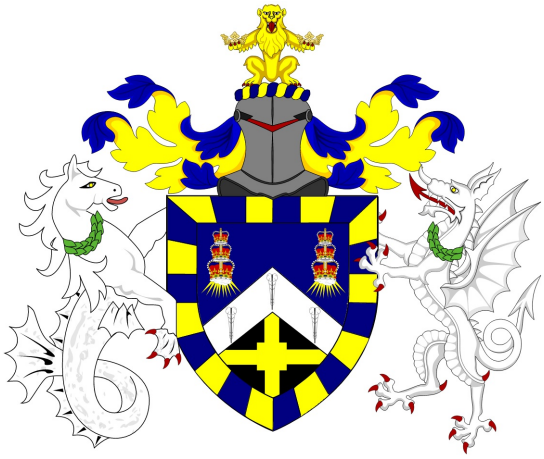
Applications of Category Theory in Data Analytics



Naman Singh, September 2020

Supervised by Dr. Behrang Noohi

School of Mathematical Sciences
Queen Mary, University of London

This dissertation is submitted for the degree of
Master's of Science in Data Analytics.

Declaration of original work

This declaration is made on 11th September 2020.

I, Naman Singh, hereby declare that the work in this thesis is my original work. I have not copied from any other students' work, work of mine submitted elsewhere, or from any other sources except where due reference or acknowledgement is made explicitly in the text, nor has any part been written for me by another person. Referenced text has been flagged by using italic fonts, using quotation marks ". . . ", or explicitly mentioning the source in the text.

Abstract

This project is a record of my engagement with [CM13] and [Spi12]. With this project, I intend to emphasize the virtue of applying Category Theory to subjects in Data Analytics. I explain with added detail, proofs, examples and background, the frameworks and their consequences put forward for clustering schemes and databases by the two papers. Conjectures regarding exciveness, fullness and faithfulness of clustering algorithms are mentioned with justification. A detailed proof of the equivalence between **Sch** and **Cat** is given and data migration functors are represented in an uncomplicated manner.

Preface

Category theory occupies a dominant role in modern mathematics. It provides a mathematical approach to formalising structure and systems of multiple structures. Since its conception, it has proven to be a great success in not just the mathematical community, but also amongst computer scientists and mathematical physicists. It is unparalleled in its capacity to bridge the gap between all kinds of mathematical subjects. Therefore, it encourages collaboration between scholars of different persuasions. This paper aims to prove the virtue of extending the application of category theory to data analytics and encourage more research of this kind.

Data analytics is notorious for its disconnect between its theoretical and practical parts. Scholars have attempted to bridge this gap using category theory (See [CM13], [Spi12], [CS13], [WSSS15], [Mil18]). The categorical paradigm has been successfully introduced to data-related topics such as cluster analysis, database theory, machine learning and programming language theory. The initial goal of this project was to give a comprehensive account of all of the applications mentioned above. However, the applications are too elaborate to be described within the time frame allotted to this project. Only the applications of category theory in cluster analysis and database theory are described.

The paper is suitable for readers with an understanding of and experience with basic graduate-level mathematics. Metric spaces, equivalence relationships,

graphs, meets and joins are some of the concepts readers are expected to be familiar with. Also, an understanding of foundational category theory concepts, such as a category, functor, natural transformations, and adjoints. The readers who are new category theory should refer to [Lei14] for an introduction. Any necessary background for the ideas discussed is provided in this paper.

The papers, [CM13] and [Spi12], are discussed and are made suitable for a more general audience. This is done by giving detailed explanations with examples wherever necessary, adding more detail to proofs, and giving background whenever necessary. Conjectures are also added in a few places.

In the first chapter, a framework for understanding clustering algorithms, as described by [CM13], is provided. The framework is then used to create classification of clustering schemes, and tackle issues related to linkage functions and density-sensitivity of clustering schemes. The framework is then used to provide a uniqueness and existence theorem of the same flavour as [Kle03].

In the second chapter, a model for databases is developed using the graphical concepts as in [Spi12]. The model is then equated with rigour to the category **Cat**, and hence, a connect between category theory and database theory is established. Data migration, data types and data filtering are discussed afterwords.

I aim to provide as much detail I can while presenting the ideas put forth in these two papers. Examples, proofs, explanations are added wherever possible to make these ideas accessible to readers with less experience with mathematical subjects.

Contents

Chapter 1

CLUSTER ANALYSIS

1.1 Introduction

Cluster analysis is an essential part of exploratory data analytics. Clustering techniques arrange data into either groups of individual points or into a hierarchy of groups. The arrangements expose underlying patterns in the data that help researchers achieve conclusions which are often impossible to confirm otherwise. The applications of this science are found in fields such as astronomy, engineering, image analysis, social media technology, bio-informatics, psychology, marketing, etc.

Despite it's heavy use and extensive literature, very little is known about the mathematical foundation of clustering. Due to the field's vast practical (and monetary) potential, its scholars often focus their energy on the application of these techniques rather than the theory behind them. The disregard for any theoretically motivated research has rendered experts of the subject unequipped to deal with some of the limitations of these methods. A mathematically sound framework is desirable and so is provided by [CM13] using category theory.

The framework is used to give insight into standard and hierarchical clustering algorithms and classifications of these algorithms are described. Issues relating to linkage functions and density-sensitivity of clustering schemes are addressed. A theorem of a similar kind to Kleinberg [Kle03] is provided. This chapter is a record of my engagement with [CM13]. First, the context of the paper is laid out.

1.1.1 Standard and Hierarchical Clustering

This is summart of classification is inspired by [JD88]. Clustering is a special type of classification applied on a data-set equipped with some notion of distance or a metric, i.e a metric space. As suggested by Lance and Williams, 1967, [LW67] the different types of classifications can be arranged into a binary tree where each leaf represent a genus of classification type. (See figure 1)

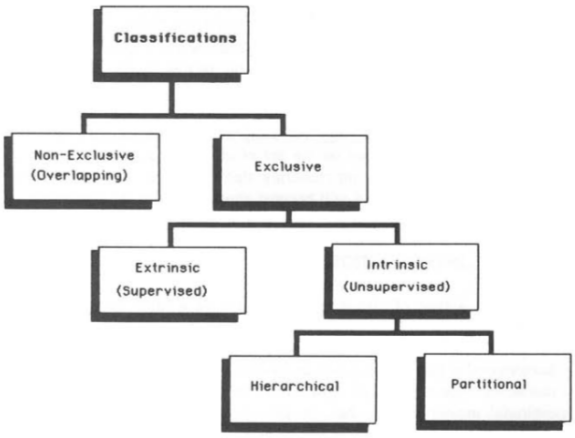


Figure 1.1: Tree of classification types, image credit [JD88]

The nodes in the figure are described as follows: [JD88]

• **Exclusive and Non-exclusive** - Exclusive classification assigns each data point to exactly one class. It creates a total partition of the data set. An example of exclusive classification is grouping people by age, height, sex, etc. Non-exclusive classification does the opposite allows a data point to exist in more than one class. An example of non-exclusive is grouping of people by their hobbies, illnesses, etc.

• **Intrinsic and Extrinsic** Intrinsic classification creates a partition using only the input metric space. It is unsupervised in the sense that it doesn't use any pre-existing knowledge of the class labels of data points. Extrinsic classification uses the input metric space and also pre-existing knowledge about class labels.

• The focus of [CM13] is solely on exclusive intrinsic classifications and the term clustering is used to refer to them. Exclusive and intrinsic classification are further divided into the following:

1. **Partitional Classification** (Or **Standard**) gives exactly one partition of the input metric space. In [CM13], it is referred to as standard clustering. An example of standard clustering is k-means clustering.

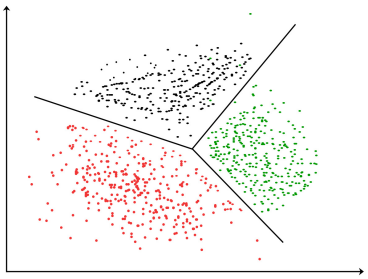


Figure 1.2: K-Means Clustering, Image from [Pri20]

2. **Hierarchical Classification** gives a sequence of nested partitions of the input. An agglomerative algorithm is a hierarchical classification which starts with a discrete partition. Using the metric, the algorithm determines how elements of the metric space are grouped together into bigger clusters with each iteration until a single cluster with all elements is reached. Each partition in the sequence created is a refinement of the partition that come after it. This procedure results in what is called a dendrogram, shown in the figure below. Aggloremative hierarchical procedures are explored through the framework put forward in the paper.

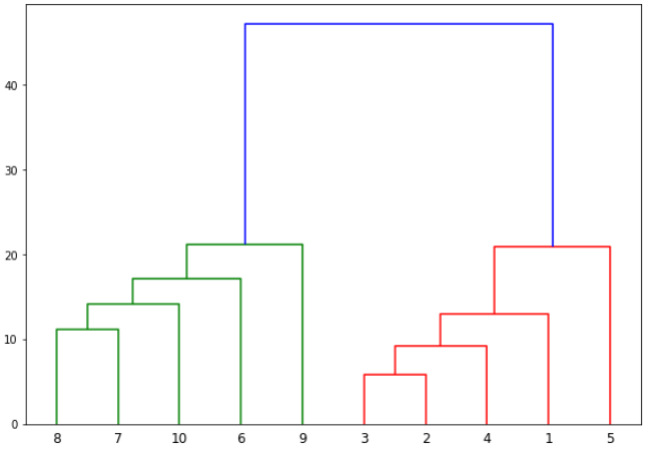


Figure 1.3: Agglomerative Hierarchical Classifiction, Image credit [Mal]

In fact, there are two types of dendrograms, threshold and proximity. Proximity dendrograms contain information about the proximity of the underlying clusters they represent whereas threshold dendrograms don't. Proximity dendrograms are almost always preferred. Therefore, the model for hierarchical clustering that is put forward have finite metric spaces as their input and proximity dendrograms as their output. As we exclusively deal with proximity dendrograms in this paper, they will simply be referred to as dendrograms for brevity.

Formally, dendrograms can be described as follows.

Definition 1.1.1 Let X be a finite set and $\mathcal{P}(X)$ be the set of all possible partitions of X. A dendrogram over X is a pair (X, θ_X) where $\theta_X : [0, \infty) \to \mathcal{P}(X)$ such that:

1. $\theta_X(0)$ is the discrete partition of X, i.e. it only contains single element blocks.
2. If $r \leqslant s$, then $\theta_X(r)$ is a refinement of $\theta_X(s)$, i.e. block in $\theta_X(r)$ is a subset of some block in $\theta_X(s)$.
3. For every $r \in [0, \infty)$, there exists an $\epsilon > 0$, such that $\theta_X(r) = \theta_X(r')$ for every $r' \in [r, r + \epsilon]$
4. There exists a $t_0 > 0$, such that $\theta_X(t)$ is the single block partition for every $t \geqslant t_0$.

The purpose of assigning dendrograms to the input data through hierachical clustering algorithms is to formalise trends of 'proximities' or 'similarities' in the data. The larger the value r for which two data points belong in the same cluster of $\theta_X(r)$, the more different they are. Clusters that occur naturally in the data are detected without any à priori knowledge or training data.

Hierarchical and partitional algorithms both are equally important in their domains. Hierarchical techniques are popular in the fields such as sociology, biology and behavioural science. Partitional algorithms are applied when the hierarchy is less relevant such as engineering and data compression and representation.

Unlike partitional algorithms, there is no need to specify that number of clusters that is to be imposed on the data in hierarchical algorithms. Instead, they offer multiple levels of clusterings, with each level operating at a different level of abstraction. It enables researchers to see data points get merged into clusters at successive levels of similarity. For example, the application of HC

algorithms to DNA sequences of animals reveals phyogenetic trees of animal evolution.

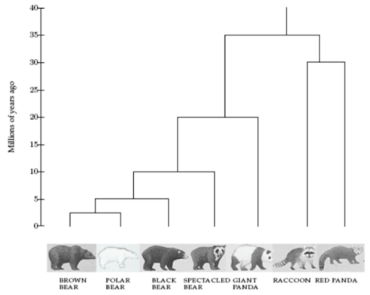


Figure 1.4: Phoyogenetic tree, Image credit [Kil]

1.1.2 Linkage Functions

There are three methods described in [CM13] through which hierarchical clustering can be performed: single link, complete link and average link. These methods have been modified so as to allow conglomeration of more than two clusters in one step. As a result, the resulting dendrogram need not be a binary tree (as opposed to classical linkage algorithms). For each method, we start with a discrete partition and start sequentially combining clusters into larger clusters, until a single block partition is achieved. At each step, the two clusters separated by the shortest distance are combined. A linkage function is the metric used to determine distances between clusters is the variable in each method.

Let (X, d_X) be the input metric space. For single linkage, the distance between two clusters A, B is given by

$$\min_{a \in A, b \in B} d_X(a, b)$$

For complete-linkage

$$\max_{a \in A, b \in B} d_X(a, b)$$

For average-linkage

$$\frac{\sum_{a \in A} \sum_{b \in B} d_X(a, b)}{|A| \cdot |B|}$$

Agglomerative algorithms such as these are described as recursive procedures

in [CM13]. They are presented below with some corrections in notation, though the general idea is the same. Let (X, d_X) be a finite metric space, $R \geqslant 0$ and $l : \mathcal{P}(X) \times \mathcal{P}(X) \to \mathbb{R}_+$ a linkage function. Define $\sim_{l,R}$ to be an equivalence relation on $\mathcal{P}(X)$, given by $B \sim_{l,R} B'$ if and only if there exists a sequence of subsets of X, $B = B_1, ..., B_n = B'$ such that $l(B_i, B_{i+1}) \leqslant R$ for all $i = 1, ..., n-1$. Start with partition $\Theta_1 = \{\{x_1\}, ..., \{x_n\}\}$ the discrete partition of X and $r_1 = 0$ and for $i \geqslant 2$

$$r_i := \min\{l(B, B') : B, B' \in \Theta_i,\ B \neq B'\}$$

$$\Theta_{i+1} := \{\cup[\alpha] : [\alpha] \in \Theta_i / \sim_{l,r_{i+1}}\}$$

where $\alpha \in \Theta_i / \sim_{l,r_{i+1}}$ is an equivalence class and the union $\cup[\alpha]$ is the union of all elements in $[\alpha]$. The associated dendrogram is given by $\theta^l : [0, \infty) \to \mathcal{P}(X)$, $r \mapsto \theta^l(r) := \Theta_{i(r)}$, where $i(r) = \max\{i | r_i \leqslant r\}$.

The union of clusters of an equivalence class under $\sim_{l,r_i}$ determines a cluster in the succeeding partition. An equivalence class itself is not a cluster in the succeeding partition. This is a mistake in the (CM). The other mistakes were starting with $\Theta_1 = \{x_1, ..., x_n\}$ and the indexing of r_i. r_1 is set to 0 in the beginning just so that θ^l is well defined. It plays no roll in the clustering.

Remark 1.1.2 Note that the dendrogram need not be a binary tree in this form of clustering, as conglomeration of more than two clusters is allowed.

Example 1.1.3 Let X a rectangular pyramid. Label its vertices $x_1, ..., x_5$ such the distances are given by the proximity matrix

$$D := \begin{bmatrix} 0 & 3 & 4 & 5 & 6 \\ 3 & 0 & 5 & 4 & 6 \\ 4 & 5 & 0 & 3 & 6 \\ 5 & 4 & 3 & 0 & 6 \\ 6 & 6 & 6 & 6 & 0 \end{bmatrix}$$

The sequences Θ_i and r_i using single-linkage are given by

	Θ_i	r_{i+1}	$\Theta_i / \sim_{l,r_{i+1}}$
1	$\{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\}\}$	3	$\{\{\{x_1\}, \{x_2\}\}, \{\{x_3\}, \{x_4\}\}, \{\{x_5\}\}\}$
2	$\{\{x_1, x_2\}, \{x_3, x_4\}, \{x_5\}\}$	4	$\{\{\{x_1, x_2\}, \{x_3, x_4\}\}, \{x_5\}\}$
3	$\{\{x_1, x_2, x_3, x_4\}, \{x_5\}\}$	6	$\{\{\{x_1, x_2, x_3, x_4\}, \{x_5\}\}\}$
4	$\{x_1, x_2, x_3, x_4, x_5\}$		

The dendrogram is given by the function

$$\theta(r) = \begin{cases} \Theta_1 & \text{if } r \in [0,3) \\ \Theta_2 & \text{if } r \in [3,4) \\ \Theta_3 & \text{if } r \in [4,6) \\ \Theta_4 & \text{if } r \in [6,\infty) \end{cases}$$

Critique of single, complete and average linkage: Many critiques of these techniques have been offered (see [CM10] [LW67]). Average and complete linkage are preferred over single linkage because they are sensitive to density in a way that single linkage is famously not. In a non technical sense, they produce more compact clusters and the empirical evidence tells us that results obtained through AL and CL are conceptually coherent. Single linkage is notorious for what is called the chaining effect. However, it possess convenient theoretical advantages which AL and CL don't. These critiques won't be discussed formally since they are beyond the scope of this paper. We do however address them with the framework under consideration.

