Category Theory

Databases: Categories, functors, and universal construction

Jakub Jurczak i Szymon Kuliński

Politechnika Krakowska

February 13, 2020

Outline

What is a database?

A database is a system of interlocking tables. Data becomes information when it is stored in a given formation. That is, the numbers and letters don't mean anything until they are organized, often into a system of interlocking tables. An organized system of interlocking tables is called a database. Below we can see an example:

What is a database

Employee	FName	WorksIn	Mngr
1	Alan	101	2
2	Ruth	101	2
3	Kris	102	3

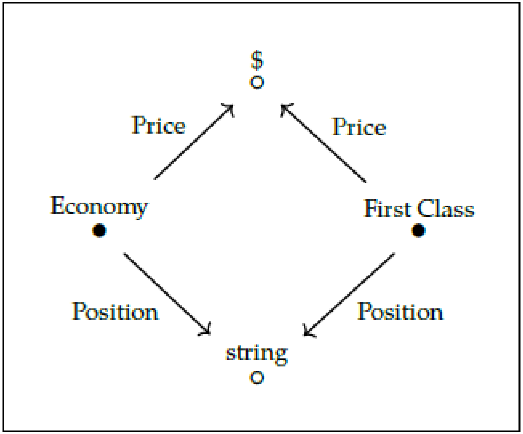
Department	DName	Secr
101	Sales	1
102	IT	3

Communication between databases

Databases are designed to store information about something. But different people or organizations might view the same sort of thing in different ways. For example, one bank stores its financial records according to European standards and another does so according to Japanese standards. If these two banks merge into one, they will need to be able to share their data despite differences in the shape of their database schemas. **Category theory** provides a mathematical approach for translating between these different organizational forms. That is, it formalizes a sort of automated reorganizing process called data migration, which takes data that fits snugly in one schema and moves it into another

Schema

$A :=$



Schema

$$\boxed{\begin{tikzcd} \overset{\$}{\circ} \\ \underset{\bullet}{\text{Airline Seat}} \arrow[u, "\text{Price}"'] \arrow[d, "\text{Position}"'] \\ \overset{\text{string}}{\circ} \end{tikzcd}} =: B$$

Category definition

A category C consists of four pieces of data - objects, morphisms, identities, and a composition rule - satisfying two properties.

Category definition

(i) one specifies a collection $Ob(C)$, elements of which are called objects.
(ii) for every two objects c,d one specifies a set $C(c, d)$, elements of which are called **morphisms** from c to d
(iii) for every object c $\in Ob(C)$, one specifies a morphism $id_c \in C(c, c)$, called **identity morphism** on c
(iv) for every three objects $c, d, e \in Ob(C)$ and morphisms $f \in C(c, d)$ and $g \in C(d, e)$ one specifies a morphism $f \fatsemi g \in C(c, e)$ **called the composite of f and g**
These constituents are required to satisfy two conditions:
(a) unitality: for any morphism $f : c \to d$, composing with the identites at c or d does nothing : $id_c \fatsemi f = f$ and $f \fatsemi id_d = f$
(b) associativity: for any three morphisms $f : c_0 \to c_1$, $g : c_1 \to c_2$, and $h : c_2 \to c_3$, the following are equal: $(f \fatsemi g) \fatsemi h = f \fatsemi (g \fatsemi h)$. We write this composite simply as $f \fatsemi g \fatsemi h$.

Free category

For any graph $G = (V, A, s, t)$ we can define a category $\mathbf{Free}(G)$, called the *free category on* G, whose objects are the vertices V and whose morphisms from c to d are the paths from c to d. The identity morphism on an object c is simply the trivial path at c. Composition is given by concatenation of paths.

Free category graph

$$\mathbf{2} := \mathbf{Free}\left(\boxed{\begin{tikzcd} \overset{v_1}{\bullet} \arrow[r, "f_1"] & \overset{v_2}{\bullet} \end{tikzcd}}\right)$$

Presenting categories via path equations

$$\text{Free\_square} := \boxed{\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "h"] \\ C \arrow[r, "i"'] & D \end{tikzcd} \quad \textit{no equations}}$$

Preorders and free categories

Kategorie typu preordery

Important categories in mathematics

The **category of sets,**denoted **Set**, is defined as follows.
(i) $Ob(\textbf{Set})$ is the collection of all sets (ii) if S and T are sets, then $\textbf{Set}(S, T) = \{f : S \to T \mid \text{f is a function}\}$
(iii) For each set S, the identity morphism is the function $id_s : S \to S$ given by $id_s := s$ for each $s \in S$
(iv) Given $f : S \to T$ and $g : T \to U$, their composite is the function $f \fatsemi g : S \to U$ given by $(f \fatsemi g)(s) := g(f(s))$

Functors, natural transformations, and databases

We want to explain what the data in a database is, as a way to introduce functors. To do so, we begin by noticing that sets and functions—the objects and morphisms in the category Set—can be captured by particularly simple databases.

