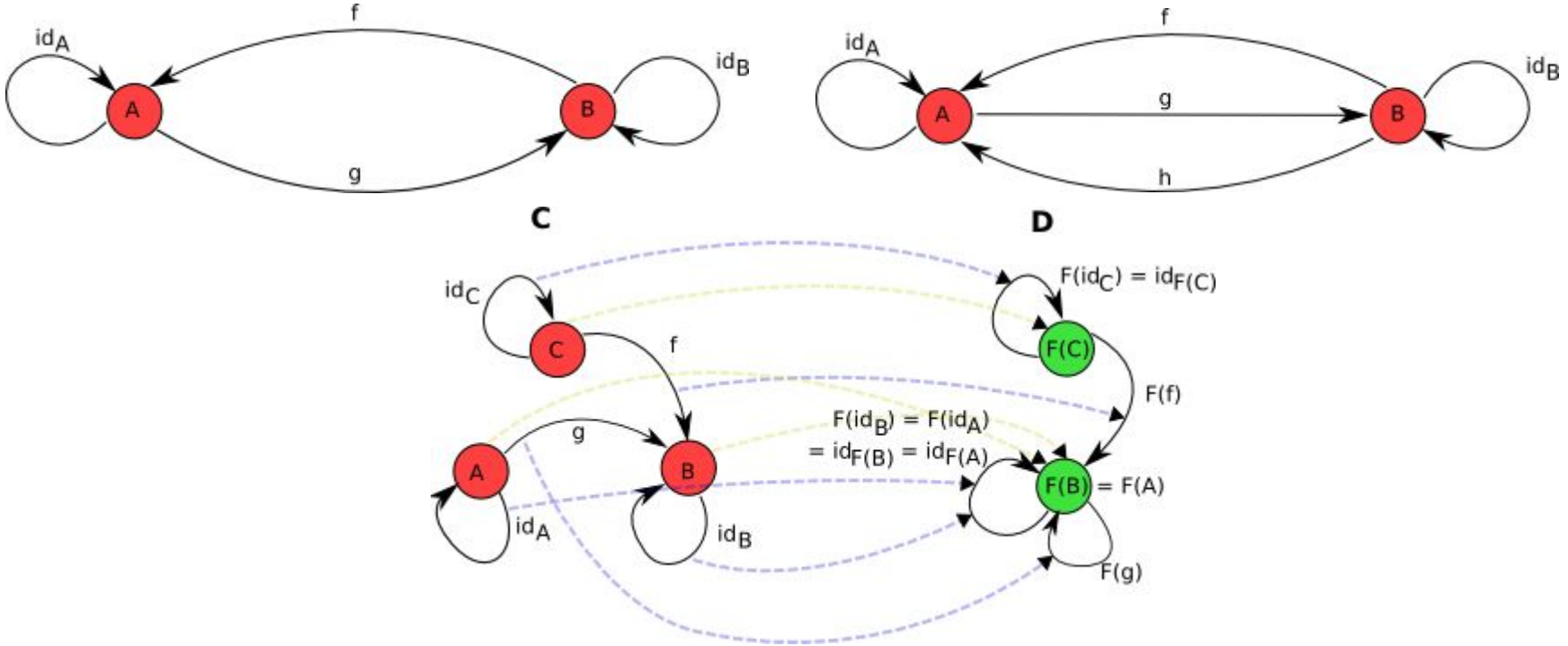
Примеры категорий

Следующие категории могут служить иллюстрациями определения:

- Категория множеств ***Set*** **:** объектами служат множества, морфизмами - отображения множеств;
- Категория групп ***Grp*** **:** объектами служат группы, морфизмами - гомоморфизмы групп;
- Категория типов языка Haskell ***Hask*** **:** объектами являются типы данных, морфизмами - функции между ними (функции, преобразующие экземпляры одного типа в экземпляры другого).