1.1.3 Kleinberg's Theorem

In [Kle03], Kleinberg states three properties which are desirable in a partitional clustering algorithm and proves that it is impossible for one to have all three. The three properties are stated as follows: Let

1. **Scale Invariance**: for all $\lambda > 0$, we have $\mathfrak{C}(X, d_X) = \mathfrak{C}(X, \lambda \cdot d_X)$.
2. **Richness**: Given any partition $P \in \mathcal{P}(X)$ of a finite set X, there exists a metric d_X such that $\mathfrak{C}(X, d_X) = P$.
3. **Consistency**: Let $\mathfrak{C}(X, d_X) = P = \{B_1, ..., B_n\}$ and d'_X a metric on X, such that
   - for all $x, x' \in B_k$, we have $d'_X(x, x') \leqslant d_X(x, x')$.
   - for all $x \in B_j$ and $x' \in B'_k$, $(k \neq j)$, we have $d_X(x, x') \leqslant d'_X(x, x')$.

   Then $\mathfrak{C}(X, d'_X) = P$.

Following this theme, in [CM10] a similar theorem was put forward in the context of hierarchical methods which stated an existence and uniqueness result. Three desirable properties which were different and slightly more relaxed than Kleinberg's were spelled out. In [CM13], the existence and uniqueness theorem is

put in a categorical framework of clustering algorithms. Through the framework, classifications of clustering algorithms are put forward. Equivalence of two important properties, excisiveness and representability is proven.

The intuition for the framework is inspired by the functorial mapping of a topological space X to the set of its connected components $F(X)$. Here the connected components are equivalence classes under the equivalence relation $\sim_{\text{Path}}$, which is defined as follows: For every $x, x' \in X$, we write $x \sim_{\text{Path}} x'$ iff there exists a continuous map $\phi : [0,1] \to X$ such that $\phi(0) = x$ and $\phi(1) = x'$, implicating there exists a continuous path between x and x'. We call the mapping $X \mapsto F(X)$ functorial since for every continuous map between topological spaces $f : X \to Y$, there is a map of sets $F(f) : F(X) \to F(Y)$. This formulation is a highly analogous to clustering procedures. Functoriality, being a key observation, is transferred to the domain of finite metric spaces which are the inputs of clustering schemes as defined in [CM13]. The main point of this paper to show that clustering *can* be functorial.

Remark 1.1.4 When a mapping is said to be functorial, it simply means that the mapping can be expressed as functor from the input space to the output space. Note that in order for functoriality to be established formally, the input and output spaces also need to be expressed as categories. The concept of functoriality has been very useful in not just mathematical subjects, but also in mathematical physics, and other parts of theoretical computer science. [CM13] sets the path for more applications of category theory to be made in data analytics.

1.2 The Categorical Perspective

A partitional clustering procedure is defined to be a functor from a category of finite metric spaces to a category of partitions of finite sets. Similary, hierarchical clustering is defined to be a functor from a category of finite spaces to a category of hierarchies of partitions of finite sets. To establish functoriality, the input and output categories need to be defined.

1.2.1 Input Categories

We describe three categories whose objects are finite metric spaces. The variable is the type of morphisms between metric spaces. The types of morphisms and their associated categories are described below.

1. **Metric maps:** For finite metric spaces (X, d_X) and (d_Y), a metric map $f : (X, d_X) \to (Y, d_Y)$ satisfies the following condition: for all $x, x' \in X$, we have

$$d_Y(f(x), f(x')) \leqslant d_X(x, x')$$

It is easy to see that the composition of metric maps and identity maps are also metric maps. Hence, we can construct a category $\mathcal{M}^{gen}$ of finite metric spaces whose morphisms are metric maps. (Here, 'gen' is short for general)

2. **Injective metric maps:** Since composition of injective maps is injective and the composition of metric maps is a metric map, the composition of injective metric maps must be an injective metric map. Hence, we can create a smaller category $\mathcal{M}^{inj}$.

3. **Isometries:** An isometry between metric spaces $f : (X, d_X) \to (Y, d_Y)$ is a bijective map $X \to Y$ such that

$$d_Y(f(x), f(x')) = d_X(x, x')$$

for all $x, x' \in X$. Clearly, the composition of isometries is also an isometry. Therefore, we construct another category of finite metric spaces $\mathcal{M}^{iso}$.

Remark 1.2.1 We note that the inclusions $\mathcal{M}^{iso} \subseteq \mathcal{M}^{inj} \subseteq \mathcal{M}^{gen}$ are functors.

Remark 1.2.2 Note that, for any (X, d_X) and (Y, d_Y) in M^{gen} - $\text{Mor}_{\mathcal{M}^{gen}}((X, d_X), (Y, d_Y)) \neq \varnothing$. We can define $f : X \to Y$ with $x \mapsto y_0$ for a fixed $y_0 \in Y$. Clearly, this is a morphism in $\mathcal{M}^{gen}$.

1.2.2 Output Categories

Definition 1.2.3 Let Y be a finite set, $P_Y \in \mathcal{P}(Y)$ and $f : X \to Y$ a set map. We construct a function $f^* : \mathcal{P}(Y) \to \mathcal{P}(X)$ from f such that $f^*(P_Y)$ is a partition of X whose blocks are the preimages of the blocks in P_Y. With this, we can construct a category $\mathcal{C}$ whose objects are pairs of finite sets and their partitions (X, P_X), and whose morphisms between any two objects, (X, P_X) and (Y, P_Y), are all set maps $f : X \to Y$ such that P_X is a refinement of $f^*(P_Y)$. We call this the **category of outputs of standard clustering schemes**.

Lemma 1.2.4 Let X be a finite set and P_X be a partition of X.

1. If $Y = \{a, b\}$ and $P_Y = \{\{a\}\{b\}\}$, then $f \in \text{Mor}((X, P_X), (Y, P_Y))$ implies that whenever $f(x) \neq f(x')$, x and x' are in different blocks.

2. If $Y = \{a, b\}$ and $P_Y = \{\{a, b\}\}$. Then $g \in \text{Mor}((Y, P_Y), (X, P_X))$ implies $g(a)$ and $g(b)$ must be in the same block.

Proof.

1. Assume $f \in \text{Mor}((X, P_X), (Y, P_Y))$ and $f(x) \neq f(x')$. Since there are only two elements in Y, assume $f(x) = a$ and $f(x') = b$. Therefore, x is in the preimage of a and x' is in the preimage of b. Hence, x and x' must be in different blocks since a and b are in different blocks. □
2. Assume $g \in \text{Mor}((Y, P_Y), (X, P_X))$ and that $g(a)$ and $g(b)$ are in different blocks. Then a and b are in the preimages of two different blocks of P_X under g. Therefore, a and b must be in different blocks. Since this is not the case when $P_Y = \{\{a, b\}\}$, $g(a)$ and $g(b)$ must be in the same block. □

Definition 1.2.5 We call (X, θ_X) a persistent set when X is a finite set, and $\theta_X : [0, \infty) \to \mathcal{P}(X)$ is a function such that:

- If $r \leqslant s$, then $\theta_X(r)$ refines $\theta_X(s)$.
- For any r, there is a number $\epsilon > 0$, such that $\theta_X(r) = \theta_X(r')$ for all $r' \in [r, r + \epsilon]$

It is clear that a persistent set is a more relaxed form of a dendrogram. In a persistent set, we need not start with discrete partition and end with a single block partition.

Definition 1.2.6 Let $\mathcal{P}$ be a category whose objects are all persistent sets and whose morphisms between any two objects, (X, θ_X) and (Y, θ_Y), are the maps $f : X \to Y$ such that for all $r \geqslant 0$, (X, θ_X) is a refinement of $f^*(Y, \theta_Y)$. We call this the **category of outputs of hierarchical clustering schemes**. We see later that relaxing the output of hierarchical clustering schemes to persistent sets is useful.

1.2.3 Clustering Schemes

Definition 1.2.7 Let $\mathcal{M}$ be one of $\mathcal{M}^{iso}$, $\mathcal{M}^{inj}$ or $\mathcal{M}^{gen}$. A clustering scheme is a functor $\mathscr{C}$ from $\mathcal{M}$ to either one of the output categories. A **standard clustering scheme** is a functor $\mathfrak{C} : \mathcal{M} \to \mathcal{C}$. Similarly, a **hierarchical clustering scheme** is a functor $\mathfrak{C} : \mathcal{M} \to \mathcal{P}$.

It represents the idea that the mapping metric spaces to one of the output spaces can be functorial. By definition, the clustering scheme will respect the morphisms in the input and output categories. Hence, for standard clustering

schemes $\mathscr{C}$, the following diagram commutes for all $(X, d_X) \xrightarrow{f} (Y, d_Y)$ in $\mathcal{M}$

$$\begin{array}{ccc} (X, d_X) & \xrightarrow{f} & (Y, d_Y) \\ {\scriptstyle \mathscr{C}}\downarrow & & {\scriptstyle \mathscr{C}}\downarrow \\ (X, P_X) & \xrightarrow{\mathscr{C}(f)} & (Y, P_Y) \end{array}$$

Similarly, we can make a similar commutative diagram for hierarchical clustering schemes $\mathscr{H}$.

$$\begin{array}{ccc} (X, d_X) & \xrightarrow{f} & (Y, d_Y) \\ {\scriptstyle \mathscr{H}}\downarrow & & {\scriptstyle \mathscr{H}}\downarrow \\ (X, \theta_X) & \xrightarrow{\mathscr{H}(f)} & (Y, \theta_Y) \end{array}$$

1.3 Standard Clustering

1.3.1 Vietoris-Rips clustering

We define an equivalence relation which is the equivalent (in a loose sense of the word) of the equivalence relation $\sim_{\text{Path}}$ in the context of finite metric spaces.

Definition 1.3.1 Let (X, d_X) be a finite metric space. For each $\delta > 0$, let $\sim_\delta$ be an equivalence relation such that for each $x, x' \in X$, $x \sim_\delta x'$ iff there is a sequence $x_0, x_1, ..., x_k \in X$ such that $x_0 = x$, $x_k = x'$ and $d_X(x_i, x_{i+1}) \leqslant \delta$ for all i. The equivalence class of x under $\sim_\delta$ is denoted by $[x]_\delta$

Using this definition we define a family of standard clustering schemes.

Definition 1.3.2 For each $\delta > 0$ we define the **Vietoris-Rips clustering functor**

$$\mathfrak{R}_\delta : \mathcal{M}^{gen} \to \mathcal{C}$$

as follows: every $(X, d_X) \in \mathcal{M}^{gen}$ gets mapped to $(X, P_X(\delta))$, where $P_X(\delta)$ is the partition of X whose blocks are the equivalence classes of X under $\sim_\delta$, and for every morphism $f : (X, d_X) \to (Y, d_Y)$, $\mathfrak{R}_\delta(f)$ is the associated set map f regarded as a morphism from $(X, P_X(\delta))$ to $(Y, P_Y(\delta))$ in $\mathcal{C}$

Proof of functoriality. Assume $f \in \text{Mor}_{\mathcal{M}^{gen}}((X, d_X), (Y, d_Y))$, and $x, x' \in X$, such that $x \sim_\delta x'$. By definition of $\sim_\delta$, there exists a sequence $x = x_0, x_1, ..., x_n = x'$ in X with $d_X(x_i, x_{i+1}) \leqslant \delta$ for all $i = 0, 1, ..., n-1$. Therefore, $d_Y(f(x_i), f(x_{i+1})) \leqslant \delta$ for all $i = 0, 1, ..., n-1$, allowing $f(x) \sim_\delta f(x')$. Hence, we notice that $f([x]_\delta) \subseteq [f(x)]_\delta$ implying $P_X(\delta)$ is a refinement of $f^*(P_Y(\delta))$. □

Note that $\mathfrak{R}_\delta$ can be restricted to the subcategories $\mathcal{M}^{iso}$ and $\mathcal{M}^{inj}$, resulting in functors $\mathfrak{R}_\delta^{iso} : \mathcal{M}^{iso} \to \mathcal{C}$ and $\mathfrak{R}_\delta^{inj} : \mathcal{M}^{inj} \to \mathcal{C}$. These will be referred to as $\mathfrak{R}_\delta$ when there is no ambiguity.

Remark 1.3.3 It is not clear why functors in this family are named Vietoris-Rips clustering schemes. One can assume that it refers to the concept of a Vietoris-Rips complex. Given $\delta > 0$, a Vietoris-Rips complex over a metric space is the collection of its subsets with diameter less than δ under d_X. The Vietoris-Rips functor, however, maps a metric space to the collection of its subsets with separation less than δ. Regardless of the name, the results in this paper still stand true. The functors will still be referred to by the same name in this document, or shortened to VR functors for brevity.

Lemma 1.3.4 The Vietoris-Rips clustering functor is surjective on $\mathrm{Ob}(\mathcal{C})$ for each $\delta > 0$.

Proof. Let X be a finite set a $P_X \in \mathcal{P}(X)$. We need to find a metric d_X such that $\mathfrak{R}_\delta(X, d_X) = (X, P_X)$. Define $d_X : X \times X \to \mathbb{R}_{\geqslant 0}$ to be

$$d_X(x, x') = \begin{cases} 0, & \text{if } x = x' \\ \delta, & \text{if } x \neq x' \text{ and } x \text{ and } x' \text{ are in the same block} \\ 2\delta, & \text{if } x \text{ and } x' \text{ are different blocks} \end{cases}$$

It is easy to see d_X is a metric on X. $\mathfrak{R}_\delta(X, d_X) = (X, P_X)$ follows directly from the definition of d_X. □

Definition 1.3.5 Let $F : \mathcal{A} \to \mathcal{B}$ be a functor. When, for each $A, A' \in \mathcal{A}$, the function

$$\begin{aligned} \mathrm{Hom}_{\mathcal{A}}(A, A') &\to \mathrm{Hom}_{\mathcal{B}}(F(A), F(A')) \\ f &\mapsto F(f) \end{aligned}$$

is injective, we call the F faithful. When the function is surjective, we say F is full.

Question 1.3.6 We can ask whether a Vietoris-Rips functor is full and/or faithful. Since we have proved that a VR functor is surjective for objects, we arbitrarily choose two objects in $\mathcal{C}$, say (X, P_X) and (Y, P_Y). We choose (X, d_X) and (Y, d_Y) such that $\mathfrak{R}_\delta(X, d_X) = (X, P_X)$ and $\mathfrak{R}_\delta(Y, d_Y) = (Y, P_Y)$, and ask whether

$$\mathrm{Hom}_{\mathcal{M}}((X, d_X), (Y, d_Y)) \to \mathrm{Hom}_{\mathcal{C}}((X, P_X), (Y, P_Y))$$

$$f \mapsto \mathfrak{R}_\delta(f)$$

is injective and/or surjective for each input category $\mathcal{M}^{gen}$, $\mathcal{M}^{inj}$ and $\mathcal{M}^{iso}$. In other words, we ask whether the set map associated to $f \in \mathrm{Hom}_{\mathcal{C}}((X, P_X), (Y, P_Y))$ in each $\mathcal{M}$ adheres to the restrictions imposed on the morphisms.

Conjecture 1.3.7

1. $\mathfrak{R}_\delta^{gen}$ is full but not faithful.
2. $\mathfrak{R}_\delta^{inj}$ is full and faithful.
3. $\mathfrak{R}_\delta^{iso}$ is not full but it is faithful.

1.3.2 Excisiveness

Once a finite metric space has been partitioned by a clustering scheme and can not be split further through recursive action of the same algorithm, we say that the clustering scheme is excisive. It is articulated formally as follows.

Definition 1.3.8 A standard clustering scheme $F : \mathcal{M} \to \mathcal{C}$ is called **excisive** if for each finite metric space (X, d_X), with $F(X, d_X) = (X, X_{\alpha\alpha\in A})$, we have

$$F(X_\alpha, d_{X|_{X_\alpha \times X_\alpha}}) = (X_\alpha, \{X_\alpha\})$$

Proposition 1.3.9 The Vietoris-Rips funtor is excisive.

