

Data Visualization From a Category Theory Perspective

Davi Sales Barreira, Asla Medeiros e Sá

FGV - EMAP, IMPATech

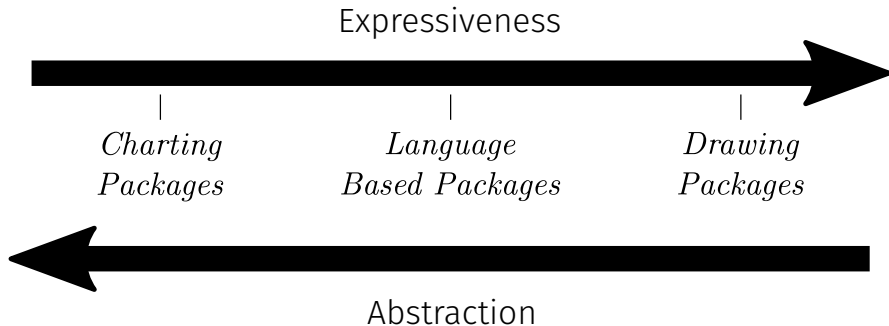
February 8, 2025

Table of contents

1. Vizagrams Hands-on
2. Category Theory behind Vizagrams

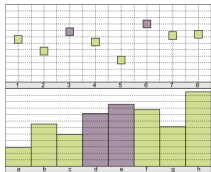
Introduction

Balance expressiveness and abstraction in data visualization frameworks.

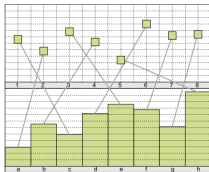


Introduction

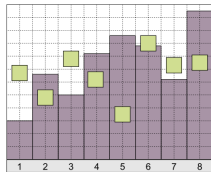
Visualization grammars often come short in terms of expressiveness, e.g., composite visualizations and visualizations with custom marks.



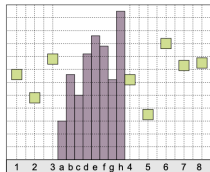
(a) Juxtaposed views.



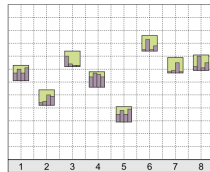
(b) Integrated views.



(c) Superimposed views.

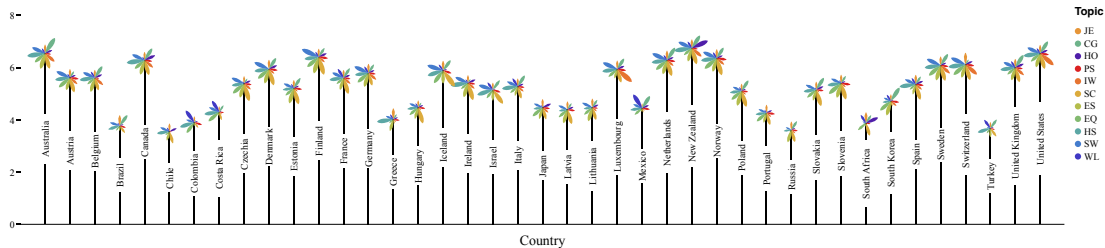


(d) Overloaded views.



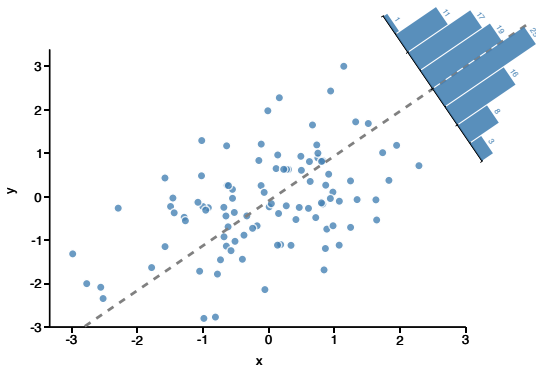
(e) Nested views.

Introduction



Introduction

```
d = scatter_plot +  
  T(x,y)R(θ)*histogram +  
  S(:strokeDasharray=>2)*Line([p1,p2])
```



Introduction - Objective and Hypotheses

Main Objective: Develop a data visualization framework that extends the expressiveness of visualization grammars while preserving a high level of abstraction.

Hypothesis 1 (Cause): Limitations in expressiveness are due to a lack of integration between graphic specification and assembly.

Hypothesis 2 (Solution): Integration of diagramming and graphic specification via a unified constructive framework.

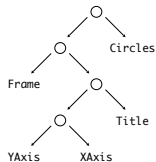
Formalization Tool: Category Theory.

