Week Twelve

Siva Sundar, EE23B151

December 23, 2024

16^{nd} December

- A **database** is an organised system of interlocking tables. **Database Scheme** are categories $\mathcal{C}$, data itself is given by a *'set-valued'* functor $\mathcal{C} \longrightarrow$**Set**, and databases can be mapped to each other via functors $\mathcal{C} \longrightarrow \mathcal{D}$.
- Category theory formalizes **data migration** between databases using **adjoint functors**.
- A category $\mathcal{C}$ consists of four constituents:
  - ⋆ a collection $\mathrm{Ob}(\mathcal{C})$, whose elements are **objects**.
  - ⋆ $\forall c, d \in \mathrm{Ob}(\mathcal{C})$, we specify the **hom-set** $\mathcal{C}(c, d)$, whose elements are called **morphisms**.
  - ⋆ $\forall c \in \mathrm{Ob}(\mathcal{C})$, we can specify the **identity morphism** on c: $id_c \in \mathcal{C}(c, c)$.
  - ⋆ $\forall c, d, e \in \mathrm{Ob}(\mathcal{C})$ and morphisms $f \in \mathcal{C}(c, d)$ and $g \in \mathcal{C}(d, e)$, we can define the **composite morphism** $f \circ g \in \mathcal{C}(c, c)$.

  It should also satisfy two properties:

  - ⋆ *Unitality*: $id_c \circ f = f \circ id_c = f$.
  - ⋆ *associativity*: $(f \circ g) \circ h = f \circ (g \circ h)$.
- For any graph $G = (V, A, s, t)$, we can define a **free category Free**(G) whose objects are *vertices* V, and morphisms from c to d are the *paths* from c to d.
- A category with one object is called **monoid**. (see page 83)
- A finite graph with path equations is called a *finite presentation* and this category is called *finitely-presented category.* (see difference between free category and commutative square in page 84)
- A preorder is a category where every two parallel arrows are the same (ie, between two points, there is *at-most* one morphism, Page 85-86). Similarly, any category can be converted to a preorder by destroying the distinction between any two parrelel morphisms.

19^{th} December

- The cateogory of finite sets, is called **FinSet**. There are many other categories as well. (See page 87)
- $f : A \longrightarrow B$ is an **isomorphism** if there exists another morphism $g : B \longrightarrow A$ satisfying $f \circ g = id_A$ and $g \circ f = id_B$. g is said to be inverse of f and A and B are said to be **isomorphic** objects.

- **Functor** is a mapping between categories. It maps objects as well as morphisms from one category to another. It obeys the *structure-preserving* rule: $F(f \circ g) = F(f) \circ F(g)$. (See page 91)
- Let $\mathcal{C}$ be a finitely-presented category. A **$\mathcal{C}$-instance** is a functor $I : \mathcal{C} \longrightarrow \mathbf{Set}$. (It is a state of the database, "at an instant in time", see ex 3.45)

  The takeway is that: 'a database schema is a category, and an instance on that schema (the data itself), is a set-valued functor. The constraints (biz rules, etc) are ensured by the structure preserving rule.'
- **Natural transformation** (say α) is a mapping between two functors (say $F : \mathcal{C} \longrightarrow \mathcal{D}$ and $G : \mathcal{C} \longrightarrow \mathcal{D}$), denoted as $\alpha : F \Longrightarrow G$:
  - ⋆ $\forall c \in \mathcal{C}$, *$c$-component of α* is the morphism $\alpha_c : F(c) \longrightarrow G(c)$ in $\mathcal{D}$.
  - ⋆ **Naturality condition:** $\forall f : c \longrightarrow d$ in $\mathcal{C}$, $F(f) \circ \alpha_d = \alpha_c \circ G(f)$.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"'] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"'] & G(d) \end{tikzcd}$$

Figure 1: Diagram commutes $\Longleftrightarrow$ naturality condition

$\alpha : F \longrightarrow G$ is said to be **natural isomorphism** if each of its component α_c is an **isomorphism in** $\mathcal{D}$.