Proof By definition, $\mathfrak{R}_\delta$ maps a finite metric space (X, d_X) to the set of its equivalence classes under $\sim_\delta$. It is clear that an equivalence class can't be split further by $\mathfrak{R}_\delta$ for a fixed δ

Non-excisive functors: All standard clustering schemes are not excisive. We describe a family of non-excisive standard clustering functors to prove this claim. Let $\underline{\mathbb{E}}$ be the category of the extended real line where the existence of a morphism $f : x \to y$ implies $x \geqslant y$. We call a functor of the form $\mathcal{J} : \mathcal{M} \to \underline{\mathbb{E}}$ an invariant. Note that this means that the following diagram is adhered to for every $(X, d_X), (Y, d_Y) \in \mathcal{M}$ for every invariant $\mathcal{J}$

$$\begin{array}{ccc} (X, d_X) & \xrightarrow{f} & (Y, d_Y) \\ {\scriptstyle\mathcal{J}}\downarrow & & \downarrow{\scriptstyle\mathcal{J}} \\ \mathcal{J}(X) & \xrightarrow{\geqslant} & \mathcal{J}(Y) \end{array}$$

meaning whenever $\mathrm{Mor}_{\mathcal{M}}(X, Y) \neq \emptyset$, we have $\mathcal{J}(X) \geqslant \mathcal{J}(Y)$.

Let $\eta : \underline{\mathbb{E}} \to \underline{\mathbb{E}}$ be a non-increasing function. Consider the clustering scheme $\widehat{\mathfrak{R}} : \mathcal{M}^{inj} \to \mathcal{C}$ defined in the following way: each finite metric space (X, d_X) is mapped to $(X, \widehat{P}_X)$ where $\widehat{P}_X$ is the partition induced by $\sim_{\eta(\mathcal{J}(X))}$ on X.

Proof of Functoriality of $\widehat{\mathfrak{R}}$. Assume $x \sim_{\eta(\mathcal{J}(X))} x'$ in X. By definition, $x = x_0, x_1, ..., x_n = x'$ in X with $d_X(x_i, x_{i+1}) \leqslant \eta(\mathcal{J}(X))$ for each $i = 0, 1, ..., n-1$. If $f \in \mathrm{Mor}_{\mathcal{M}^{inj}}(X, Y)$, then $\mathcal{J}(X) \geqslant \mathcal{J}(Y)$ and $d_Y(f(x_i), f(x_{i+1})) \leqslant \eta(\mathcal{J}(X))$. Since η is non-increasing, $d_Y(f(x_i), f(x_{i+1})) \leqslant \eta(\mathcal{J}(Y))$. Therefore, we have $f(x) \sim_{\eta(\mathcal{J}(Y))} f(x')$. □

Remark 1.3.10 In [CM13], the collection of the clustering schemes described above is called the family of non-excisive clustering schemes. However, $\widehat{\mathfrak{R}}$ is proved to be non-excisive in [CM13] for only one choice of $(\mathcal{J}, \eta)$, which is $\mathcal{J}(X) = \mathrm{sep}(X)$ (meaning each metric space (X, d_X) is mapped to its separation) and $\eta(r) = r^{-1}$.

Let $\widehat{\mathfrak{R}}(X, d_X) = (X, \{X_\alpha | \alpha \in A\}$. Pick an $\alpha \in A$ and consider the inclusion $X_\alpha \hookrightarrow X$. This is clearly a morphism in $\mathcal{M}^{inj}$. This implies for the $\mathcal{J}$ associated to $\widehat{\mathfrak{R}}$, we have $\mathcal{J}(X_\alpha) \geqslant \mathcal{J}(X)$. Since the η is non-increasing, $\eta(\mathcal{J}(X_\alpha)) \leqslant \eta(\mathcal{J}(X))$, implying, the partition $\widehat{P}_{X_\alpha}$ is a refinement of $\{X_\alpha\}$. With this, we can posit the following conjectures.

Conjecture 1.3.10

1. For any $\mathcal{J} : \mathcal{M}^{inj} \to \underline{\mathbb{E}}$ and a non-increasing $\eta : \underline{\mathbb{E}} \to \underline{\mathbb{E}}$, the associated clustering functor $\widehat{R}$ is non-excisive.
2. Any non-excisive clustering functor $F : \mathcal{M}^{inj} \to \mathcal{C}$ can be described by a choice of $(\mathcal{J}, \eta)$.
3. Or simply, any non-excisive clustering functor $F : \mathcal{M}^{inj} \to \mathcal{C}$ can be described by a choice of $\mathcal{K} : \mathcal{M}^{inj} \to \underline{\mathbb{E}}^{op}$

1.3.3 Classification of $\mathcal{M}^{iso}$-functorial clustering schemes

This section uses concepts from group theory that are not in the scope of this project. However, for the sake of completeness, the following theorem is included for the readers who are familiar with the necessary concepts. The theorem concerns the classification of $\mathcal{M}^{iso}$-functorial clustering schemes. In the following, $\mathcal{I}$ is the set of the isometry classes of all finite metric spaces. For each $\zeta \in \mathcal{I}$, let (X_ζ, d_{X_ζ}) denote an element of the class, G_ζ the isometry group of the class $\mathcal{I}$, the set of all fixed points p_ζ of the the action of G_ζ on $\mathcal{P}(X_\zeta)$.

Theorem 1.3.11 Any $\mathcal{M}^{iso}$-functorial clustering scheme determines a choice of $p_\zeta \in \Xi_\zeta$ for each $\zeta \in \mathcal{I}$, and conversely, each choice of p_ζ for each $\zeta \in \mathcal{I}$ determines an $\mathcal{M}^{iso}$-functorial clustering scheme.

1.3.4 Representable Clustering Functors

In this section, $\mathcal{M}$ is either $\mathcal{M}^{inj}$ or $\mathcal{M}^{gen}$. Let $\Delta_k(\delta)$ be the metric space with k points with $d(a,b) = \delta$ whenever $a \neq b$. We use this definition to make the following observation: for a fixed $\delta > 0$, when $d_X(x,x') \leqslant \delta$ in a metric space (X, d_X), we can say there exists a map $f \in \mathrm{Mor}_{\mathcal{M}}(\Delta_k(\delta), (X, d_X))$ with $\{x, x'\} \subseteq \mathrm{Im}(f)$. We use this observation to redefine the definition of $\sim_\delta$: when there exists a sequence $x = x_0, x_1, ..., x_n = x'$ in X and $f_1, f_2, ..., f_n$ in $f \in \mathrm{Mor}_{\mathcal{M}}(\Delta_2(\delta), (X, d_X))$ with $\{x_i, x'_{i+1}\} \subseteq \mathrm{Im}(f_i)$ for all $i = 1, ..., n-1$, we say $x \sim_\delta x'$.

In this way, we can say that, in a certain way, the Vietoris-Rips functor $\mathfrak{R}_\delta$ is generated by $\{\Delta_2(\delta)\}$. We can generalise this procedure to define a family of clustering schemes.

Let Ω be a fixed collection of finite metric spaces. We define the clustering functor $\mathfrak{C}^\Omega : \mathcal{M} \to \mathcal{C}$ as follows: each $(X, d_X) \in \mathrm{Ob}(\mathcal{M})$ is mapped to $(X, X_\alpha | \alpha \in A)$. Two points x, x' in X are in the same block if and only if there exist:

1. a sequence $x = x_0, x_1, ..., x_n = x'$ in X, and
2. a sequence of metric spaces $\omega_1, \omega_2, ..., \omega_n \in \omega$, and
3. for each $i = 1, 2, .., n-1$, there are pairs $(\alpha_i, \beta_i) \in \omega_i$ and morphisms $f \in \mathrm{Mor}_{\mathcal{M}}(\omega_i, (X, d_X))$ such that $f_i(\alpha_i) = x_i$, $f_i(\beta_i) = x_{i+1}$.

Also, $\mathfrak{C}^\Omega(f) = f$ for all morphisms.

Remark 1.3.12 Let Ω be a collection of finite metric spaces. Then, it is clear that for any $\omega \in \Omega$ $\mathfrak{C}^\Omega(\omega, \{\omega\}) = (\omega, \{\omega\})$. Just as in lemma (add number), the following is true: for any $f \in \mathrm{Mor}_{\mathcal{M}}(\omega, (X, d_X))$ for any finite metric space (X, d_X) and $\omega \in \Omega$, $\mathrm{Im}(f)$ is contained in a single block of $\mathfrak{C}^\Omega(X, d_X)$.

Definition 1.3.13 We say a clustering functor $\mathfrak{C}$ is representable when there exists, Ω, a collection of finite metric spaces such that $\mathfrak{C} = \mathfrak{C}^\Omega$. When this is the case, we say $\mathfrak{C}$ is represented by Ω. When Ω is a finite collection, we say $\mathfrak{C}$ is finitely representable.

Theorem 1.3.14 Let $\mathcal{M}$ be either $\mathcal{M}^{inj}$ or $\mathcal{M}^{gen}$. Then a clustering functor

$\mathfrak{C} : \mathcal{M} \to \mathcal{C}$ is excisive if and only if it is representable.

Sketch of Proof.(The proof for this theorem is as detailed as possible i [CM13]. To avoid recreating the same proof, a sketch of it is given here). When $\mathfrak{C}$ is excisive, we consider the collection of all blocks of all partitions obtained by applying $\mathfrak{C}$ to all metric spaces, i.e.

$$\Omega = \{(B, d_{X|_{B\times B}}) | B \in P, \text{ where } (X, P) = \mathfrak{C}(X, d_X) \text{ for some } (X, d_X)\}$$

to show that $\mathfrak{C} = \mathfrak{C}^{\Omega}$. This is done by proving the partitions induced by each functor refine each other.

For the second part of the proof i.e. proving $\mathfrak{C}$ is excisive when it is representable by, say Ω, remark 1.3.12 is used to prove that for each block B of a partition induced by $\mathfrak{C} = \mathfrak{C}^{\Omega}$, all $x \in B$ are in the same block of $\mathfrak{C}^{\Omega}(B, d_{X|_{B\times B}})$ where $d_{X|_{B\times B}}$ is restricted metric of the space whose partition B belongs to. □

1.3.5 Factorization Theorem

We describe a method of factorizing each representable clustering functor through the Vietoris-Rips functor. Representable functors are presented as the composition of $\mathfrak{R}_1$ and the functor $\mathfrak{T}$, which applies a change of metric on the input metric space as described below. The intuition for this section comes from the fact dendrograms can be expressed as ultrametric (a metric which satisfies $d(x, z) \leqslant \max(d(x, y), d(y, z)))$. This is explained in detail in [CM10a].

Definition 1.3.15 Let X be a finite set and $W_X : X \times X \to \mathbb{R}_+$ a symmetric function such that for all $x \in X$, $W_X(x, x) = 0$. Then, the maximal subdominant ultramteric relative to W_X, $\mathcal{U}(W_X)$ is given by

$$\mathcal{U}(W_X)(x, x') = \min\{\max_i W_X(x_i, x_{i+1}) | \{x_i\}_{i=0}^{k} \text{ with } x_0 = x,\ x_k = x'\}$$

Using this construct, we define a functor through which we can factor every finitely representable clustering scheme $\mathfrak{C}$ over $\mathcal{M}^{inj}$ and $\mathcal{M}^{gen}$. The functor is

$$\mathfrak{T}^{\Omega} : \mathcal{M} \to \mathcal{M}.$$

It maps each finite metric space (X, d_X) to (X, d_X^{Ω}), where the metric d_X^{Ω} is

given by $d_X^\Omega = \mathcal{C}(W_X^\Omega)$, with $W_X : X \times X \to \mathbb{R}_+$ defined by

$$\inf \{\lambda > 0 \,|\, \exists \omega \in \Omega \text{ and } f \in \mathrm{Mor}_{\mathcal{M}}(\lambda \cdot \omega, X) \text{ with } \{x, x'\} \subset \mathrm{Im}(f)\}$$

when $x \neq x'$, and 0 otherwise. Inf over an empty set is set to be $+\infty$. Note that the set is not empty as long as $|X| \geq |\omega|$ for some $\omega \in \Omega$. $\mathfrak{T}^\Omega(f) = f$ for all morphisms in $\mathcal{M}$.

Theorem 1.3.16 Let $\mathfrak{C}$ be clustering scheme over $\mathcal{M}^{inj}$ or $\mathcal{M}^{gen}$ which is finitely represented by $\Omega \subset \mathcal{M}$. Then $\mathfrak{C}$ can written as the composition $\mathfrak{C} = \mathfrak{R}_1 \circ \mathfrak{T}^\Omega$, where $\mathfrak{R}_1$ is the Vietoris-Rips clustering functor with $\delta = 1$

Proof. Let (X, d_X) be in $\mathcal{M}$ and set $\mathfrak{C}(X, d_X) = (X, P)$ and $\mathfrak{R}_1 \circ \mathfrak{T}^\Omega(X, d_X) = (X, P')$. If $x, x' \in B \in P$, there is a sequence $x = x_0, x_1, ..., x_k = x' \in X$, $\omega_1, ..., \omega_n \in \Omega$, $f_i \in \mathrm{Mor}_{\mathcal{M}}(\omega_i, X)$, and $(\alpha_i, \beta_i) \in \omega_i$ such that $f_i(\alpha_i) = x_{i-1}$ and $f_i(\beta_i) = x_i$ for each i. Since there is no rescaling of ω's required, we have $W_X^\Omega(x_i, x_{i+1}) \leq 1$ and therefore $\max_i W_X^\Omega(x_i, x_{i+1}) \leq 1$. It follows that $d_X^\Omega(x, x') \leq 1$. It follows that x and x' are in the same block of P', which means P refines P'.

Assume $x, x' \in B' \in P'$. Therefore, we have $x \sim_1 x'$ in $(X, d_X^\Omega$ which means there exist $x = x_0, x_1, ..., x_k = x' \in X$ with $d_X^\Omega(x_i, x_{i+1}) \leq 1$ for each i. By definition of d_X^Ω, each (x_i, x_{i+1}) can be further split into $x_i = y_0^{(i)}, .., y_{k_i}^{(i)} = x_{i+1}$ and $W_X^\Omega(y_j^{(i)}, y_{j+1}^{(i)}) \leq 1$ for each j. Stacking up all the $\{y_j^{(i)}\}_{j=1}^{k_i}\}$, we obtain $x = z_0, ..., z_N = x'$ with $W_X^\Omega(z_i, z_{i+1}) \leq 1$ for all i. By definition of W_X^Ω, there exist for each i $\lambda_i \in (0, 1]$, $\omega_i \in \Omega$, and $\phi_i \in \mathrm{Mor}_{\mathcal{M}}(\lambda_i \cdot \omega_i, X)$ with $\{z_i, z_{i+1}\} \subset \mathrm{Im}(\phi_i)$. This implies $\phi_i \in \mathrm{Mor}_{\mathcal{M}}(\omega_i, X)$ and $\mathfrak{C}$ being represented by Ω, we see that x and x' must be in the same block of P. Therefore P' refines P. □

1.3.6 Clustering functors for $\mathcal{M}^{gen}$

With the following theorem regarding uniqueness, we see that the collection of clustering schemes on $\mathcal{M}^{gen}$ is severly restricted.

Theorem 1.3.17 Let $\mathfrak{C} : \mathcal{M}^{gen} \to \mathcal{C}$ be a functor such that there exists a $\delta_{\mathfrak{C}}$ so that

- $\mathfrak{C}(\Delta_2(\delta))$ has one block for all $\delta \in [0, \delta_{\mathfrak{C}}]$
- $\mathfrak{C}(\Delta_2(\delta))$ has two blocks for all $\delta > \delta_{\mathfrak{C}}$

Then, $\mathfrak{C}$ is the Vietoris-Rip functor with $\delta = \delta_{\mathfrak{C}}$, i.e. $\mathfrak{C} = \mathfrak{R}_{\delta_{\mathfrak{C}}}$

Proof[1] Let $\mathfrak{C}(X, d_X) = (X, P)$ and $\mathfrak{R}_{\delta_\mathfrak{C}}(X, d_X) = (X, P')$ for some finite metric space (X, d_X). We divide the proof in two parts. In the first, we prove that whenever $x, x' \in B \in P'$, i.e x, x' are in the same block of P', they must be in the same block of P. In the second, we prove that whenever x, x' are in different blocks of P', they must be in different blocks of P.