Introduction - Overview

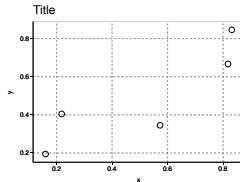
Specification

```
Plot(
  data = df,
  encodings=(
    x=(field=col_1,datatype=:q),
    y=(field=col_2,datatype=:q),
    color=(field=col_2,datatype=:n),
  ),
  ...
)
```

Assembly



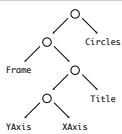
Display



Graphic Expression

```
I(row-> begin
  S(:fill=>row[:color])*
  T(row[:x],row[:y])*Circle()
end)
```

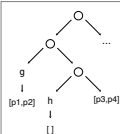
TMark



Mark \cong (Type, θ)

```
(XAxis,  $\theta$ )
struct XAxis <: Mark
  axis_ticks
  axis_title
  axis_length
  ...
end
 $\theta(x::XAxis)::T[Prim] = \dots$ 
```

T[Prim]



[Prim]

[g(p1), g(p2), p3, p4...]

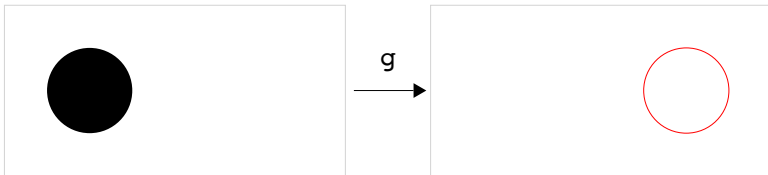
Render [Prim] with SVG

```
Circle(r=1) + <center r="1"/>
Circle(r=2) + <center r="2"/>
Line( ... ) + <polyline .../>
...
```


Data Visualization + Category Theory: Diagrams as Monoids

The basic building block for any diagram is the primitive. A primitive is any geometry that can be drawn and manipulated via geometric or stylistic transformations.

```
g = (T(1,0), Dict(:fillOpacity =>0,:stroke =>:red))
```



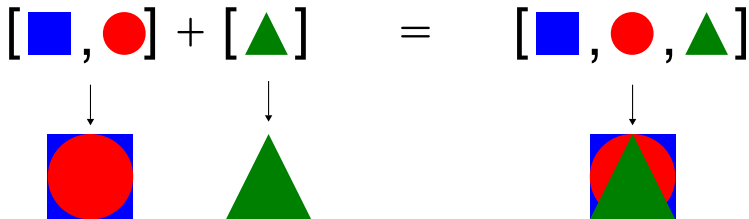
Data Visualization + Category Theory: Diagrams as Monoids

A diagram can be defined as an ordered list of primitives. $([\text{Prim}], +, [\])$ can be interpreted as a monoid.

```
abstract type Prim end
struct Circle <: Prim
  radius::Real
  center::Tuple{Real, Real}
end
+(d1::Vector{Prim}, d2::Vector{Prim}) = vcat(d1,d2)
```

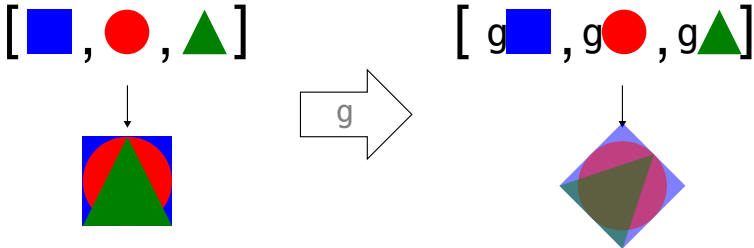
Data Visualization + Category Theory: Diagrams as Monoids

Diagrams are drawn by rendering the list of primitives on top of each other.



Data Visualization + Category Theory: Diagrams as Monoids

We can apply transformations to diagrams by applying the transformations to each primitive in the list.

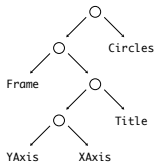


Data Visualization + Category Theory

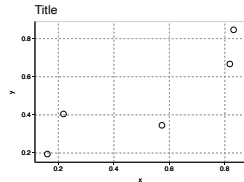
Specification

```
Plot(
  data = df,
  encodings=(
    x=(field=:col_1,datatype=:q),
    y=(field=:col_2,datatype=:q),
    color=(field=:col_2,datatype=:n),
  ),
  ...
)
```

Assembly



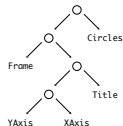
Display



Graphic Expression

```
Σ(row-> begin
  S(:fill=>row[:color])*
  T(row[:x],row[:y])*Circle()
end)
```

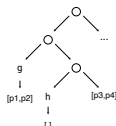
$\mathbb{T}[\text{Mark}]$



Mark \cong (Type, θ)

```
(XAxis,  $\theta$ )
struct XAxis <: Mark
  axis_ticks
  axis_title
  axis_length
  ...
end
 $\theta(x::XAxis)::T[\text{Prim}] = \dots$ 
```

$\mathbb{T}[\text{Prim}]$



[Prim]

[g(p1), g(p2), p3, p4...]