Sets and functions as databases

Planet of Sol
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune

Prime number
2
3
5
7
11
13
17
;

Flying pig

The first observation is that any set can be understood as a table with only one column: the ID column.

Sets and functions as databases

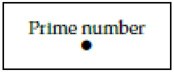
The above databases have schemas consisting of just one vertex:

Sets and functions as databases

Planet of Sol

Sets and functions as databases

Sets and functions as databases



Sets and functions as databases

A function $f : A \to$ can almost be depicted as a two-column table. Except it is unclear whether the elements of the right-hand column exhaust all of B. What if there are rock-and-roll instruments out there that none of the Beatles played?

Beatle	**Played**
George	Lead guitar
John(Lemon)	Rhytm guitar
Paul	Bass guitar
Ringo	Drums

Sets and functions as databases

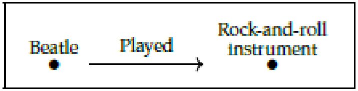
So a function $f : A \to B$ requires two tables, one for A and its f column, and one for B:

Beatle	**Played**
George	Lead guitar
John(Lemon)	Rhytm guitar
Paul	Bass guitar
Ringo	Drums

Rock-and-roll instrument
Bass guitar
Drums
Keyboard
Lead guitar
Rhythm guitar

Sets and functions as databases

Thus the database schema for any function is just a labeled version of:



Functors

A functor is a mapping between categories. It sends objects to objects and morphisms to morphisms, all while preserving indentities and composition. Here is the formal definition:
Let C and D be categories. To specify a functor from C to D, denoted $F : C \to D$:
(i) for every object $c \in Ob(C)$, one specifies an object $F(c) \in Ob(D)$
(ii) for every morphism $f : c_1 \to c_2$ one specifies a morphism $F(f) : F(c_1) \to F(c_2) inD$.
The above constituens must satisfy two properties:
(a) for every object $c \in Ob(C)$ we have $F(id_c) = id_{F(c)}$
(b) for every three objects $c_1, c_2, c_3 \in Ob(C)$ and two morphisms $f \in C(c_1, c_2), g \in C(c_2, c_3)$, the equation $F(f \mathbin{⨟} g) = F(f) \mathbin{⨟} F(g)$ holds in D

Data Migrations

We have talked about howset-valued functors on a schema can be understood as filling that schema with data. But there are also functors between schemas. When the two sorts of functors are composed, data is migrated. This is the simplest form of data migration:

Data Migrations

To begin, we will migrate data between the graph-indexing schema Gr and the loop schema, which we call DDS, shown below.

Data migrations

We begin by writing down a sample instance I :**DDS** $\rightarrow$**Set** on this schema:

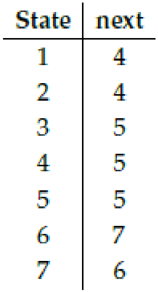
Data migrations

$$\mathbf{Gr} := \boxed{\begin{tikzcd} \overset{\text{Arrow}}{\bullet} \arrow[r, shift left, "\text{source}"] \arrow[r, shift right, "\text{target}"'] & \overset{\text{Vertex}}{\bullet} \end{tikzcd} \quad \textit{no equations}}$$

Data migrations

$$\mathbf{DDS} := \boxed{\begin{tikzcd} \overset{\text{State}}{\bullet} \arrow[loop below, "\text{next}"] \\ \textit{no equations} \end{tikzcd}}$$

Data migrations

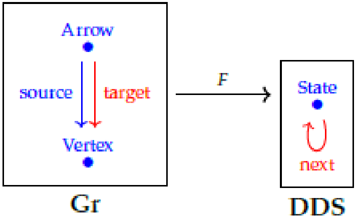


State	next
1	4
2	4
3	5
4	5
5	5
6	7
7	6

Data migrations

We call the schema DDS to stand for discrete dynamical system. Indeed, we may think of the data in the DDS-instance of Eq. (3.65) as listing the states and movements of a deterministic machine: at every point in time the machine is in one of the listed states, and given the machine in one of the states, in the next instant it moves to a uniquely determined next state. Our goal is to migrate the data in from the tabla to data on Gr; this will give us the data of a graph and so allow us to visualise our machine. We will use a functor connecting these schemas in order to move data between them.We will use a specific functor F :**Gr**$\to$**DDS** to migrate data in a way that makes sense to us, the authors. Here we draw F, using colors to hopefully aid understanding:

Data migrations



The functor F sends both objects of **Gr** to the ‘State’ object of **DDS** (as it must). On morphisms, it sends the ‘source’ morphism to the identity morphism on ‘State’, and the ‘target’ morphism to the morphism ‘next’.

End

Dziękujemy za uwagę