Assume x, x' are in some $B \in P'$. By definition, there exist a sequence $x = x_0, x_1, ..., x_n = x'$, such that for all i, we have $d_X(x_i, x_{i+1}) \leqslant \delta_\mathfrak{C}$. Therefore, we can reinterpret this as: for all i there exist $f_i \in \mathrm{Mor}_{\mathcal{M}_{gen}}(\Delta_2(\delta_\mathfrak{C}), X)$ such that $\mathrm{Im}(f_i) = \{x_i, x_{i+1}\}$. By functoriality, the following diagram must commute

$$\begin{array}{ccc} \Delta_2(\delta_\mathfrak{C}) & \xrightarrow{f_i} & (X, d_X) \\ \mathfrak{C}\downarrow & & \downarrow\mathfrak{C} \\ \mathfrak{C}(\Delta_2(\delta_\mathfrak{C})) & \xrightarrow{\mathfrak{C}(f_i)} & (X, P) \end{array}$$

By definition of $\mathfrak{C}$, $\mathfrak{C}(\Delta_2(\delta_\mathfrak{C}))$ is one piece. Therefore, since $\mathrm{Im}(f_i) = \{x_i, x_{i+1}\}$, x_i and x_{i+1} must be in the same block of P for each $i = 1, ..., n-1$.

Now for the second part of the proof, assume that x, x' are in different blocks of P'. Let $x \in B$ and $x' \in X\backslash B$. Set $\delta := \min\{d_X(b, b') | b \in B, b' \in X\backslash B\}$ and note that, by definition, we have $\delta > \delta_\mathfrak{C}$. Let p, q be two distinct elements in $\Delta_2(\delta)$ and define a map $\phi : X \to \Delta_2(\delta)$ such that $\phi(z) = p$ for all $z \in X\backslash B$ and $\phi(z) = q$ for all $z \in B$. Now to prove that ϕ is metric map, if either $x, x' \in B$ or $x, x' \in X\backslash B$, then $\phi(x) = \phi(x')$. If $x \in B$ and $x' \in X\backslash B$, we have, by definition of δ $d_X(x, x') \geqslant \delta = d(p, q)$. Therefore, it is clear that ϕ is a metric map. Considering the following diagram

$$\begin{array}{ccc} (X, d_X) & \xrightarrow{\phi} & \Delta_2(\delta_\mathfrak{C}) \\ \mathfrak{C}\downarrow & & \downarrow\mathfrak{C} \\ (X, P) & \xrightarrow{\mathfrak{C}(\phi)} & \mathfrak{C}(\Delta_2(\delta_\mathfrak{C})) \end{array}$$

and noting that $\mathfrak{C}(\Delta_2(\delta_\mathfrak{C}))$ is in two pieces, we see that x, x' must be in two separate blocks of P. □

Remark 1.3.18 The definition of $\mathfrak{R}_\delta$ requires that x, x' are clustered in the same block if and only if there exist $x = x_0, .., x_n = x'$ such that $d(x_i, x_{i+1}) \leqslant \delta$ for each. If there inequality is changed to $d(x_i, x_{i+1}) < \delta$, the proof above can be altered easily to prove the following: Let $\mathfrak{C} : \mathcal{M}^{gen} \to \mathcal{C}$ be a functor such that there exists a $\delta_\mathfrak{C}$ so that

[1] There were a few typos in [CM13]'s proof of this theorem. The typos have been corrected and more detail has been added.

- $\mathfrak{C}(\Delta_2(\delta))$ has one block for all $\delta \in [0, \delta_{\mathfrak{C}})$
- $\mathfrak{C}(\Delta_2(\delta))$ has two blocks for all $\delta \geqslant \delta_{\mathfrak{C}}$

Then, $\mathfrak{C}$ is the Vietoris-Rip functor with $\delta = \delta_{\mathfrak{C}}$, i.e. $\mathfrak{C} = \mathfrak{R}^{\circ}_{\delta_{\mathfrak{C}}}$

1.3.7 Scale Invariance

We now apply Kleinberg's scale invariance restriction on $\mathcal{M}^{gen}$ and $\mathcal{M}^{inj}$ functorial clustering scheme. It is interesting to see that scale invariance imposes severe restrictions on the mentioned schemes.

Definition 1.3.19 We define $\sigma_\lambda : \mathcal{M}^{gen} \to \mathcal{M}^{gen}$ to be a functor such that for all objects $\sigma_\lambda(X, d_X) = (X, \lambda \cdot d_X)$ and for all morphisms $\sigma_\lambda(f) = f$. Note that the functor can be restricted to $\mathcal{M}^{iso}$ and $\mathcal{M}^{inj}$.

Theorem 1.3.20 Let $\mathfrak{C} : \mathcal{M}^{gen} \to \mathcal{C}$ be a clustering functor such that $\mathfrak{C} \circ \sigma_\lambda = \mathfrak{C}$ for all $\lambda > 0$. Then either

- $\mathfrak{C}$ assigns each finite metric space to its single partition.
- $\mathfrak{C}$ assigns each finite metric space to its partition of singletons, or

Proof We start the proof by observing that $\Delta_2(1)$ is either in one piece or two pieces.

Assume $\Delta_2(1)$ is in one piece. Then if $\mathfrak{C} = \mathfrak{C} \circ \sigma_\lambda$, $\Delta_2(\delta)$ must be in one piece too for all $\delta > 0$. Choose a finite metric space (X, d_X) and let $\mathfrak{C}(X, d_X) = (X, P)$. Let $x, x' \in X$. Let $d_X(x, x') = \delta_0$. Choose a morphism $\phi : \Delta_2(\delta_0) \to (X, d_X)$ such that $\phi(p) = x$ and $\phi(q) = x'$, where p and q are the two elements of $\Delta_2(\delta_0)$. It is easy to see that $\phi \in \mathcal{M}^{gen}(\Delta_2(\delta), (X, d_X))$. We see that the following diagram commutes

$$\begin{array}{ccc} \Delta_2(\delta_0) & \xrightarrow{\phi} & (X, d_X) \\ \mathfrak{C}\downarrow & & \downarrow\mathfrak{C} \\ \mathfrak{C}(\Delta_2(\delta_0)) & \xrightarrow{\mathfrak{C}(\phi)} & (X, P) \end{array}$$

Therefore, by lemma 1.2.4, x and x' must in the same block. Since x, x' were arbitrarily chosen, all of X must be in the same block i.e. $\mathfrak{C}(X, d_X) = (X, \{X\})$.

Assume $\Delta_2(1)$ is in two pieces. Then if $\mathfrak{C} = \mathfrak{C} \circ \sigma_\lambda$, $\Delta_2(\delta)$ must be in two pieces too for all $\delta > 0$. Choose a finite metric space (X, d_X) and let $\mathfrak{C}(X, d_X) = (X, P)$. Let $x, x' \in X$. Note we can construct a partition $\{A, B\}$ of such that $x \in A$ and $x' \in B$. let separation$(X) = \delta_0$. Construct a $\phi : (X, d_X) \to \Delta_2(\delta_0)$ such that $\phi(a) = p$ for all $a \in A$ and $\phi(b) = q$ for all $b \in B$ otherwise. It is

easy to see that $\phi \in \mathcal{M}^{gen}((X, d_X), \Delta_2(\delta_0))$. We see that following diagram commutes

$$\begin{array}{ccc} (X, d_X) & \xrightarrow{\phi} & \Delta_2(\delta_0) \\ \mathfrak{C}\downarrow & & \downarrow\mathfrak{C} \\ (X, P) & \xrightarrow{\mathfrak{C}(\phi)} & \mathfrak{C}(\Delta_2(\delta_0)) \end{array}$$

Therefore, by lemma 1.2.4, x and x' are different blocks of P. Since a partition like the one above can constructed for all x and x', all points are in different blocks of P. Since, X was arbitrarily chosen, $\mathfrak{C}$ assigns every finite metric spaces its discrete partition given $\Delta_2(1)$ is in two pieces. □

Theorem 1.3.20 Let $\mathfrak{C} : \mathcal{M}^{inj} \to \mathcal{C}$ be a clustering functor such that $\mathfrak{C} \circ \sigma_\lambda = \mathfrak{C}$ for all $\lambda > 0$. Let

$$K(\mathfrak{C}) := \{k \geqslant 2;\ \mathfrak{C}(\Delta_k(1) \text{ is in one piece})\}$$

- $K(\mathfrak{C}) = \emptyset$, then $\mathfrak{C}$ assigns to each finite metric space its partition of singletons
- Otherwise, let $k_{\mathfrak{C}} = K(\mathfrak{C})$. Then
  - for each $k \geqslant k_{\mathfrak{C}}$, $\mathfrak{C}$ assigns to each finite metric space of size k, the partition of a single block.
  - for each $2 \leqslant k \leqslant k_{\mathfrak{C}}$, $\mathfrak{C}$ assigns to each finite metric space of size k, the partition of singletons.

Proof. Let's start with the case when $K(\mathfrak{C}) = \emptyset$. Choose any X and let $k = |K|$. Choose arbitrarily x and x' and an injective function $\phi : X \to \Delta_k(\text{sep}(X))$. $K(\mathfrak{C}) = \emptyset$ implies $\mathfrak{C}(\Delta_k(1)$ is a discrete partition which in turn implies $\mathfrak{C}(\Delta_k(\text{sep}(X))$ is a discrete partition. We have chosen $\text{sep}(X)$ for δ because then ϕ is a morphism in $\mathcal{M}$ automatically. Through functoriality, x and x' must be in different blocks of the partition imposed by $\mathfrak{C}$. Since x and x' were chosen arbitrarily, all points are in different blocks of $\mathfrak{C}(X)$

Assume now $K(\mathfrak{C}) \neq \emptyset$ and $k_{\mathfrak{C}}$. Pick X of size k. $\mathfrak{C}(\Delta_k(1)$ is in one piece implies $(\Delta_k(\text{diam}(X))$ must be in one piece too. Note that an injective function $\phi : \Delta_k(\text{diam}(X) \to X)$ must be in $\mathcal{M}$. Through functoriality, we conclude $\mathfrak{C}(X)$ must be in one piece.

Assume $2 \leqslant k_{\mathfrak{C}}$ Choose any X with $k = |K|$. Choose arbitrarily x and x' and an injective function $\phi : X \to \Delta_2(\text{sep}(X))$. $2 \leqslant k_{\mathfrak{C}}$ implies $\mathfrak{C}(\Delta_k(1)$ is a discrete partition which in turn implies $(\Delta_k(\text{sep}(X))$ is a discrete partition.

Like in the first case of this proof, we have chosen $\text{sep}(X)$ for δ because then ϕ is a morphism in $\mathcal{M}$ automatically. Through functoriality, x and x' must be in different blocks of the partition imposed by $\mathfrak{C}$. Since x and x' were chosen arbitrarily, all points are in different blocks of $\mathfrak{C}(X)$. □

1.3.8 Clustering functors on $\mathcal{M}^{inj}$ and density

In this section, we use the concept of representability to propose a family of functors which contains the Vietoris-Rips functors. We see how functors in this family have practical interest when it comes to dealing with density.

We saw how a Vietoris-Rips functor $\mathfrak{R}_\delta$ is a represented by $\Omega = \{\Delta_2(\delta)\}$ in section 3.4. We consider the family of clustering functors represented by $\{\Delta_n(\delta)\}$ for each $n \in \mathbb{N}$ and $\delta > 0$. Consider the following figure

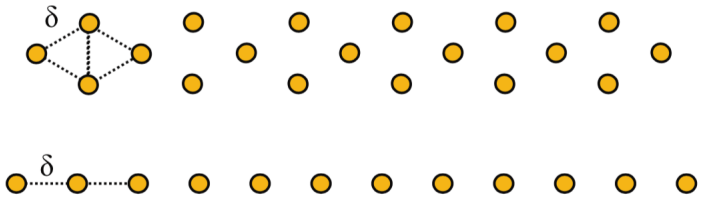


Figure 1.5: Image from [CM13]

Note that for a fixed δ, $\mathfrak{C}^{\Delta_2(\delta)} = \mathfrak{R}_\delta$ will cluster each of the two metric spaces into a single blocks partitions. However, $\mathfrak{C}^{\Delta_3(\delta)}$ will only cluster the metric space on top into a single blocks and will assign a discrete partition to the metric space on the bottom. It is clear that as we increase n, $\mathfrak{C}^{\Delta_n(\delta)}$, will become more sensitive to density.

For each $n \in \mathbb{N}$ and $\delta > 0$, we can associate a clustering functor $\mathfrak{C}^{\Delta_n(\delta)}$ to an equivalence relation $\sim_{\Delta_n\delta}$ which can be defined as follows: given a finite metric space (X, d_X), we say $x \sim_{\Delta_n\delta} x'$ iff there exist a sequence $x = x_0, ..., x_k = x'$ in X such that each consecutive $\{x_i, x_{i+1}\}$ is contained in a subset S_i of X such that $|S_i| = n$ and $\text{diameter}(S_i) \leqslant \delta$.

In [CM13], the following corollary is offered.

Corollary 1.3.21 On $\mathcal{M}^{inj}$, for each $m \in \mathbb{N}$

$$\mathfrak{C}^{\Delta_m(\delta)} = \mathfrak{R}_\delta \circ \mathfrak{T}^m$$

where $\mathfrak{T}^m = \mathfrak{T}^\Omega$ for $\Omega = \{\Delta_m(\delta) | \delta \geqslant 0\}$ is the functor from section 3.5.

It is mentioned that the corollary follows from 3.5.2, however it is not clear how as a sufficient explanation is provided. It is easy to see that $\mathfrak{C}^{\Delta_m(\delta)}$ can be equated with $\mathfrak{R}_1 \circ \mathfrak{T}^{\Delta_m(\delta)}$ using theorem 3.5.2, the jump from this to the corollary is not clear.

1.4 Hierarchical Clustering

We define the functor on $\mathcal{M}^{gen}$

$$\mathfrak{R} : \mathcal{M}^{gen} \to \mathcal{P}$$

as follows. Each finite metric space (X, d_X) is assigned to the persistent set (X, θ_X^{VR}), where for each $r \geqslant 0$ $\theta_X^{VR}(r)$ is the partition induced by $\sim_r$. This is clearly an object an $\mathcal{P}$. For each $f : X \to Y$, $\mathfrak{R}(f) = f$ is simply the set map f regarded as an morphism in $\mathcal{P}$ from $(X, \theta_X^{VR}) \to (Y, \theta_Y^{VR})$.

Clearly this functor is hierarchical version of single linkage clustering in the sense that for each $\delta \geqslant 0$, if one writes $\mathfrak{R}_\delta(X, d_X) = (X, P_X(\delta))$, then $(P_X(\delta) = \theta_X^{VR}(\delta)$.

1.4.1 Functoriality over $\mathcal{M}^{gen}$: a uniqueness theorem

The theorem below is the analogue of Kleinberg's theorem for hierarchical clustering schemes provided in [CM13]. Given some plausible restrictive requirements, existence and uniqueness is claimed instead of impossibility.

Theorem 1.4.1 Let $\mathfrak{H} : \mathcal{M}^{gen} \to \mathcal{P}$ be a hierarchical clustering funtor with the following properties:

- **(I)** Let $\alpha : \mathcal{M}^{gen} \to \mathbf{Set}$ and $\beta : \mathcal{P} \to \mathbf{Set}$ be forgetful functors $(X, d_X) \mapsto X$ and $(X, \theta_X) \mapsto X$. Then we assume $\beta \circ \mathfrak{H} = \alpha$, the functor $\mathfrak{H}$ does not change the underlying set.
- **(II)** For $\delta \geqslant 0$, let $\Delta_2(\delta)$ denote the two point metric space with underlying set $\{\mathrm{p}, \mathrm{q}\}$. Then $\mathfrak{H}(\Delta_2(\delta))$ is the persistent set $(\{\mathrm{p}, \mathrm{q}\}, \theta_{\Delta_2(\delta)})$, where $\theta_{\Delta_2(\delta)}(t)$ is the discrete partition for $t > \delta$, and is the partition with a single block for $t \leqslant \delta$
- **(III)** When $\mathfrak{H}(X, d_X) = (X, \theta^{\mathfrak{H}})$, the partition $\theta^{\mathfrak{H}}$ is the discrete partition for any $t < \mathrm{Sep}(X)$.

Then $\mathfrak{H}$ is equal $\mathfrak{R}$.