Render [Prim] with SVG

```
Circle(r=1) + <center r="1"/>
Circle(r=2) + <center r="2"/>
Line( ... ) + <polyline .../>
```

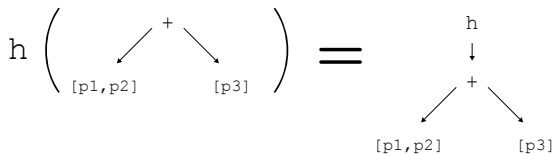
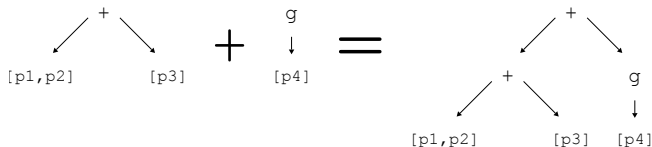
Data Visualization + Category Theory: Free Monads

Let us improve our representation using expression trees. To do this, first we define a functor F that encodes diagram composition (i.e. $+$) and diagram transformations.

```
@data F{a} begin
  Comp(::a, ::a)
  Act(::H, ::a)
end
fmap(f::Function, x::Comp) = Comp(f(x._1),f(x._2))
fmap(f::Function, x::Act) = Act(x._1,f(x._2))
```

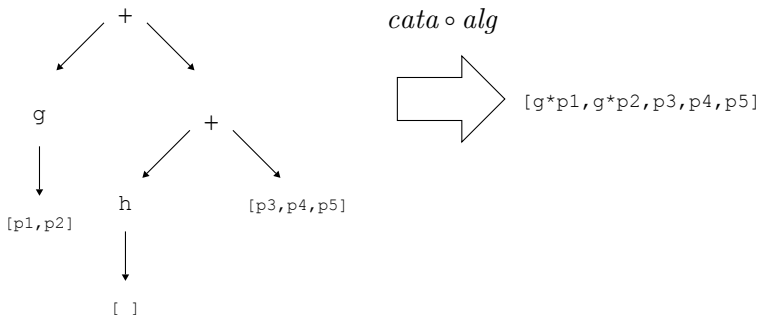
Data Visualization + Category Theory: Free Monads

Using our functor F , we can define a free monad over such functor, i.e $\mathbb{T} := \text{Free}F$. Thus, (\mathbb{T}, η, μ) is our monad, where \mathbb{T} is a parametric type representing trees with F as possible operations. To ease the process of expressing trees, we overload the $+$ and the $*$ operators to represent composition and transformation application respectively.



Data Visualization + Category Theory: Free Monads

A diagram tree is a value of type $\mathbb{T} [\text{Prim}]$. For this representation to be useful, we must be able to flatten it, i.e. we must define a function $f : \mathbb{T}[\text{Prim}] \rightarrow [\text{Prim}]$. We can do this using F -algebras and catamorphism.

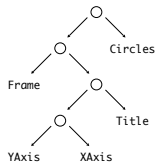


Data Visualization + Category Theory

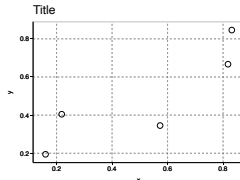
Specification

```
Plot(
  data = df,
  encodings=(
    x=(field=:col_1,datatype=:q),
    y=(field=:col_2,datatype=:q),
    color=(field=:col_2,datatype=:n),
  ),
  ...
)
```

Assembly



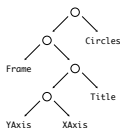
Display



Graphic Expression

```
I(row-> begin
  S(:fill=>row[:color]))*
  T(row[:x],row[:y])*Circle()
end)
```

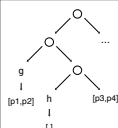
TMark



Mark \cong (Type, θ)

```
(XAxis,  $\theta$ )
struct XAxis <: Mark
  axis_ticks
  axis_title
  axis_length
  ...
end
 $\theta(x::XAxis)::T[Prim] = \dots$ 
```

T[Prim]



[Prim]

[g(p1), g(p2), p3, p4...]