- **Functor category:** Let $\mathcal{C}$ and $\mathcal{D}$ be categories. We define the **category of functors** $\mathcal{D}^{\mathcal{C}}$ with *objects* as functors $F : \mathcal{C} \longrightarrow \mathcal{D}$ and with *morphisms* $\mathcal{D}^{\mathcal{C}}(F, G)$ as the natural transformations $\alpha : F \longrightarrow G$. (See ex 3.72)

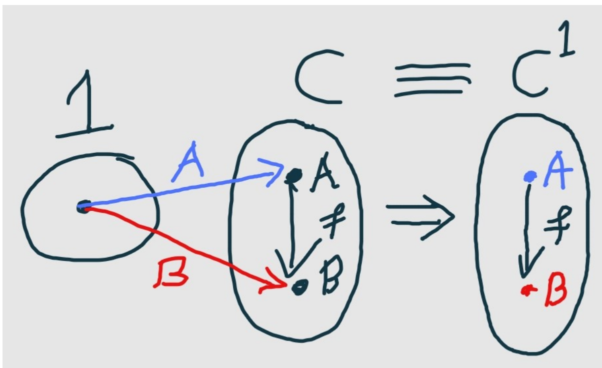

Figure 2: The category $\mathcal{C}^1$ is equivalent to $\mathcal{C}$.

The category of preorders is equivalent to category of **Bool**-categories. (Page 97)

22^{nd} December

- Let $\mathcal{C}$ be a *database scheme* and $I, J : \mathcal{C} \longrightarrow \mathbf{Set}$ be *database instances*. An **instance homomorphism** between I and J is a *natural transformation* $\alpha : I \longrightarrow J$ and these are included in the *functor category* $\mathcal{C} - \mathbf{Inst} := \mathbf{Set}^{\mathcal{C}}$.

- The objects in the functor category **Gr-Inst** are graphs and the morphisms init are called *graph homomorphisms* which must *respect source and target*. (See page 98)
- Let $\mathcal{C}$ and $\mathcal{D}$ be categories and let $F : \mathcal{C} \longrightarrow \mathcal{D}$ be a functor. For any set-valued functor $I : \mathcal{D} \longrightarrow \mathbf{Set}$, the composite functor $F \circ I$ is called **pullback of I along F**.
- Let $L : \mathcal{C} \longrightarrow \mathcal{D}$ and $R : \mathcal{D} \longrightarrow \mathcal{C}$ be functors. L is the **left adjoint** to R and R is the **right adjoint** to L if there exist an isomorphism (see page 102):

  $$\forall c \in \mathcal{C} \ \ \& \ \ d \in \mathcal{D}, \qquad \alpha_{c,d} : \mathcal{C}(c, R(d)) \xrightarrow{\cong} \mathcal{C}(L(c), d)$$

  In set theory, given a set B, we have an adjunction called **currying** B: (see page 103)

  $$\mathbf{Set}(A \times B, C) \cong \mathbf{Set}(A, C^B)$$

- Given $F : \mathcal{C} \longrightarrow \mathcal{D}$, the **data migration functor** Δ_F turns $\mathcal{D}$-instances into $\mathcal{C}$-instances.

$$\begin{tikzcd}
\mathcal{C}\textbf{-Inst} \arrow[rr, bend left, "\Sigma_F"] \arrow[rr, bend right, "\Pi_F"'] \arrow[rr, phantom, shift left=3, "\Rightarrow"] \arrow[rr, phantom, shift right=3, "\Leftarrow"] & & \mathcal{D}\textbf{-Inst} \arrow[ll, "\Delta_F" description]
\end{tikzcd}$$

Figure 3: Σ_F and Π_F are the Left and Right (Pushforward) adjoint functors resp. (See page 105)

23^{th} December

- Made a **C++ program** for the database example given in Page 105.
- **Terminal Objects** is the most basic limit.

  Let $\mathcal{C}$ be a category. Then object Z in $\mathcal{C}$ is a *terminal object* if $\forall C \in \mathcal{C}$, there is a **unique morphism** $! : C \longrightarrow Z$.

  Because of the *uniqueness* condition, we say these terminal objects have a **universal property**. ie, All terminal objects in a category $\mathcal{C}$ are isomorphic.