Proof. Choose a finite metric space (X, d_X) and let $\mathfrak{R}(X, d_X) = (X, \theta^{VR})$. Assuming **(I)**, let $\mathfrak{H}(X, d_X) = (X, \theta^{\mathfrak{H}})$. We prove that, assuming the listed conditions, for all $r \geqslant 0$, **(A)** $\theta^{VR}(r)$ is a refinement of $\theta^{\mathfrak{H}}(r)$ and that (B) $\theta^{\mathfrak{H}}(r)$ is a refinement of $\theta^{VR}(r)$. From **(A)** and **(B)**, the claim of the theorem follows.

We need to show that whenever x, x' lie in the same block of $\theta^{VR}(r)$, i.e. $x \sim_r x'$, they also lie in the same block in $\theta^{\mathfrak{H}}(r)$.

Assume $d_X(a, b) \leqslant r$, then we can say the function $\phi : \Delta_2(r) \to X$, with $\phi(p) = a$ and $\phi(q) = b$ is a morphism in $\mathcal{M}^{gen}$. With functoriality we obtain, $\mathfrak{H}(\phi) : \mathfrak{H}(\Delta_2(r)) \to \mathfrak{H}(X, d_X)$. Using **(II)**, we know that p and q lie in the same block of $\theta_{\Delta_2(r)}$. Therefore a and b must be in the same block in $\theta^{\mathfrak{H}}(r)$. To summarise, whenever $d_X(a, b) \leqslant r$, a and b lie in the block of $\theta^{\mathfrak{H}}(r)$. Using this statement, we prove whenever $x \sim_r x'$, x and x' lie in the same block in $\theta^{\mathfrak{H}}(r)$.

Let $x \sim_r x'$. Then there exist $x = x_0, ..., x_n = x'$ such that $d_X(x_i, x_{i+1}) \leqslant r$ for each i. Therefore, each pair (x_i, x_{i+1}) is in the same block in $\theta^{\mathfrak{H}}(r)$. Therefore, x and x' must lie in the same block of $\theta^{\mathfrak{H}}(r)$ also. With this, we have proved **(A)**.

Now we prove **(B)**. Let (X_r, d_r) be the metric space whose points are the equivalent classes of (X, d_X) under $\sim_r$, and d_r, the metric $\mathcal{U}(W_X)$, where W_X is given by

$$W_X(B, B') = \min_{x \in B} \min_{x' \in B'} d_X(x, x').$$

Note that for any $B, B' \in X_r$ we know $d_r(B, B')$ must be strictly greater than r. Therefore the separation of $(X_r, d_r) > r$.

Let $\mathfrak{H}(X_r, d_r) = (X_r, \theta^{\mathfrak{H}}_{X_r})$. Since, $\text{sep}(X_r) > 0$, using **(III)**, we directly see that the blocks of the partition $\theta^{\mathfrak{H}}_{X_r}$ are exactly the equivalence classes of X under the equivalence relation $\sim_r$, i.e. $\theta^{\mathfrak{H}}_{X_r} = \theta_{VR}$. Now consider,

$$\pi_r : (X, d_X) \to (X_r, d_r)$$

in $\mathcal{M}^{gen}$ such that $\pi_r(x) = [x]_r$. By functoriality, $\mathfrak{H}\pi_r : \mathfrak{H}(X, d_X) \to \mathfrak{H}(X_r, d_r)$ is persistence preserving, and therefore, $\theta^{\mathfrak{H}}(r)$ is refinement of $\theta^{VR}(r)$. □

Comments 1.4.2 Kleinberg's conditions, however reasonable, were too restrictive. With the restrictions imposed on the hierarchical clustering functors, we see how three analogues of Kleinberg's condition are satisfied.

- It is interesting to see that the hierarchical Vietoris-Rips functor satisfies

the property $\mathfrak{R} \circ \sigma_\lambda = s_\delta \circ \mathfrak{R}$. Here, $s_\lambda : \mathcal{P} \to \mathcal{P}$ is defined by $s_\lambda(X, \theta_X) = (X, \theta_X^\lambda)$, with $\theta_X^\lambda(r) = \theta_X(\frac{r}{\lambda})$ for each $r \in [0, \infty]$. This implies that *scaling metric spaces before applying the VR functor gives an appropriately scaled output.* This condition is trivially satisfied. This result is ideal since scaled inputs should result in equally scaled outputs and it indicates that $\mathfrak{R}$ is consistent.

- Each dendrogram for a dataset can be achieved as an output of $\mathfrak{R}^{gen}$ applied on a pseudometric space. To see this, choose a persistent set (X, θ_X). Let $r_1, ..., r_n$ be the transition points of the persistent set. Define the pseudo metric, $d(x, x') = \min\{r_i | x, x' \textit{ belong to the same block of } \theta_X(r_i)\}$. To see that this is a pseudometric space, note that the $d(x, x) = 0$ and $d(x, y) = d(y, x)$ are trivially satisfies. Let $d(x, x') = r_i$ and $d(x', x'') = r_j$. Then $d(x, x'') = \max\{r_i, r_j\}$. It follows that $d(x, x'') \leqslant r_i + r_j = d(x, x') + d(x', x'')$. This proves d is pseudometric on X.

- Consisteny, another one Kleinberg's conditions is trivially satisfied. It is also important to note that expecting consistency as defined by Kleinberg from standard clustering scheme is unreasonable. If all clusters are kept the same size, except just one which is shrunk, the shrinking will cause points to get further away from some but also closer to the others.

1.4.2 Functoriality over $\mathcal{M}^{inj}$

Let (X, d_X) be a finite metric space and $r \geqslant 0$. Let $c : X \to \mathbb{N}$ be function with $c(x) = |[x]_r|$, where $[x]_r$ is the equivalence class of x under $\sim_r$. For any m, we define $X_m \subseteq X$ with $X_m = \{x \in X | c(x) \geqslant m\}$.

Lemma 1.4.3 Let $f : X \to Y$ be a morphism in $\mathcal{M}^{inj}$. Then $f(X_m) \subseteq Y_m$. *Proof* We need to show whenever $x \in X_m$, and $f(x) = y$, we have $y \in Y_m$. Let $x \in X_m$. Then $|[x]_r| \geqslant m$. Whenever $x \sim_r x'$, we know $f(x) \sim_r f(x')$. Then $f([x]_r) \subseteq [f(x)]_r$. Since f is injective, $|f([x]_r)| \leqslant |[f(x)]_r|$ implying $|[y]_r| \geqslant m$. □

Remark 1.4.4 Note that this lemma would not be true if f wasn't injective.

With this observation, we can define an equivalence relation $\sim_r^m$ on X with $x \sim_r^m$ iff $x \sim_r x'$ and $c(x) \geqslant m$, otherwise the equivalence class of x is just $\{x\}$. Note that $x \sim_r^m$ refines $x \sim_r$. With the we define a new persistent set (X, θ_X^m), where $\theta_X^m(r)$ is the partition induced by $\sim_r^m$. (prove functoriality).

Definition 1.4.5 For each $m \in \mathbb{N}$, let $\mathfrak{T}^m : \mathcal{M}^{inj} \to \mathcal{P}$ be the funtor that maps finite metric spaces (X, d_X) to the persistent set (X, θ_X^m).

These clustering scheme can be used to stop cluster of cardinality $< m$ from accumulating.

Definition 1.4.6 For each $m \in \mathbb{N}$, define $\mathfrak{R}^{\Delta_m} : \mathcal{M}^{inj} \to \mathcal{P}$ given by

$$\mathfrak{R}^{\Delta_m} = \mathfrak{R} \circ \mathfrak{T}^m$$

where $\mathfrak{T}^m = \mathfrak{T}^\Omega$ for $\Omega = \{\Delta_m(\delta) | \delta \geqslant 0\}$.

1.5 Discussion

I hope a decent account of [CM13] has been provided with adequate additions in detail to make this accessible to a more general audience. It is clear that categorical concept provide building blocks for a strong framework of clustering schemes. It can be said that given constraints on clustering schemes, there is likely to be a unique universal construction that adheres to the constraints. More exploration of constraints is encouraged. Also, more possible applications of CT can be expected to model other types of classification such as fuzzy clustering.

Chapter 2

DATABASE THEORY

2.1 Model for Databases

In this chapter, a model for databases as described by [Spi12] is explained. First, a rigorous equivalence of the model is established with the category **Cat**, and hence, relation between category theory and the theory of databases is demonstrated consequently. Then, data migration functors are described with as little complexity as possible without compromising on key details. We finish off with incorporating data types and data filtering. Many categorical database models had been described before [Spi12], (some of them being [JD01], [BPV06], [JRW02], [RW92]) but no prior knowledge is required to understand the [Spi12] model. We start by considering the example of a database given below and develop the model using the vocabulary of graphs.

Example 2.1.1 Consider the following tables:

Employee			
ID	EName	WorksIn	EmployedBy
E11	John	Com1.13	Com1
E22	Julie	Com22.3	Com22
E33	Bella	Com1.9	Com1
E44	Clive	Com7.4	Com7
E55	Tom	Com22.3	Com22

String
ID
John
Julie
Bella
Clove
Kameca
Borenns
Bloom
Tom
IT
Sales
Marketing
Security

Department		
ID	DName	Deptin
Com1.13	Sales	Com1
Com22.3	Marketing	Com22
Com1.9	IT	Com1
Com7.4	Security	Com7

Company	
ID	SName
Com1	Kameca
Com22	Borenns
Com7	Bloom

Data is only useful when it is stored in a suitable formation which accurately represents how different data points are related to each other. A database, such as the one represented by the tables above, can be described as a collection of tables where each table must contain an ID column and may or may not contain foreign keys which connect the table to other tables. An ID column contains entries that act like row labels. For example, E11 in the 'Employee' table represents its row. Therefore, row labels must be unique. A foreign key, in say table t, refers to an entry in the ID column of t to an entry in an ID column of another table, hence interlocking the two tables and creating a coherent representation of data points are related. For example, 'WorksIn' is a foreign key in the 'Employee' table and refers to the 'Department' table. The implication is that E11 works in Com1.13.

2.1.1 Database schemas as graphs

Now we see how graphs can act as templates for databases. There are multiple accepted definitions of a graph. Each definition has its own domain of importance and researchers pick the definition best suitable for their purposes. For the model described in [Spi12], the following definition is used. Definitions relevant to the model based on the notation given in the graph definition are also given.

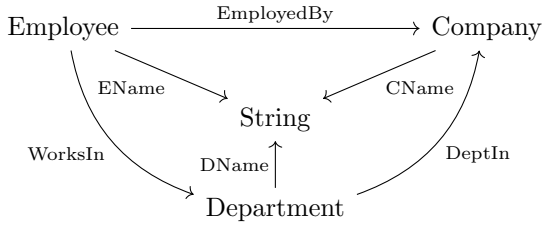
Definition 2.1.2 A *graph* G contains two sets, the set of its vertices V and the set of its arrows A, and two functions $s, t : A \to V$, where s maps each arrow to its source and t maps each arrow to its target. Therefore, given $a \in A$ with $s(a) = x$ and $t(a) = y$, we say a is an arrow from x to y. We write $G = (V, A, s, t)$

Definition 2.1.3 A *path* in a given graph G, is a concatenation of arrows where the target of one arrow is the source of the next one. Formally, a path of length n, from a vertex x to y, is a sequence $(a_k)_{k=1}^{n}$ such that $s(a_1) = x, t(a_1) = s(a_2), ..., s(a_{n-1}) = t(a_n)$. The set of paths of length n in a given graph G is denoted by $\text{Path}_G^{(n)}$. Hence, we can write A as $\text{Path}_G^{(1)}$, V as $\text{Path}_G^{(0)}$ and the set of all paths in G as

$$\text{Path}_G = \bigcup_{n=1}^{\infty} \text{Path}_G^{(n)}$$

Example 2.1.4 Note that the following graph can be interpreted as a template

for the database described on the previous page



With this representation, we can define a *database schema* to be a graph where each vertex represents a table and each arrow a represents a foreign key in the table represented by $s(a)$ pointing towards the table represented by $t(a)$. Arrows can be concatenated so that a sequence $v_1 \xrightarrow{a_1} v_1 \to \cdots \xrightarrow{a_{n-1}} v_n$ links the tables associated to vertices v_1 and v_n through the foreign keys associated with the arrows. It is possible to have two different paths from v_1 to v_2 but they need not represent the same mapping. For example, the database described through tables is constructed in such a way that the two mappings 'DeptIn $\circ$ WorksIn' and 'EmployedBy' are the same. However, the mapping 'EName' and 'DName $\circ$ WorksIn' represent two different mappings even though their sources and targets are the same. The data and the architect of the schema determine whether two paths are equivalent or not. We declare two paths are equivalent with the sign $\simeq$. Path equivalence represent key relations in database and hence, are an important feature in this model.

Roughly speaking, a category can be defined through the vocabulary of graphs and an equivalence relation with which ones declares two paths equal. We have now establish a rough relationship between databases and categories. This relationship is precisely what is to be established formally in this section.

Using the ideas presented above, the following 'normal form' of databases is described in [Spi12].

Definition 2.1.5 A database has a categorical normal iff

- every table t has a single primary key column ID_t, chosen at the outset. The cells in this column are called the row-ids of t;
- for every column c of a table t, there exists some target table t' such that the value in each cell of column c refers to some row-id of t'. We denote this relationship by $c : t \to t'$.
- in particular, if some column d of t consists of "pure data" (such as strings or integers), then its target table t' is simply a 1-column table (i.e. a

controlled vocabulary, containing at least the active domain of column d), and we still write $d : t \to t'$; and

- when there are two paths p, q through the database from table t to table u (denoted $p : t \to u, q : t \to u$) and it is known to the schema designer that p and q should correspond to the same mapping of row-ids, then this path equivalence must be declared as part of the schema. We denote this path equivalence by $p \simeq q$

2.1.2 Path Equivalence

We need to formally articulate the notion of path equivalence. The path equivalence needs to be an equivalence relation which respects sources, targets and composition. Hence, the following definition arises.

Definition 2.1.6 Let $G = (V, A, s, t)$ be a graph. A categorical path equivalence relation (or CPER) on G is an equivalence relation $\simeq$ on $\mathrm{Path_G}$ that has the following properties:

1. If $p \simeq q$ then $s(p) = s(q)$.
2. If $p \simeq q$ then $t(p) = t(q)$.
3. Suppose $p, q : b \to c$ are paths, and $m : a \to b$ is an arrow. If $p \simeq q$ then $mp \simeq mq$.
4. Suppose $p, q : a \to b$ are paths, and $n : b \to c$ is an arrow. If $p \simeq q$ then $pn \simeq qn$.

We prove that the definition given above requires that the composition of two pairs of equivalent paths must be equivalent.

Lemma 2.1.7 Suppose that G is a graph and $\simeq$ is a CPER on G. Suppose $p \simeq q : a \to b$ and $r \simeq s : b \to c$. Then $pr \simeq qs$.

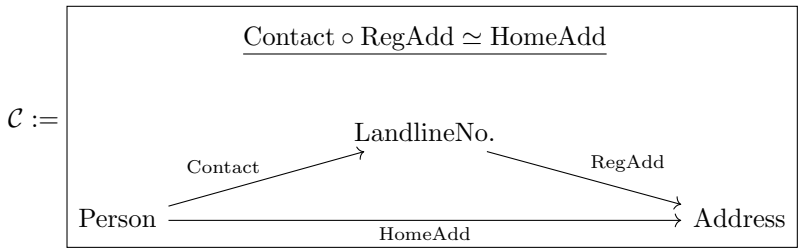
Proof. $p \simeq q : a \to b$ and $s : b \to c$ imply $ps \simeq qs$. And $p : a \to b$ and $r \simeq s : b \to c$ imply $pr \simeq ps$. $pr \simeq ps$ and $ps \simeq qs$ together imply $pr \simeq ps$. □

The definition of a database schemas is reinterpreted as follows.

Definition 2.1.8 A *database schema* $\mathcal{C}$ consists of a pair $\mathcal{C} := (G, \simeq)$ where G is a graph and $\simeq$ is a categorical path equivalence relation on G. We sometimes refer to a database schema as a schema for brevity.

Example 2.1.9 The graph in example 1.3 is an example of a database schema. However, the path equivalence wasn't explicitly mentioned in its presentation.

Here's another example of a schema with a path equivalence mentioned explicitly.



The schema by design requires that the registered address of a person's landline number should be the same as their home address.