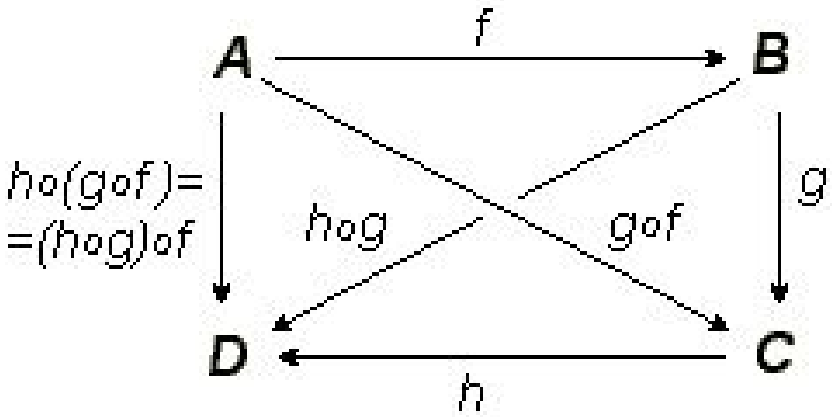
Наглядное представление категорий

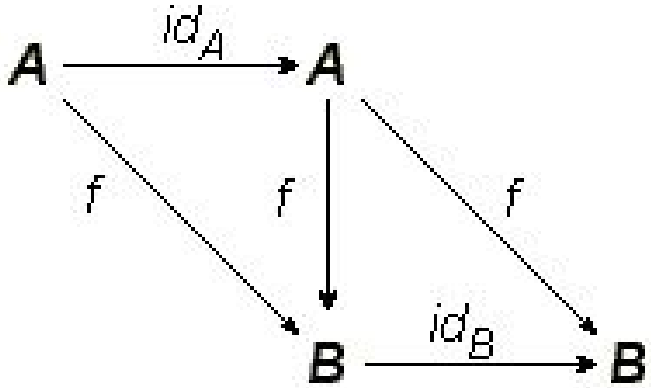
Категории частно иллюстрируют диаграммами (графами) следующего вида:



Коммутативные диаграммы

Примеры утверждений в ТК иллюстрируют т.н. коммутативными диаграммами. Коммутативность означает равенство объектов, полученных различными путями по направлениям стрелок.



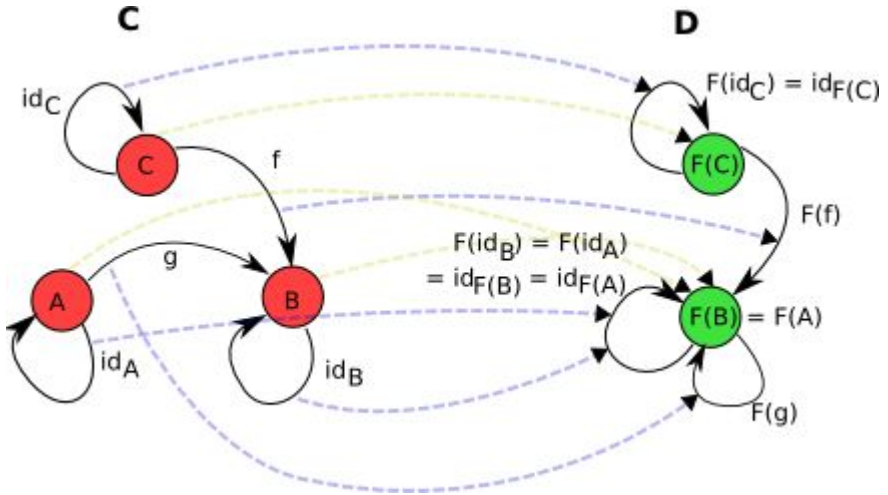


Морфизмы категорий. Функторы.

Для отображения из одной категории в другую используют функторы. Поскольку в категории изначально рассматривают два типа компонент - объекты и стрелки, функтор действует на элементы обеих.

А именно (ковариантный) функтор F:C→D имеет в себе два отображения

- $F(id_A) = id_{F(A)}$;
- $F(g) \circ F(f) = F(g \circ f)$.