Render [Prim] with SVG

```
Circle(r=1) + <center r="1"/>
Circle(r=2) + <center r="2"/>
Line( ... ) + <polyline .../>
...
```

Data Visualization + Category Theory

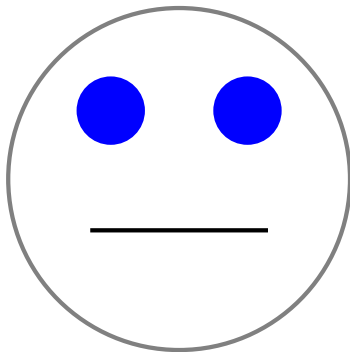
The current definitions of marks equate them with primitives. We propose a new definition:

Def: A **Graphical Mark** is a tuple (A, θ_A) , where A is a type and θ_A is a function $\theta_A : A \rightarrow \mathbb{T}[\text{Prim}]$.

```
struct Face <: Mark
  size::Real
  smile::Real
end

function  $\theta$ (face::Face)
  eyes = Circle(...) + Circle(...)
  smile = Bezier(...)
  head = Circle(...)

  return head + eyes + smile
end
```

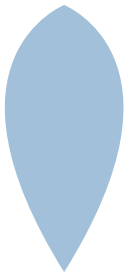


Data Visualization + Category Theory

We want to be able to define marks using previously defined marks. In other words, instead of specifying $\theta_A : A \rightarrow \mathbb{T}[Prim]$, we want to define a function $\zeta_A : A \rightarrow \mathbb{T}Mark$. In other words, we want use ζ_A to infer θ_A .

Petal

```
struct Petal <: Mark
  height::Real
  width::Real
  color::String
end
function  $\theta$ (p::Petal)
  return  $\eta$ ([Prim(
    BezierPolygon(...),
    Dict{:fill $\Rightarrow$ p.color}
  )])
end
```



Flower

```
struct Flower <: Mark
  heights::Vector{Real}
  widths::Vector{Real}
  colors::Vector{String}
end
function  $\zeta$ (fl::Flower)
  """
  Create petals and
  apply rotations.
  """
  ...
end
```



Data Visualization + Category Theory

Our ζ function can be formalized through the use of slice categories.

Def. The category of marks \mathcal{M} is a subcategory of $\mathbf{Set}_{\mathbb{T}}/[\mathbf{Prim}]$.

Assuming this category, a morphism $\zeta_A \in \mathrm{Hom}_{\mathcal{M}}(A, B)$ induces

$$\theta_A = \theta_{\mathbb{T}B} \circ \zeta_A = \mu \circ \mathbb{T}\theta_B \circ \zeta_A$$

This means that given an existing mark (B, θ_B) , we can define a new mark (A, θ_A) by defining ζ_A and inferring θ_A .

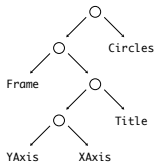
Programmatically, we can define an abstract type 'Mark', such that every subtype of 'Mark' has a *theta* function.

Data Visualization + Category Theory

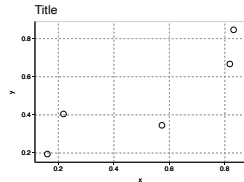
Specification

```
Plot(
  data = df,
  encodings=(
    x=(field=:col_1,datatype=:q),
    y=(field=:col_2,datatype=:q),
    color=(field=:col_2,datatype=:n),
  ),
  ...
)
```

Assembly



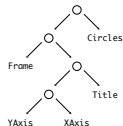
Display



Graphic Expression

```
Σ(row-> begin
  S(:fill=>row[:color]))*
  T(row[:x],row[:y])*Circle()
end)
```

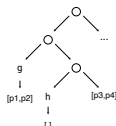
\mathbb{T} Mark



Mark \cong (Type, θ)

```
(XAxis,  $\theta$ )
struct XAxis <: Mark
  axis_ticks
  axis_title
  axis_length
  ...
end
 $\theta(x::XAxis)::\mathbb{T}[\text{Prim}] = \dots$ 
```

$\mathbb{T}[\text{Prim}]$



[Prim]

[g(p1), g(p2), p3, p4...]