2.1.3 Schema instances

Database schemas are graphical templates of databases. The database itself is presented using a **Set**-valued instance of the database schema.

Definition 2.1.10 Let $\mathcal{C} = (G, \simeq)$ be a database schema, where $G = (V, A, s, t)$. An instance I of $\mathcal{C}$ consists of:

(a) For every $v \in V$, a set $I(v)$.

(b) For every $a : v \to v'$, a function $I(a) : I(v) \to I(v')$

(c) For each path equivalence $p \simeq q$, $I(p) = I(q)$ must be true.

Example 2.1.11 The tables given in the beginning of example 1.1 define an instance of the schema defined in example 1.4. Here's an instance defined on the schema given in example 1.9

Person		
ID	Contact	HomeAdd
P1	LLN90	Add67
P2	LLN81	Add98
P3	LLN79	Add78

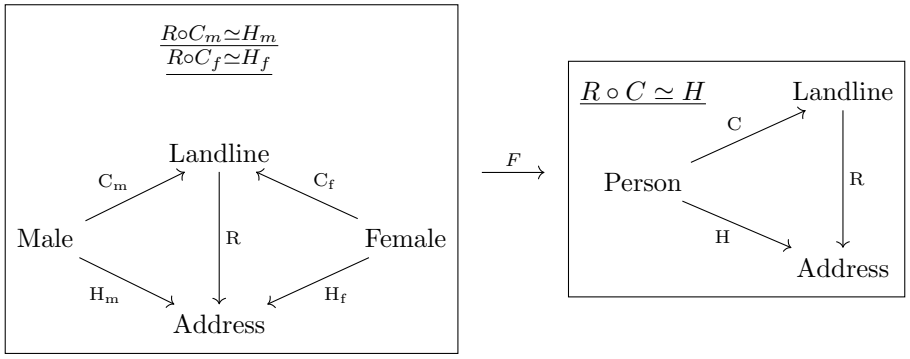
Landline	
ID	RegAdd
LLN35	Add17
LLN26	Ad45
LLN46	Add24
LLN90	Add67
LLN81	Add98
LLN79	Add78

Address
ID
Add17
Ad45
Add24
Add67
Add98
Add78
Add999

For each $v \in V = \{\text{Person, Landline, Address}\}$, the set $I(v)$ is given by the points of the column in the corresponding table. For example, $I(\text{Person}) = \{\text{P1, P2, P3}\}$. For each arrow $a \in A = \{\text{Contact, RegAdd, HomeAdd}\}$, the function $I(a) : I(v) \to I(w)$, is given by the foreign key associated to the arrow in $I(v)$. For example, $I(\text{Contact})(\text{P3}) = \text{LLN79}$. Also, note that the path equivalence is adhered to by the instance, i.e $I(HomeAdd)(i) = I(RegAdd \circ Contact)(i)$ for each $i \in I(\text{Person})$.

2.1.4 Translations of schemas

Example 2.1.12 A translation between schemas $F : \mathcal{C} \to \mathcal{D}$ is a mapping that takes objects in $\mathcal{C}$ to objects in $\mathcal{D}$ and the same is done for arrows in such a way that the mapping of objects and arrows, the sources, targets and path equivalences are respected. We use the following translation to give an outline of the paradigm.



Let the schema on the left be $\mathcal{C}$ and the schema on the right be $\mathcal{D}$. Both 'Male' and 'Female' are mapped to 'Person' in $\mathcal{D}$, and the other vertices are mapped to the vertices marked with the same name. Since the translation must respect sources, targets and path equivalences, there is only one possible way to map the arrows. A formal definition of a translation is given below.

Definition 2.1.13 Let $G = (V_G, A_G, s_G, t_G)$ and $H = (V_H, A_H, s_H, t_H)$ be graphs, and let $\mathcal{C} = (G, \simeq_C)$ and $\mathcal{D} = (H, \simeq_D)$ be categorical schemas. A translation $F : \mathcal{C} \to \mathcal{D}$ contains

1. a function $V_F : V_G \to V_H$
2. a function $A_F : A_G \to Path_H$

such that the functions are subject to the following restrictions:

(a) A_F must preserve sources and targets meaning the following diagrams must commute

$$\begin{array}{ccc} \mathrm{A_G} & \xrightarrow{\mathrm{A_F}} & \mathrm{Path_H} \\ {\scriptstyle \mathrm{s_G}}\downarrow & & \downarrow{\scriptstyle \mathrm{s_H}} \\ \mathrm{V_G} & \xrightarrow[\mathrm{V_F}]{} & \mathrm{V_H} \end{array} \qquad \begin{array}{ccc} \mathrm{A_G} & \xrightarrow{\mathrm{A_F}} & \mathrm{Path_H} \\ {\scriptstyle \mathrm{t_G}}\downarrow & & \downarrow{\scriptstyle \mathrm{t_H}} \\ \mathrm{V_G} & \xrightarrow[\mathrm{V_F}]{} & \mathrm{V_H} \end{array}$$

(b) A_F must preserves path equivalences, i.e suppose we are given lengths $m, n \in N$ and paths $p = id_{v_0} f_1 f_2 ... fm$ and $q = id_{v_0} g_1 g_2 ... g_n$ in G. Let $v_0 = V_F(v_0)$ and for each $i \leq m$ (resp. $j \leq$), let $f'_i = A_F(f_i)$ (resp. $g'_j = A_F(g_j)$), and let $p' = id_{v'_0} f'_1 f'_2 ... f'_m$ (resp. $q' = id_{v'_0} g'_1 g'_2 ... g'_n$). If $p \simeq_{\mathcal{C}} q$ then $p' \simeq_{\mathcal{C}} q'$.

Definition 2.1.14 Two translation $F, F' : \mathcal{C} \to \mathcal{D}$ are said to be identical if $V_F = V_{F'}$ and for every arrow $f \in \mathcal{C}$, there is a path equivalence $A_F(f) \simeq_{\mathcal{D}} A_{F'}(f)$.

2.1.5 The equivalence Sch≃Cat

Definition 2.1.15 A category $\mathcal{C}$ is called a small category if ob($\mathcal{C}$) and Hom($\mathcal{C}$) are sets and not proper classes. The category **Cat**, the category of small categories, contains all small categories as its objects and all functors between small categories as its morphisms. Similarly, using the definition of categorical schemas and translations, we define **Sch**, the category of all schemas whose objects are all schemas and whose morphisms are all translations between schemas.

To prove that there is an equivalence between **Sch** and **Sch**, [Spi12] describes two functors between the two categories in opposing directions and proves that the two functors are mutual inverse equivalences [See def 1.3.15 of [Lei14]]. This proves that the equivalence exists.

Definition 2.1.16 Given a category $\mathcal{C}$, a *congruence relation* R on $\mathcal{C}$ consists of an equivalance relation $R_{X,Y}$ on Hom(X, Y) for each pair of objects in $X, Y \in \mathcal{C}$, such that equivalence relations respect the composition of morphisms in $\mathcal{C}$. Specifically, if $f_1, f_2 : X \to Y$ are related in Hom(X, Y) and $g_1, g_2 : Y \to Z$ are related in Hom(Y, Z) then $g_1 \circ f_1$ and $g_2 \circ f_2$ are related in Hom(Y, Z).
Given a congruence relation R on $\mathcal{C}$, we can define a new category $\mathcal{C}/R$, called the *quotient category*, whose objects are those of $\mathcal{C}$ and whose morphisms are

the equivalence classes of morphisms in $\mathcal{C}$, meaning for each $X, Y \in \mathcal{C}$

$$\mathrm{Hom}_{\mathcal{C}/R}(X, Y) = \mathrm{Hom}_{\mathcal{C}}(X/Y)/R_{X,Y}$$

The composition of equivalence classes as morphisms in $\mathcal{C}/R$ is given as follows: if $f : X \to Y$ and $g : Y \to Z$ in $\mathcal{C}$, the corresponding morphisms in $\mathcal{C}/R$, $[f] : X \to Y$ and $[g] : Y \to Z$ are composed as $[g][f] = [gf]$.

Definition 2.1.17 Given a graph $G = (V, A, s, t)$, the category **Free**(G), called the *free category* on G, is defined to be the category whose objects are the vertices in G and whose morphisms are the paths between all pairs of vertices in G. The identity morphism is given by the trivial path from a vertex to itself and composition of morphisms is given by concatenation of paths.

Construction 2.1.18 (From schemas to categories). A functor $L : \mathbf{Sch} \to \mathbf{Cat}$ is constructed as follows. Let $C = (G, \simeq_{\mathcal{C}})$ be schema, where $G = (A, V, s, t)$. $L(\mathcal{C})$ is defined to be the quotient of the free category $\mathcal{C}'$ described by the equivalence relation $\simeq_{\mathcal{C}}$. For each translation, $F : \mathcal{C} \to \mathcal{D}$, there is an induced functor on free categories $F' : \mathcal{C}' \to \mathcal{D}'$ whose objects function is the same as V_F and for each arrow $a \in A$ as a morphism in $\mathrm{Hom}_{\mathcal{C}'}(s(a), t(a))$, $F'(a) \in \mathcal{D}'$ is given by the morphism corresponding to the path $A_F(a)$. The preservation of path equivalences ensures that the $L(F) : L(\mathcal{C}) \to L(\mathcal{D})$ is a functor on the quotient categories. It is clear that L preserves composition and that identity maps in $L(\mathcal{C})$ are mapped to identity maps in $L(\mathcal{D})$.

Example 2.1.19 Consider the schema $\mathcal{C}$ described in example 1.11. A description of $L(\mathcal{C})$ is given here to explain how L works. The set of morphisms between 'Male' and 'Address' in $L(\mathcal{C})$ is given by $\mathrm{Hom}_{L(\mathcal{C})}(\mathrm{Male}, \mathrm{Address}) = \{[H_m]\}$, where the equivalence class $[H_m] = \{H_m, R \circ C_m\}$ (since $H_m \simeq R \circ C_m$). All equivalent morphism get grouped into one set, and the set itself is thought of as a morphism. A morphism in quotient category doesn't map any elements from its source to its target, and simply exists as an entity. The table given on the next page describes the morphisms between each objects in $L(\mathcal{C})$. X is represented by the column and Y is represented by the row. Appropriate abbreviations have been made.

Hom(X, Y) in $L(\mathcal{C})$				
	Male	Landline	Female	Address
Male	$\{[id_m]\}$	$\{[C_m]\}$	$\varnothing$	$\{[H_m]\}$
Landline	$\varnothing$	$\{[id_L]\}$	$\varnothing$	$\{[R]\}$
Female	$\varnothing$	$\{[C_f]\}$	$\{[id_f]\}$	$\{[H_f]\}$
Address	$\varnothing$	$\varnothing$	$\varnothing$	$\{[id_A]\}$

With this clarity, it is easy to see how L would act on a translation.

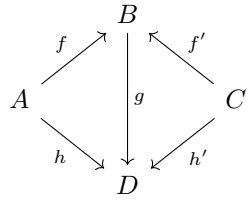
Construction 2.1.19 (From categories to schemas). A functor $R : \mathbf{Cat} \to \mathbf{Sch}$ is constructed as follows. Given a small category $\mathcal{C}$, a graph $G = (\mathrm{Ob}(\mathcal{C}), \mathrm{Hom}(\mathcal{C}), s, t)$ can be constructed, where s and t are the source and target functions on $\mathrm{Hom}(\mathcal{C})$. An equivalence relation is defined as follows: or all $f, g \in \mathrm{Hom}(\mathcal{C})$ with $t(f) = s(g)$ we put

$$fg \simeq (g \circ f).$$

It is important to note here that fg is in a path of length 2 in G whereas $g \circ f$ is an arrow, i.e path of length 1 in G. We define $R(\mathcal{C})$ to be the schema $(G, \simeq)$. A functor $F : \mathcal{C} \to \mathcal{D}$ induces a translation $R(F) : R(\mathcal{C}) \to R(\mathcal{D})$. Both restrictions on $V_{R(F)}$ and $A_{R(F)}$ are satisfied naturally.

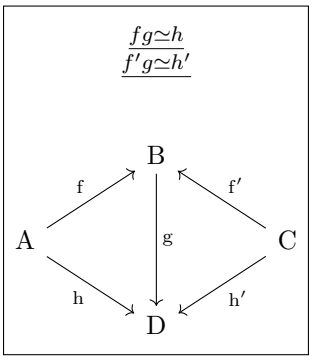
Example 2.1.20 In this example, we calculate the image of the small category $\mathcal{D}$ given below to learn how R works.

$\mathcal{D} :=$



Assume that the diagram commutes. Let $R(\mathcal{D}) = (G, \simeq_G)$. It is clear that G here is given by $(\{A, B, C, D\}, \{f, f', g, h, h'\}, s, t)$ where s and t are the appropriate source and target functions. Note here $g \circ f$ and $g \circ f'$ are not listed in the list of arrows because the diagram is said to commute. $\simeq$, by definition, gives $fg \simeq h$ and $f'g \simeq h'$. Note that $\simeq$ satisfies the requirements listed in the path equivalence relationship trivially. Therefore, the schema given below

is obtained



The schema obtained is practically the same as $\mathcal{C}$ from example (add no.), with the only difference being the names objects and arrows. This example is given as an attempt to show that for any small category $\mathcal{D}$, there is a clear isomorphism of categories (See (add leinster)) $\mathcal{D} \cong L(R(\mathcal{D}))$. Note that this implies that there exists a natural isomorphism $n : \mathrm{id}_{\mathbf{Cat}} \to L \circ R$.

Theorem 2.1.21 The functors

$$L : \mathbf{Sch} \leftrightarrows \mathbf{Cat} : \mathrm{R}$$

are mutually inverse equivalences.

Proof It is clear from example (add no.) that there must exist a natural isomorphism $n : \mathrm{id}_{\mathbf{Cat}} \to L \circ R$; i.e. for any category $\mathcal{D}$, there is an isomorphism $\mathcal{D} \cong L(R(\mathcal{D}))$. This means that L is essentially surjective. (Leinster, add theorem). We first show that L is fully faithful. Choose schemas X and Y, and suppose $X = (A_X, V_X, s_X, t_X)$; we must show that the function $L_1 : \mathrm{Hom}_{\mathbf{Sch}}(X, Y) \to \mathrm{Hom}_{\mathbf{Cat}}(LX, LY)$ is a bijection.

To see that L_1 is injective, take any two translations $F, G : X \to Y$ and assume $LF = LG$. Note that by our assumption, $V_F = V_G$ must be true. Also we can take any arrow $f \in X$ and say that $LF([f]) = LG([f])$ where $[f]$ is the equivalence class of f under $\simeq_X$ (follows from assumption). This means $[A_F(f)] = [A_G(f)]$. With this we have $A_F(f) \simeq_Y A_G(f)$ as desired, because this concludes F and G are identical (Definition (add number)).

To see that it is surjective, take any functor $G : L(X) \to L(Y)$, we need to define a translation $F : X \to Y$ such that $L_1(F) = G$. Define F on vertices of X as G is defined on objects of LX. Define F on arrows of X via the function $A_X \to \mathrm{Path}_X \to \mathrm{Hom}(LX) \xrightarrow{G} \mathrm{Hom}(LY)$, and choose a representative for its equivalence class from $\mathrm{Path}_{L(Y)}$. Two equivalent paths in X maps to the

same element i.e their equivalent class in $\mathrm{Hom}(LX)$ b, so F preserves path equivalence. Therefore, we conclude L is an equivalence.

Now we show, with similar processes, that R is fully faithful, this will conclude that it is an inverse to L. Reset variables and choose small categories X and Y arbitrarily, and let $R_1 : \mathrm{Hom}_{\mathbf{Cat}}(X, Y) \to \mathrm{Hom}_{\mathbf{Sch}}(RX, RY)$. We need to show R_1 is a bijection.

To see that it is injective, take any two functors $F, G : X \to Y$ and assume $RX = RY$. By our assumption, $\mathrm{Ob}(X) \xrightarrow{F} \mathrm{Ob}(Y)$ must be the same function as $\mathrm{Ob}(X) \xrightarrow{G} \mathrm{Ob}(Y)$. Also, we can take any morphism $f \in RX$ and say that $A_{RF}(f) \simeq_{RY} A_{RG}(f)$ (true by assumption). This implies $F(f) = G(f)$, and hence $F = G$. Therefore, R_1 is injective.