Примеры

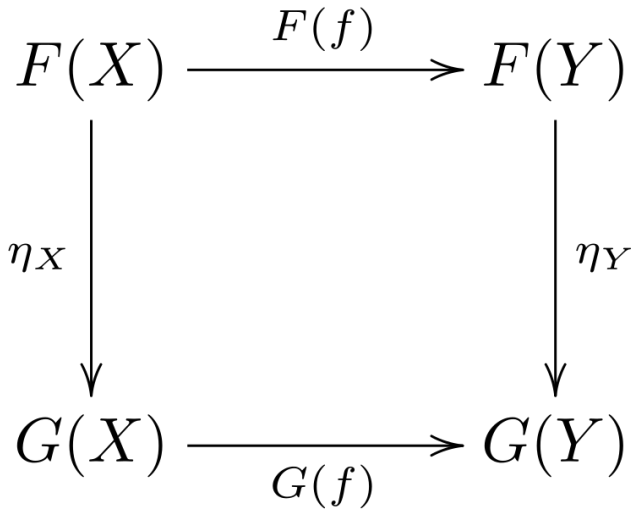
- Функтор вложения в подкатегорию;
- Забывающий функтор, например, ***Grp*** → ***Set*** сохраняет группы, как “просто” множества, а гомоморфизмы как “просто” отображения множеств;
- PowerSet функтор P : ***Set*** → ***Set***, ставящий каждому множеству множество его подмножеств $P(S) = 2^S$, P (f) = map f.
- Эндофунктор Maybe : ***Hask*** → ***Hask***.
  - t - объект в ***Hask*** (тип), Maybe (t) = Maybe t;
  - f: a → b - морфизм типов, mf = Maybe f : Maybe a → Maybe b: mf (Just x) = Just (f x), mf (Nothing) = Nothing;
  - Maybe (id) = id;
  - Maybe (f◦g) (Just x) = Just (f◦g x) = Maybe (f) ◦ Maybe (g) (Just x)
  - Maybe (f◦g) Nothing = Nothing = Maybe (f) ◦ Maybe (g) (Nothing).

Примеры (продолжение)

- Эндофунктор list : ***Hask*** → ***Hask***.
  - list (a) = list a;
  - f: a -> b - морфизм типов, lf = list f : list a → list b: lf = map f;
  - lf (id) = map id_a = $id_{list\ a}$;
  - lf (f◦g) = map (f◦g) = map f ◦ map g.
- Бифункторы, из категории произведения категорий, например, гомофунктор Hom : C^{Op}×C -> ***Set*** для локально малой категории C:
  - B - объект в C, Hom(A,-) B = Hom (A,B);
  - f - морфизм в C из X в Y, Hom(A,-) f : Hom (A,X) → Hom (A,Y), Hom(A,f) (h_{AX}) = f◦h_{AX};
  - Hom(A,id_X) = $id_{Hom(A,X)}$;
  - Hom(A, f◦g) h = f◦g◦h = (Hom(A, f) ◦ Hom(A, g)) h.
- Функторы образуют категорию, являясь объектами, а морфизмами - естественные преобразования. Можно считать, что ЕП появились как попытка сделать из функторов кактгорию.

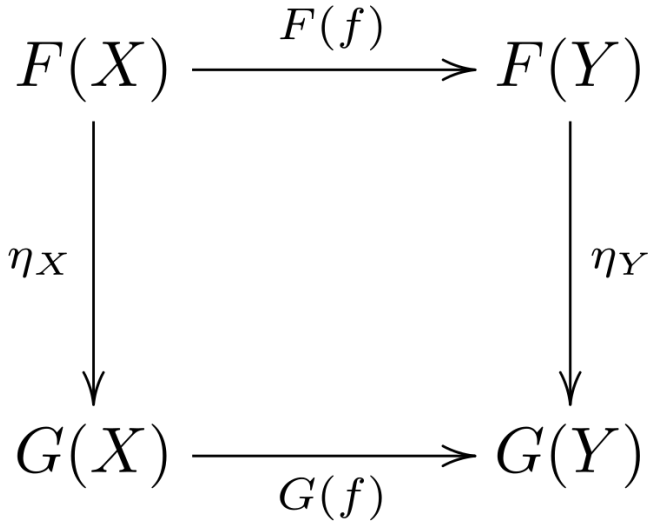
Естественное преобразование

Связь между функторами выражает понятие естественного преобразования: для двух функторов F,G : C → D естественным преобразованием называется совокупность отображений η_X: F(X) → G(X) в категории D, делающих следующую диаграмму коммутативной:



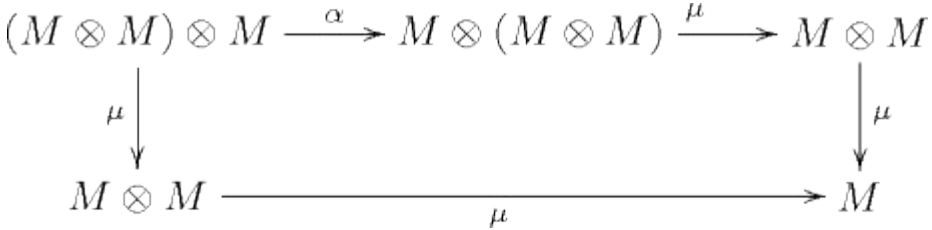
Примеры

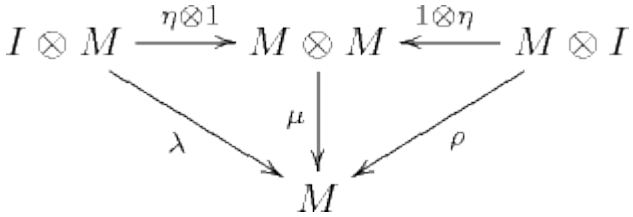
- Естественные преобразования являются морфизмами в категории функторов.
- $id_F : F \to F$, $F: C \to D$. $(id_F)_A = id_{F(A)}$.
- Синглетон sing : $Id_{\mathbf{Set}} \to P$ (: ***Set*** → ***Set***). $sing_X : Id(X) \to P(X)$, $sing_X\ x = \{x\}$, $x \in X$.



Моноид

- Моноид как алгебраическая структура является более общим случаем, чем группа, и более частным, чем полугруппа, а именно:
- Совокупность элементов M с ассоциативной бинарной операцией (умножение) μ:M×M→M и единицей η:M, действующей тривиально относительно умножения;
- В ТК моноид может быть представлен в терминах моноидальной категории с объектом M и морфизмами α, μ, η, λ, ϱ.

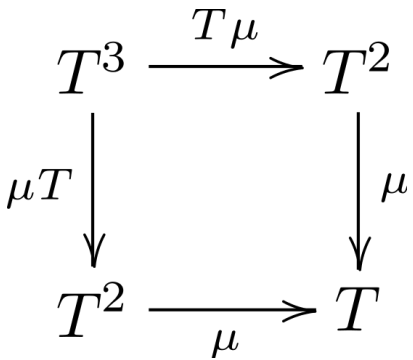


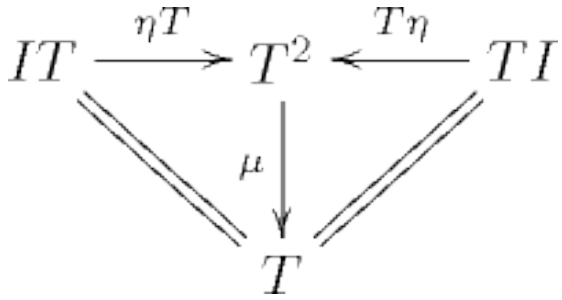


Монада

Эквивалентом моноида в категории функторов является монада - совокупность трех объектов (T, η, μ):

- Эндофунктора T : K → K;
- Естественного преобразования η : id_K → T;
- Естественного преобразования μ : T^2 → T.