- **Products**: Let $\mathcal{C}$ be a category, and let $X, Y \in \mathcal{C}$. A *product* of X and Y is an object $X \times Y$, together with morphisms $p_X : X \times Y \longrightarrow X$ and $p_Y : X \times Y \longrightarrow Y$ such that, $\forall C \in \mathcal{C}$ with morphisms $f : C \longrightarrow X$ and $g : C \longrightarrow Y$, there exists a unique morphism $< f, g >: C \longrightarrow X \times Y$, for which the below diagram commutes.

$$\begin{tikzcd}
& C \arrow[dl, "f"'] \arrow[dr, "g"] \arrow[dd, dotted, "\langle f,g \rangle"] & \\
X & & Y \\
& X \times Y \arrow[ul, "p_X"] \arrow[ur, "p_Y"'] &
\end{tikzcd}$$

Whatever be f, g, f', g', they all contribute to a *unique morphism* $< f, g >=< f', g >=< f, g' >=< f', g' >$. Hence, we say products also have a **universal property**.

- **Category of Cones** over X and Y, **Cone**(X, Y) has objects $\mathcal{X} \xleftarrow{f} C \xrightarrow{g} Y$ and morphisms from $\mathcal{X} \xleftarrow{f} C \xrightarrow{g} Y$to $\mathcal{X} \xleftarrow{f'} C' \xrightarrow{g'} Y$ is $a : C \longrightarrow C'$ such that the below diagram commutes.

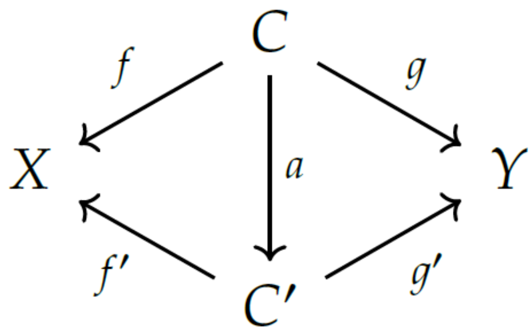

Figure 4: Caption

- A **diagram** in $\mathcal{C}$ is a functor $D : \mathcal{J} \longrightarrow \mathcal{C}$, where $\mathcal{J}$ is the **indexing category** of the diagram D.

  A *cone*(C, c_*) over D consists of an object $C \in \mathcal{C}$ and $\forall j \in \mathcal{J}$, a morphism $c_j : C \longrightarrow D(j)$. These cones should also satisfy the property (cone should **commute**):

$$\forall f : j \longrightarrow k \text{ in } \mathcal{J}, \quad c_k = D(f) \circ c_j$$

- The **Limit** of category D is the *terminal object* in the category **Cone**(D). In the cone $\lim(D) = (C, c_*)$, C is the **limit object** and the map c_j $\forall j \in \mathcal{J}$ is the j^{th} **projection map**.

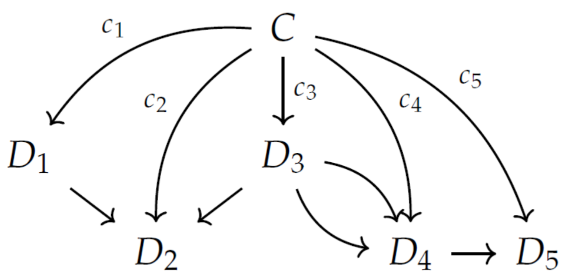

Figure 5: The diagram should **commute** for a valid cone.

  Also, any two parallel paths that start from limit object C are considered the same.

- Let $\mathcal{J}$ be a category presented by finite graph (V, A, s, t) together with some equations, and let $D : \mathcal{J} \longrightarrow$ **Set**. The set

$$\lim_{\mathcal{J}} D := \{(d_1, \ldots, d_n) | d_i \in D(v_i) \forall 1 \leqslant i \leqslant n \;\; \& \;\; \forall a : v_i \longrightarrow v_j \in A, D(a) \circ d_i = d_j\}$$

  together with the projection maps $p_i : (\lim_{\mathcal{J}} D) \longrightarrow D(v_i)$ given by $p_i(d1, \ldots, d_n) := d_i$ is a limit of D.

The condition $D(a) \circ d_i = d_j$ allows us to use limits to select data that satisfies certain equations or constraints. ie, we can express queries in terms of limits. Example 3.99 explains this statement and what pullbacks mean. **Pullback** is the limit of the *cospan* graph.

- **Colimits** are the opposite of a **limit**.

  Given a category $\mathcal{C}$, **cocone** in $\mathcal{C}$ is a cone in $\mathcal{C}^{op}$. Given a diagram $D : \mathcal{J} \longrightarrow \mathcal{C}$, the colimit is the limit of the functor $D^{op} : \mathcal{J}^{op} \longrightarrow \mathcal{C}^{op}$.