We can obtain a functor $F : X \to Y$ given $G : RX \to RY$, by defining F's map on its objects to be the same as V_G and defining its map on morphisms to be given by $\mathrm{Mor}(X) \to A_{RX} \xrightarrow{G} \mathrm{Path}_{RY}$. □

Now a rigorous equivalence has been established between the database model and category theory. A list of equivalent definition is given below. Equivalent terms will be used interchangeably in the rest of this paper.

Equivalent definitions in Database Model and Category Theory	
Database Model Concept	Categorical Interpretation
Database Schema, $\mathcal{C}$	Category, $\mathcal{C}$
Table $T \in \mathcal{C}$	Object $T \in \mathcal{C}$
Foreign key in table T	Morphism $f \in \mathrm{Hom}(T, \cdot)$
ID column in table T	Identity morphism, $\mathrm{id}_T : T \to T$
Sequence of foreign keys	Composition of morphisms
Equivalence of paths	Commutative diagram
Database instance on $\mathcal{C}$, I	**Set**-valued functor $I : \mathcal{C} \to \mathbf{Set}$
Translation $F : \mathcal{C} \to \mathcal{D}$	Functor $F : \mathcal{C} \to \mathcal{D}$

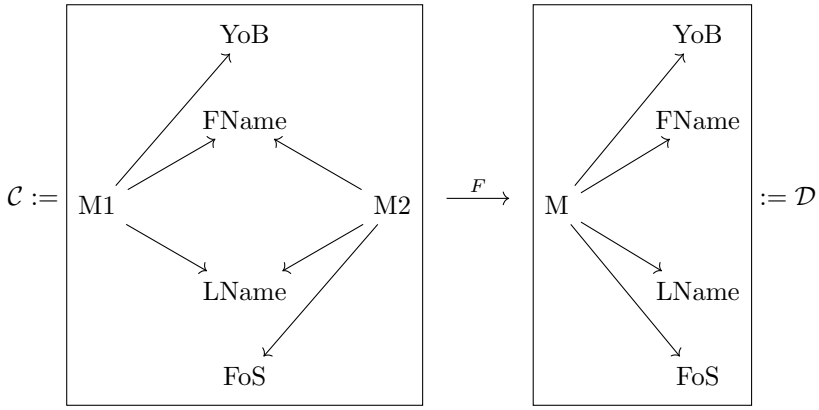
2.2 Data migration functors

In this section, we discuss the data migration functors given by [Spi10]. The function of data migration functors is to convert $\mathcal{C}$-instances into $\mathcal{D}$-instances, or vice versa. Before we proceed, we note that the collection of instances on a category can be described as a functor category between the said category and **Set**.

Definition 2.2.1 Let $\mathcal{C}$ be a schema. $\mathcal{C}$–**Inst** is defined as the category whose objects are **Set**-valued instances on $\mathcal{C}$, i.e. functors $I : \mathcal{C}$–**Set** and whose morphisms are natural transformations between instances. [1]

2.2.1 Informal description of data migration

Example 2.2.2 The data migration functors are described with the help of an example first. The construction of the functors through categorical concepts will be described once their action on instances has been understood. Consider the schema translation



This diagram represents a translation between schemas which contains information about mathematicians ('M1', 'M2' and 'M'), their names ('FName' and 'LName'), year of birth (YoB) and fields of study (FoS). The schema translation is depicted through the diagram as clearly as possible and makes it clear that sources and targets are respected by the translation. 'M1' and 'M2' are mapped to 'M' in $\mathcal{D}$ and all the other vertices are mapped to their equivalents in $\mathcal{D}$ with the same name.

[1] In this paper, we only concern ourselves with **Set**-valued instances of schemas. General $\mathbb{S}$-valued instances are mentioned in [Spivak] but are not within the scope of this project.

1. Given a schema translation as the one above, the **pullback functor**, Δ_F : $\mathcal{D}$–**Inst** $\to$ $\mathcal{C}$–**Inst**, converts each $\mathcal{D}$-instance into a $\mathcal{C}$-instance. Δ_F projects an instance on $\mathcal{D}$ onto $\mathcal{C}$ through F. For example, consider the instance I on $\mathcal{D}$ given by

M				
ID	DoB	FName	LName	FoS
A	1906	Kurt	Godel	Logic
B	1862	David	Hilbert	Everything
C	1872	Bertrand	Russell	Logic

The tables for 'YoB', 'FName', 'LName' and 'FoS' only contain ID columns. They have been omitted for brevity. The image of this instance under Δ_F is given by the tables below

M1			
ID	DoB	FName	LName
A1	1906	Kurt	Godel
A2	1862	David	Hilbert
A3	1872	Bertrand	Russell

M2			
ID	FName	LName	FoS
101	Kurt	Godel	Logic
102	David	Hilbert	Everything
103	Bertrand	Russell	Logic

The contents of 'M1' and 'M2' are determined with the translation. In this example, 'M1' and 'M2' are projections of 'M' only through our choice of F. The key observation here is that Δ_F is a canonical construction given a translation $F : \mathcal{C} \to \mathcal{D}$ and an instance I on $\mathcal{D}$. The instance on $\mathcal{C}$ is given by selecting the relevant columns for both 'M1' and 'M2' and presenting them with their elements (their elements are the same by choice).

2. The **right pushforward functor**, written as $\Pi_F : \mathcal{C}$–**Inst** $\to$ $\mathcal{D}$–**Inst**, takes a $\mathcal{C}$-instance and converts it into a $\mathcal{D}$-instance. Note that Δ_F and Π_F point in opposite direction. Given the following instance on $\mathcal{C}$

M1			
ID	YoB	FName	LName
11	1862	David	Hilbert
12	1845	Georg	Cantor
13	1909	Saunders	MacLane

M2			
ID	FName	LName	FoS
101	Kurt	Godel	Logic
102	David	Hilbert	Everything
103	Bertrand	Russell	Logic
104	Georg	Cantor	Set Theory

is converted into $\Pi_F(I)$ which is given by

M				
ID	DoB	FName	LName	FoS
B	1862	David	Hilbert	Everything
C	1845	Georg	Cantor	Set Theory

The key observation here is that Π_F has produced a join.

3. The **left pushforward functor**, written as $\Sigma_F : \mathcal{C}$–**Inst** $\to \mathcal{D}$–**Inst**, takes a $\mathcal{C}$-instance and converts it into a $\mathcal{D}$-instance. The image of the $\mathcal{C}$-instance mentioned earlier under Σ_F is

M1				
ID	DoB	FName	LName	FoS
11	1906	Kurt	Godel	11.FoS
12	1862	David	Hilbert	12.FoS
13	1909	Saunders	MacLane	13.FoS
101	101.DoB	Kurt	Godel	Logic
102	102.DoB	David	Hilbert	Everything
103	103.DoB	Bertrand	Russell	Logic
104	104.DoB	Georg	Cantor	Set Theory

The key observation here is that Σ_F has produced a meet.

2.2.2 Formal description of data migration

The migration functors may seem simple and uncomplicated through their informal description. The pullback functor is, in fact, conceptually the simplest. However the pushforward are derived through fairly complicated categorical constructions. The pullback functor Δ_F, being the simplest, is described first.

Definition 2.2.3 Let $F : \mathcal{C} \to \mathcal{D}$ be a translation between schemas. The **pullback functor** $\Delta_F : \mathcal{D}$–**Inst** $\to \mathcal{C}$–**Inst** maps each $\mathcal{D}$-instance $I \mapsto I \circ F$, converting it into a $\mathcal{C}$-instance and maps a natural transformation $\alpha : I \to J$ in $\mathcal{D}$–**Inst** to the natural transformation $m \circ F$ in $\mathcal{C}$–**Inst**.

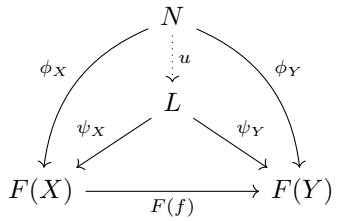
Pushforward functors

The left and right pushforward functors, Π_F and Σ_F, are constructed as the left and right Kan extensions (see [Wik20]) (respectively) of Δ_F. The left and right pushforward functors are in fact the left and right adjoint (respectively)

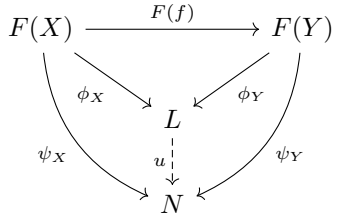
of the pullback functor. Why they are adjoints is not within the scope of the project. The readers familiar with MacLane book can look at ([ML13], X.3.2).

Definition 2.2.4 Let $\mathcal{C}$ and $\mathcal{J}$ be categories. [[Lei14]]

- A **diagram** of **shape** $\mathcal{J}$ in $\mathcal{C}$ is functor $F : \mathcal{J} \to \mathcal{C}$. The category $\mathcal{J}$ is also referred to as the index category.
- Let $F : \mathcal{J} \to \mathcal{C}$ be a diagram. A **cone** to F is an object N in $\mathcal{C}$ together with a family $\psi_X : N \to F(X)$ of morphisms indexed by the objects X of $\mathcal{J}$, such that for every morphism $f : X \to Y$ in J, we have $F(f) \circ \psi_X = \psi_Y$.
- A **limit** of the diagram $F : \mathcal{J} \to \mathcal{C}$ is a cone (L, ϕ) to F such that for any other cone (N, ψ) to F there exists a unique morphism $u : N \to L$ such that $\phi_X \circ u = \psi_X$ for all X in $\mathcal{J}$.



- Let $F : \mathcal{J} \to \mathcal{C}$ be a diagram. A **co-cone** of F is an object N in $\mathcal{C}$ together with a family $\psi_X : F(X) \to N$ of morphisms indexed by the objects X of $\mathcal{J}$, such that for every morphism $f : X \to Y$ in J, we have $\psi_Y \circ F(f) = \psi_X$.
- A **co-limit** of the diagram $F : \mathcal{J} \to \mathcal{C}$ is a cone (L, ϕ) of F such that for any other cone (N, ψ) of F there exists a unique morphism $u : L \to N$ such that $u \circ \phi_X = \psi_X$ for all X in $\mathcal{J}$.



If you are seeking an elaborate understanding of these definitions, a good reference is [Lei14]. For the sake of brevity, example of these concepts are not given

Definition 2.2.5 Given categories and functors

$$\begin{array}{ccc} & & \mathcal{B} \\ & & \downarrow Q \\ \mathcal{A} & \xrightarrow[P]{} & \mathcal{C} \end{array}$$

the **comma category** $(P \downarrow Q)$ (often written as $(P \Rightarrow Q)$) is the category defined as follows:

- objects as triples (A, h, B) with $A \in \mathcal{A}$, $B \in \mathcal{B}$ and $h : P(A) \to Q(B)$ in $\mathcal{C}$;
- maps $(A, h, B) \to (A', h', B)$ are pairs $(f : A \to A', g : B \to B')$ of maps such that the square

$$\begin{array}{ccc} P(A) & \xrightarrow{P(f)} & P(A') \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h'} \\ Q(B) & \xrightarrow[Q(g)]{} & Q(B') \end{array}$$

  commutes.

Let's now look at the schema translation functor $F : \mathcal{C} \to \mathcal{D}$. Given $D \in \mathrm{Ob}(\mathcal{D})$, we can construct a comma category in the following way. [2]

$$\begin{array}{ccc} & & \mathcal{C} \\ & & \downarrow F \\ \mathbf{1} & \xrightarrow[D]{} & \mathcal{D} \end{array}$$

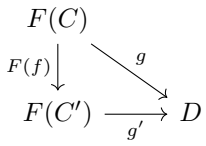
Here the object are pairs $(C \in \mathcal{C}, g : D \to F(C))$. A map $(C, g) \to (C', g')$ in $(D \downarrow F)$ is a map $f : C \to C'$ in $\mathcal{C}$ making the triangle

$$\begin{array}{ccc} D & \xrightarrow{g} & F(C) \\ & \searrow^{g'} & \downarrow F(f) \\ & & F(C') \end{array}$$

commute.

It is important to note here that these commutative diagrams for each C, C' are sections of a cone. Similarly, we can construct the category $(F \downarrow D)$. Here the object are pairs $(C \in \mathcal{C}, g : F(C) \to D)$. A map $(C, g) \to (C', g')$ in $(F \downarrow D)$ is a map $f : C \to C'$ in $\mathcal{C}$ making the triangle

[2] Here, $\mathbf{1}$ is a category with a single object and its single identity morphism. We use $'D'$ as a functor from $\mathbf{1}$ to $\mathcal{D}$ because a functor from $\mathbf{1}$ can be thought of as choosing an object in the target category. We label the functor with chosen object. In this case, it is the given D.



commute. Again, note here that these commutative diagrams for each C, C' are sections of a cocone. We define, π^F and pi_F as forgetful functors such that for each $D \in Ob(\mathcal{D})$

$$\pi^F(D) : (D \downarrow F) \to C$$
$$\pi_F(D) : (F \downarrow D) \to C,$$

both of which are given by $(C, g) \mapsto C$. We consider the category $(D \downarrow F)$ together with the functor $\pi^F(D) : (D \downarrow F) \to C$ as an object of the slice category $\mathbf{Cat}/\mathcal{C}$. Similarly, We consider the category $F \downarrow D)$ together with the functor $\pi_F(D) : (F \downarrow D) \to C$ as an object of $\mathbf{Cat}/\mathcal{C}$.

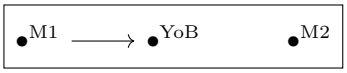
Now we can describe π^F and π_F as

$$\pi^F : \mathcal{D}^{op} \to \mathbf{Cat}/\mathcal{C}$$
$$\pi_F : \mathcal{D} \to \mathbf{Cat}/\mathcal{C}$$

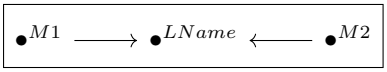
Definition 2.2.6 Let $F : \mathcal{C} \to \mathcal{D}$ be a functor schemas. The functor $\pi^F : \mathcal{D}^{op} \to \mathbf{Cat}/\mathcal{C}$ is called the **F-right schema functor** on $\mathcal{D}$ and the functor $\pi_F : \mathcal{D} \to \mathbf{Cat}/\mathcal{C}$ is called the **F-left schema functor** on $\mathcal{D}$. For any object $D \in \mathrm{Ob}(\mathcal{D})$ the category $(D \downarrow F)$ is called the **schema F-right** of D and the category $(F \downarrow D)$ is called the **schema F-left** of D.

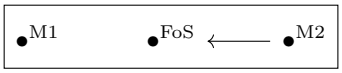
Example 2.2.7 Let us calculate $\pi_F : \mathcal{D} \to \mathbf{Cat}/\mathcal{C}$ in the case defined in example 2.2. The image of each object in $\mathcal{D}$ is given below

$\pi_F(\mathrm{M}) =$ 

$\pi_F(\mathrm{YoB}) =$ 

$\pi_F(\mathrm{FName}) =$ 

$\pi_F(\mathrm{LName}) =$ 

$\pi_F(\mathrm{FoS}) =$ 

$\pi_F(D)$ is simply obtained by looking at the objects X in $\mathcal{D}$ such that there morphisms $X \to D$ and then constructing a sub-category of all the preimages of all such X (Here X is said to be on left of D). Let us now define what $\pi_F : \mathcal{D}^{op} \to \mathbf{Cat}/\mathcal{C}$. On the each of the four objects YoB, FName, LName, and FoS, π_F yields the subcategory of $\mathcal{C}$ consisting of that object alone: for example $\pi_F(\text{YoB}) = (\{\text{YoB}\} \to \mathcal{C})$. This is valid because there are objects to the right of the ones listed above (for example there is no morphism $YoB \to X$ for some X in $\mathcal{D}$ which isn't YoB itself). On the object M, we get the whole of $\mathcal{C}$.

Definition 2.2.8 (*Right Pushforward*) Given a morphism of schemas $F : \mathcal{C} \to \mathcal{D}$, the left adjoint to $\Delta_F : \mathcal{D}$–**Set** $\to \mathcal{C}$–**Set**, the associated data pull-back functor, is denoted

$$\Pi_F : \mathcal{C}\text{–}\mathbf{Set} \to \mathcal{D}\text{–}\mathbf{Set},$$

and is defined as follows: given a $\mathcal{C}$–instance $I : \mathcal{C} \to \mathbf{Set}$ define $\Pi_F(I)$ on an object $D \in \text{Ob}(\mathcal{D})$

$$\Pi_F(I)(D) := \text{colim}_{(F \downarrow D)}(I \circ \pi_F(D))$$

which is the colimit of the functor $(F \downarrow D) \xrightarrow{\pi_F(D)} \mathcal{C} \xrightarrow{I} \mathbf{Set}$. Given a map $g : D \to D'$ in $\mathcal{D}$ one obtains a map $\Pi_F(I)(g) : \Pi_F(I)(D) \to \Pi_F(I)(D')$ by the universal property of limits.