Монадные правила

- $\mu \circ T\mu = \mu \circ \mu T$ или $\mu_X \circ T(\mu_X) = \mu_X \circ \mu_{T(X)}$;
- $\mu \circ T\eta = \mu \circ \eta T = id_T$ или $\mu_X \circ T(\eta_X) = \mu_X \circ \eta_{T(X)} = id_T$;
- Данные правила являются ничем другим, как свойствами моноида в моноидальной категории эндофункторов - ассоциативности и левой/правой единицы;
- В языке Haskell монада определяется как функтор (*→ *) операцией *fmap* : (a → b) → f a → f b, *fmap* id = id, *fmap* (f . g) = *fmap* f . *fmap* g, с двумя дополнительными преобразованиями *unit* и (*join*/*bind*). Преобразование *unit* является эквивалентом η, а *join* - μ. Преобразование *bind* (>>=) введено для удобства записи последовательности монадных вычислений. Вот эти определения
- *unit* : X → M X; *join*: M M X → M X; *bind* : M X → (X → M Y) → M Y;
- *join* mmx := mmx >>= id; mx >>= f := *join* (*fmap* f mx);
- *join* . (*fmap join*) = *join* . *join*;
- *join* . (*fmap unit*) = *join* . *unit* = *id*; (три моноидных правила)
- *join* . *fmap* (*fmap* f) = (*fmap* f) . *join*;
- (*fmap* f) . *unit* = *unit* . f (два правила из определения ЕП)

Примеры

Монада PowerSet (P, $unit_P$, $join_P$):

- $P : S \mapsto 2^S$, $P(f)(W) = \{f(x), x \in W\}$;
- $unit_P$: $Id \mapsto P$; $unit_{PX} = sing_X$;
- $join_P : P^2 \mapsto P$; $join_{PX} : P^2(X) \to P(X)$; $join_{PX}(W) = \cup W$
- $\{1,2\} \mapsto \{\{1\},\{2\},\{1,2\}\} \mapsto \{\{\{1\}\},\{\{2\}\},\{\{1,2\}\},\{\{1\},\{2\}\},\{\{2\},\{1,2\}\},\{\{1\},\{1,2\}\},\{\{1\},\{2\},\{1,2\}\} \mapsto \{\{1\},\{2\},\{1,2\},\{1\},\{2\},\{2\},\{1,2\},\{1\},\{1,2\},\{1\},\{2\},\{1,2\}\} \mapsto \{\{1\},\{2\},\{1,2\}\}$.
- *join . (fmap join) = join . join.* $(: P^3 \to P), \{A,B\} \mapsto \{\cup A, \cup B\} \mapsto (\cup A) \cup (\cup B) \leftarrow \cup (A \cup B) \leftarrow A \cup B \leftarrow \{A,B\}$
- *join . (fmap unit) = join . unit = id.* $(: P \to P)$, $\{a,b\} \mapsto \{\{a,b\}\} \mapsto \{a,b\} \leftarrow \{\{a\},\{b\}\} \leftarrow \{a,b\}$
- *join . fmap (fmap f) = (fmap f) . join.* $(: P^2 \to P)$, $\{A,B\} \mapsto \{f \twoheadrightarrow A, f \twoheadrightarrow B\} \mapsto (f \twoheadrightarrow A) \cup (f \twoheadrightarrow B) \leftarrow (f \to A \cup B) \leftarrow A \cup B \leftarrow \{A,B\}$
- *(fmap f) . unit = unit . f.* $(: Id \to P)$, $a \mapsto \{a\} \mapsto \{f\ a\} \leftarrow f\ a \leftarrow a$.

Монадные правила и do нотация

Монадные правила создают возможность использовать do нотацию с ожидаемым результатом, а именно:

`return x >>= f ‘=’ f x`	`do { v <- return x; f v } ‘=’ do { f x }`
`m >>= return ‘=’ m`	`do { v <- m; return v } ‘=’ do { m }`

`(m >>= f) >>= g ‘=’ m >>= (\x -> f x >>= g)`	`do { y <- do { x <- m; f x };`
	`  g y }`
	`‘=’`
	`do { x <- m;`
	`  y <- f x;`
	`  g y }`

Монады в языке Haskell

- Монады и функторы представлены в языке Haskell как классы над унарными конструкторами типов *→ *
  - `class Functor (f :: * -> *) where`
    `fmap :: (a -> b) -> f a -> f b`
  - `class Functor m => Monad m where`
    `return :: a -> m a`
    `(>>=) :: m a -> (a -> m b) -> m b`
- Примерами монад выступают Maybe, list, Either *, ST * и т.д.
- Монады выступают в качестве специального вычислительного контекста;
- Кроме того важной конструкцией являются монадные преобразования (monad transformers):
  - t: `(* -> *) -> * -> *` ; `lift :: m a -> t m a`
  - `lift . return = return`
  - `lift (m `bind` k) = (lift m) `bind` (lift . k)`
- С их помощью выражаются такие сущности как option MT, reader/writer MT, state MT, exception MT, и другие.