Definition 2.2.9 (*Left Pushforward*) Given a morphism of schemas $F : \mathcal{C} \to \mathcal{D}$, the right adjoint to $\Delta_F : \mathcal{D}$–**Set** $\to \mathcal{C}$–**Set**, the associated data pull-back functor, is denoted

$$\Sigma_F : \mathcal{C}\text{–}\mathbf{Set} \to \mathcal{D}\text{–}\mathbf{Set},$$

and is defined as follows: given a $\mathcal{C}$–instance $I : \mathcal{C} \to \mathbf{Set}$ define $\Sigma_F(I)$ on an object $D \in \text{Ob}(\mathcal{D})$

$$\Sigma_F(I)(D) := \lim_{(D \downarrow F)}(I \circ \pi^F(D))$$

which is the limit of the functor $(F \downarrow D) \xrightarrow{\pi_F(D)} \mathcal{C} \xrightarrow{I} \mathbf{Set}$. Given a map $g : D \to D'$ in $\mathcal{D}$ one obtains a map $\Sigma_F\gamma(g) : \Sigma_F\gamma(D) \to \Sigma_F\gamma(D')$ by the universal property of limits.

Explanation 2.2.10 For the readers unfamiliar with Kan extensions, even thought the pushforward functors haven't been defined directly using the definition of left and right Kan extension, they are exactly that. All of this seems

very complicated because of all the new concepts and symbols but the idea is simple. We want to convert a $\mathcal{C}$-instance I in a $\mathcal{D}$-instance. Π_F does this for each $D \in \mathcal{D}$ by taking objects which are to the left of D (as described in exmpale 2.2.7), taking the set of rows assigned to those objects by I and assigning $\Pi_F(I)$ the colimit of all these sets. Σ_F does this for each $D \in mathcal D$ by taking objects which are to the right of D (as described in exmpale 2.6), taking the set of rows assigned to those objects by I and assigning $\Sigma_F(I)$ the limit of all these sets.

$\Sigma_F(I)$ applied to YoB, FName, LName, and FoS is just I applied to the object's equivalent in $\mathcal{C}$ and $\Sigma_F(I)$ applied to M is the limit of the diagram I meaning a row in $\Sigma_F(I)$ is a row in $M1$ and a row in $M2$ such that their FName and LName match. A joint is obtained!

For all D, the functor from $\pi_F(D)$ to **Set** is obtained by composing its inclusion into $\mathcal{C}$ with I. To compute $\Pi_F(I)$, we calculate colimit of each of the following functors: $\gamma(\text{M1}) \vee \gamma(\text{M2}); \gamma(\text{SSN}) \vee \gamma(\text{M2}); \gamma(\text{First}); \gamma(\text{Last}); \gamma(\text{M1}) \vee \gamma(\text{Salary})$ and a meet is obtained!

2.3 Data types and filtering

So far, database have only been expressed as collections of connected datapoints. Data points are arranged in the form of tables and connected through foreign keys. The concept of datatypes has not been included in the model we have described so far. Data is expressed through datatypes, such as float, string, integeres etc. Not only do datatypes provide a way to represent data in consistent way, they can also be used to represent value in a meaningful way, for example through integeres. Through this section, the relationship between data and datatypes as presented by [Spi12] is explored. Categorical concepts are used to provide a connection between the established database model and programming languages. First, we see how datatypes can be attached to schemas and the morphisms on type signatures are defined. The formulation described will be used to provide a new perspective data filtering.

Example 2.3.1 One way to interpret the phrase 'a set of integers' is by its usual sense. We can generalise the interpretation to a pair (X, f) such that $f : X \to \mathbb{Z}$. Note that X and f may vary but $\mathbb{Z}$ is always the same. This can categorical formulated as

1. as a database instance $I : \mathcal{C} \to$ **Set** on the schema

$$\mathcal{C} := \boxed{\bullet^X \longrightarrow \bullet^{\mathbb{Z}}}$$

such that $\bullet^{\mathbb{Z}}$ is fixed as $I(\bullet^{\mathbb{Z}}) = \mathbb{Z}$; or

2. as a database instance $J : \mathcal{D} \to \mathbf{Set}$ on the schema

$$\mathcal{D} := \boxed{\bullet^X}$$

equipped with a natural transformation $f : J \to \mathbb{Z}$. Here $\mathbb{Z}$ means the functor $\mathcal{D} \to \mathbf{Set}$ defined by $D(\bullet^{\mathbb{Z}}) = \mathbb{Z}$.

Example 2.3.2 A mathematical relationship between two columns of a table can established using categorical language. For example, let a column c called "number of children" (integer) be related to column c', called "education fees" (as a dollar-figure) by a mathematical function $c' = f(c)$, say

$$c'(x) = f(c(x)) := \$9000 \times t(x)$$

for each $x \in X$. This situation could be categorically represented as follows. We can interpret this as an collection of instance $J : \mathcal{D} \to \mathbf{Set}$ on a schema given below

$$\mathcal{D} := \boxed{\begin{array}{ccc} X & \xrightarrow{f} & Y \\ & \searrow^{c'} & \downarrow{\scriptstyle c} \\ c' = fc & & Z \end{array}}$$

Here is an instance on the $\mathcal{D}$

X		
ID	t	d
A1	2	\$18000
A2	5	\$45000
A3	6	\$54000

We note that a natural transformation $n : J \to P$, where P is typing instance, can be used to enforce the fact that $c' = f(c)$ and that c' has dollar-figure datatype. This observation leads to the following definition.

Definition 2.3.4 Let $\mathcal{C}$ be a schema and let $P \in \mathrm{Ob}(\mathcal{C})$ be an instance. The category of P-typed instances on $\mathcal{C}$, denoted $\mathcal{C}$–Inst$/P$, is defined to be the slice category of instances over P. In other words, a P-typed instance on $\mathcal{C}$ is a pair (I, τ) where I is an instance and $\tau : I \to P$ is a natural transformation; and a morphism of P-typed instances is a commutative triangle. Any instance $P \in \mathrm{Ob}(\mathcal{C}$–**Inst**$)$ is refered to as a typing instance if our plan is to consider the category $\mathcal{C}$–Inst/P, the category of P-typed instances.

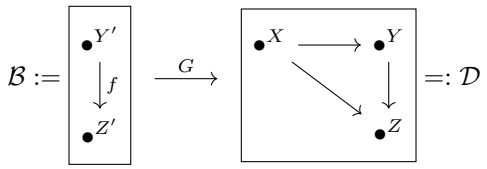
Remark 2.3.5 Fix a schema $\mathcal{C}$ and a category $\mathcal{S}$ and let $\mathcal{E} := \mathcal{C}$–$\mathrm{Inst}_{\mathbb{S}}$ denote the category of $\mathbb{S}$-valued instances on $\mathcal{C}$.

Construction 2.3.6 Given a **Type** category of types in a programming language and a $V \in \mathbf{Type} - \mathbf{Inst}_{\mathbb{S}}$, typing information can be assigned to our database schema $\mathcal{C}$ with a fragment of **Type**, if the fragment is given by a functor $\mathcal{B} \xrightarrow{F} \mathbf{Type}$ and $\mathcal{B}$ is associated to the schema $\mathcal{B} \xrightarrow{G} \mathcal{C}$ via a functor,

$$\mathbb{S} \xleftarrow{V} \mathbf{Type} \xleftarrow{F} \mathcal{B} \xrightarrow{G} \mathcal{C}$$

Here, $P := \Pi_G \circ \Delta_F(V)$ is the implied typing instance. The sequence $(\mathcal{B}, F, G)$ is called the typing auxiliary in this setup.

Example 2.3.7 Applying this construction on example (add number), a typing auxiliary is described. Let $\mathcal{B}$ be one-arrow category drawn below, and let $G : \mathcal{B} \to \mathcal{D}$ be the suggested functor



Let $F : \mathcal{B} \to \mathbf{Type}$ be a functor such that $F(Y') =$`Int`, the type of integers, $F(Z') =$`Dollar`, the type of dollar figures, $F(r')$ to the function that multiplies an integer by 9000.

With $V : \mathbf{Type} \to \mathbb{S}$ like in the Construction, the implied typing instance $P := \Pi_{\mathcal{G}} \circ \triangle_F(V) : \mathcal{D} \to \mathbb{S}$ has

$$P(\mathtt{X}) = \mathtt{Int} \times \mathtt{Dollar} \quad P(\mathtt{Y}) = \mathtt{Int} \quad P(\mathtt{Z}) = \mathtt{Dollar}$$

and $P(r) : P(\mathtt{Y}) \to P(\mathtt{Z})$ is indeed the multiplication by 9000 map.

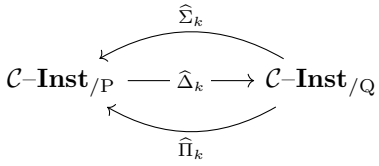
Now a P-typed instance $\tau : I \to P$ is exactly what we want. For each of

X, Y, Z, it is a set with a map to the given data type, and the naturality of τ ensures the properties described in Example 5.1.2. The value of a row in column c' of $J(X)$ has to be 9000 times the value of the cell in column c.

In other words, it ensures that for any row in $J(X)$, the value of the cell in column c' will be 9000 times the value of the cell in column c.

2.3.1 Morphisms of type signatures

In this subsection, a formal description of changing data types of schema instances is given. Given a schema $\mathcal{C}$, typing instances $P, Q \in \mathcal{C} - \mathbf{Inst}$ and a natural transformation $k : P \to Q$, there are induced adjunctions



Definition 2.3.8 Let $\mathcal{C}$ be a schema and $k : P \to Q$ a morphism of typing instances. We refer to the induced functors

$$\widehat{\Sigma}_k, \widehat{\Pi}_k : \mathcal{C}\text{–}\mathbf{Inst}_{/\mathrm{P}} \to \mathcal{C}\text{–}\mathbf{Inst}_{/\mathrm{Q}}, \qquad \widehat{\triangle}_k : \mathcal{C}\text{–}\mathbf{Inst}_{/\mathrm{Q}} \to \mathcal{C}\text{–}\mathbf{Inst}_{/\mathrm{P}}$$

as *type-change functors*. To be more specific, $\widehat{\Sigma}_k$ will be called the *left pushforward type-change functor*, $\widehat{\Pi}_k$ will be called the *right pushforward type-change functor*, and $\widehat{\triangle}_k$ will be called the *pullback type-change functor*.

Example 2.3.9 The application of type-change functor $\widehat{\Sigma}_k$ on the case in example (add no.) Let $k : P \to Q$ sends a dollar figure x to `True` if $x \leqslant \$20,000$ and `False` if $x > 20000$. The functor $\widehat{\Sigma}_k$ converts the table on the left to the table on the right below:

X		
ID	t	d
A1	2	\$18000
A2	5	\$45000
A3	6	\$54000

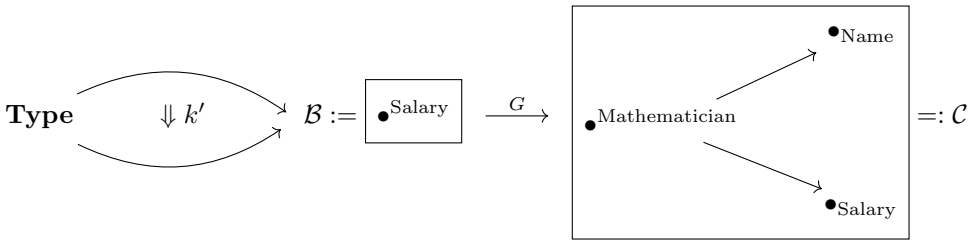
X		
ID	t	d
A1	4	False
A2	7	True
A3	2	True

The other type-change functors, $\widehat{\Pi}_k$ and $\widehat{\triangle}_k$ do not have useful results in the context of this particular example.

2.3.2 Filtering Data

For the final section of this paper, we see how it is possible to filter data through the pullback type-change functor $\widehat{\Delta}$. With the methodology described in this section, it is possible to answer queries such as "return the set of employees with salary less than \$100".

Example 2.3.10 To filter mathematicians by their salaries, in particular showing only those with salaries less than \$100, Suppose we have a schema $\mathcal{C}$ and two typing auxiliaries, $(\mathcal{B}, P, G)$ and $(\mathcal{B}, P, G)$, shown below:



Here we want Q'(Salary) to be the dollar-figure data type, we want P(Salary) to be the subtype given by requiring that a dollar figure x be less than \$100, and we want $k' : P' \to Q'$ to be the inclusion. These typing auxiliaries induce a morphism of typing instances

$$k := \widehat{\Pi}_G(k') : P \to Q, \quad \text{where} P := \widehat{\Pi}_G(P') \text{ and } Q := \widehat{\Pi}_G(Q')$$

Now suppose that $I \in \mathcal{C}$–**Inst** is the Q-typed $\mathcal{C}$-instance. Then $\widehat{\Pi}_k(I)$ is the P-typed instance to the right below

X		
ID	Name	Salary
Em101	Smith	\$65
Em102	Juarez	\$120
Em103	Jones	\$105
Em104	Lee	\$90
Em105	Carlsson	\$80

X		
ID	Name	Salary
Em101	Smith	\$65
Em104	Lee	\$90
Em105	Carlsson	\$80

Therefore, we see that filter can be understood with the data migration framework described in [Spi12].

2.4 Discussion

This chapter reinforces the potential virtue that can be achieved through application of category theory in data analytics. Category theory provides a strong and intuitive model for databases as is seen in this chapter. The applications of this model are ample and the interested reader can seek them [Spi12].

Bibliography

[BPV06] Aaron Bohannon, Benjamin Pierce, and Jeffrey Vaughan. Relational lenses: A language for updatable views. pages 338–347, 01 2006.

[CM10] Gunnar Carlsson and Facundo Mémoli. Characterization, stability and convergence of hierarchical clustering methods. *Journal of Machine Learning Research*, 11(47):1425–1470, 2010.

[CM13] G. Carlsson and F. Mémoli. Classifying clustering schemes. *Foundations of Computational Mathematics*, 13:221–252, 2013.

[CS13] J. Culbertson and K. Sturtz. Bayesian machine learning via category theory. *arXiv: Category Theory*, 2013.

[JD88] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[JD01] Michael Johnson and C. N. G. Dampney. On category theory as a (meta) ontology for information systems research. Association for Computing Machinery, 2001.

[JRW02] Michael Johnson, Robert Rosebrugh, and R. J. Wood. Entity-relationship-attribute designs and sketches. *Theory and Applications of Categories*, 10(1):94–112, 2002.

[Kil] Doruk Kilitcioglu. *Hierarchical Clustering and its Applications*. https://towardsdatascience.com/hierarchical-clustering-and-its-applications-41c1ad4441a6.

[Kle03] Jon M Kleinberg. An impossibility theorem for clustering. In *Advances in neural information processing systems*, pages 463–470, 2003.

[Lei14] T. Leinster. Basic category theory. 2014.

[LW67] G. Lance and W. T. Williams. A general theory of classificatory sorting strategies: 1. hierarchical systems. *Comput. J.*, 9:373–380, 1967.

[Mal] Usman Malik. *Hierarchical Clustering with Python and Scikit-Learn*. https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/.

[Mil18] Bartosz Milewski. *Category theory for programmers*. Blurb, 2018.

[ML13] Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.

[Pri20] Surya Priy. *Clustering in Machine Learning*, 2020. https://www.geeksforgeeks.org/clustering-in-machine-learning/.

[RW92] Robert Rosebrugh and RJ Wood. Relational databases and indexed categories. In *Proceedings of the International Category Theory Meeting 1991, CMS Conference Proceedings*, volume 13, pages 391–407, 1992.

[Spi10] David I. Spivak. Functorial data migration. *CoRR*, abs/1009.1166, 2010.

[Spi12] David I. Spivak. Functorial data migration. *Inf. Comput.*, 217:31–51, 2012.

[Wik20] Wikipedia contributors. Kan extension — Wikipedia, the free encyclopedia, 2020. [Online; accessed 11-September-2020].

[WSSS15] Ryan Wisnesky, David I. Spivak, P. Schultz, and E. Subrahmanian. Functorial data migration: From theory to practice. *ArXiv*, abs/1502.05947, 2015.