Категория Kleisli

- Пусть дана монада <T, μ, η> и категория C. Категорией Клейсли над C, связанной с монадой T, называется категория K:
  - Ob(K) = Ob(C);
  - $Hom_K(X,Y) = Hom_C(X,TY)$;
  - $g \circ_K f = \mu \circ Tg \circ f$; $x \mapsto_f T\ y \mapsto_{Tg} T\ T\ z \mapsto_{\mu} T\ z$;
  - $id_{KX} = \eta_X$.
- ```
  class Kleisli m where
    idK  :: a -> m a
    (*>) :: (a -> m b) -> (b -> m c) -> (a -> m c)
  ```
  - ```
    (+>) :: Kleisli m => (a -> m b) -> (b -> c) -> (a -> m c)
    f +> g = f *> (g >> idK)
    ```

Категория Клейсли порождает монадные композиции как обычные for free.

Категория Клейсли для монады Maybe

- ```
  instance Kleisli Maybe where
    idK     = Just
    f *> g = \a -> case f a of
                     Nothing -> Nothing
                     Just b  -> g b
  ```
- С помощью стрелок Клейсли мы можем элегантно композировать частично определнные функции.
  ```

Понятия гомотопической теории типов

Различные интерпретации теоретико-типовых понятий

Интуиционистская теория типов	Логика	Теория множеств	Теория гомотопий
тип A	высказывание A	множество A	пространство A
$a : A$	доказательство высказывания A	a — элемент множества A	a — точка пространства A
зависимый тип $B(x)$	предикат $B(x)$	семейство множеств	расслоение B_x
$b(x) : B(x)$	условное доказательство	семейство элементов	сечение[en]
$\mathbf{0}, \mathbf{1}$	$\bot, \top$	$\varnothing, \{\varnothing\}$	$\varnothing, *$
$A + B$	$A \vee B$	$A \uplus B$ (дизъюнктное объединение)	$A \oplus B$ (копроизведение)
$A \times B$	$A \wedge B$	$A \times B$ (декартово произведение)	$A \times B$ (произведение пространств)
$A \to B$	$A \Rightarrow B$	множество функций $\{f \mid f : A \to B\}$	функциональное пространство B^A
$\mathbf{\Sigma}_{x:A} B(x)$	$\exists_{x:A} B(x)$	$\biguplus_{x \in A} B(x)$ (дизъюнктное объединение)	тотальное пространство
$\mathbf{\Pi}_{x:A} B(x)$	$\forall_{x:A} B(x)$	$\prod_{x \in A} B(x)$ (декартово произведение)	пространство сечений
Id_A	равенство ($=$)	$\{(x, x) \mid x \in A\}$	пространство путей $A^{\mathbb{I}}$

Литература

- http://learnyouahaskell.com/a-fistful-of-monads
- https://en.wikipedia.org/wiki/Monad_(category_theory)
- https://en.wikipedia.org/wiki/Monad_(functional_programming)
- https://ncatlab.org/nlab/show/monad+%28in+computer+science%29
- https://anton-k.github.io/ru-haskell-book/book/6.html
- Mac Lane, Saunders (1998). *Categories for the Working Mathematician*. Graduate Texts in Mathematics **5** (2nd ed.). Springer-Verlag. ISBN 0-387-98403-8. Zbl 0906.18001
- https://en.wikibooks.org/wiki/Haskell/Category_theory
- https://en.wikipedia.org/wiki/Kleisli_category
- https://homotopytypetheory.org/
- https://en.wikipedia.org/wiki/Homotopy_type_theory