Render [Prim] with SVG

```
Circle(r=1) + <center r="1"/>
Circle(r=2) + <center r="2"/>
Line( ... ) + <polyline .../>
...
```

Data Visualization + Category Theory

In summary:

- A diagram is a value of $[\text{Prim}]$;
- A diagram tree is a value of $\mathbb{T}[\text{Prim}]$;
- A mark is a value of type $A <: \text{Mark}$ for which $\exists \theta : A \rightarrow \mathbb{T}[\text{Prim}]$;

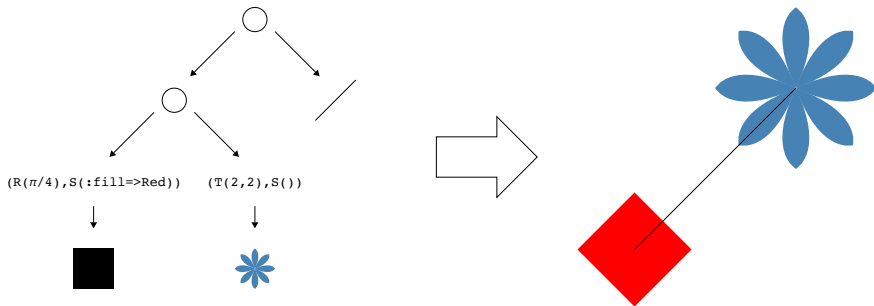
With all these concepts in mind, we can define **graphic** as a value of $\mathbb{T}\text{Mark}$, i.e. a tree where the leafs contain marks. Since each mark has a θ function, we can turn a $\mathbb{T}\text{Mark}$ into a $\mathbb{T}[\text{Prim}]$ using $\mu \circ \mathbb{T}\theta$.

Note that $\mathbb{T}\theta : \mathbb{T}\text{Mark} \rightarrow \mathbb{T}\mathbb{T}\text{Mark}$ and $\mu : \mathbb{T} \circ \mathbb{T} \rightarrow \mathbb{T}$.

Data Visualization + Category Theory

Using $+$ as diagram composition and $*$ to apply transformations, we can construct values of $\mathbb{T}\text{Mark}$ using a notation similar to mathematical expressions:

```
S(:fill=>:red)*R( $\pi/4$ )*Square() + T(2,2)*Flower() + Line([0,0],[2,2])
```

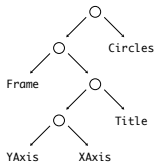


Data Visualization + Category Theory

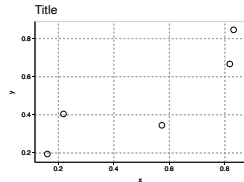
Specification

```
Plot(
  data = df,
  encodings=(
    x=(field=:col_1,datatype=:q),
    y=(field=:col_2,datatype=:q),
    color=(field=:col_2,datatype=:n),
  ),
  ...
)
```

Assembly



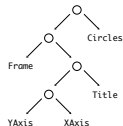
Display



Graphic Expression

```
Σ(row-> begin
  S(:fill=>row[:color])*
  T(row[:x],row[:y])*Circle()
end)
```

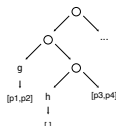
\mathbb{T} Mark



Mark \cong (Type, θ)

```
(XAxis,  $\theta$ )
struct XAxis <: Mark
  axis_ticks
  axis_title
  axis_length
  ...
end
 $\theta(x::XAxis)::\mathbb{T}[Prim] = \dots$ 
```

$\mathbb{T}[Prim]$



[Prim]

[g(p1), g(p2), p3, p4...]

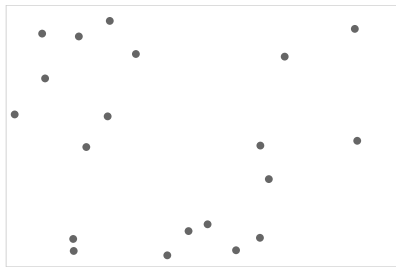
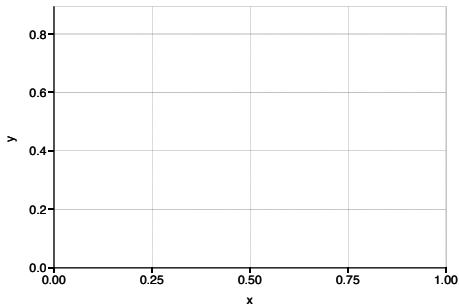
Render [Prim] with SVG

```
Circle(r=1) + <center r="1"/>
Circle(r=2) + <center r="2"/>
Line( ... ) + <polyline .../>
...
```


Data Visualization + Category Theory

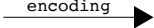
Plot Specifications

Plot = Guide + Graphic Expression \circ Scale(Data, Encodings)



Data Visualization + Category Theory

Guides are the visual aids which enable users to decode information regarding the visualization. Encodings are dictionaries which describe how to scale data attributes, with each encoding having a channel, a field and a scale.

Topic	Value	Country		x	y	color
JE	0.42	Australia		22.73	84.0	#E69537
CG	0.88	Australia		22.73	176.0	#6EB38A
...
SW	0.53	Costa Rica		185.7	106.0	#447CCD
WL	0.63	Costa Rica		185.7	126.0	#4039C4

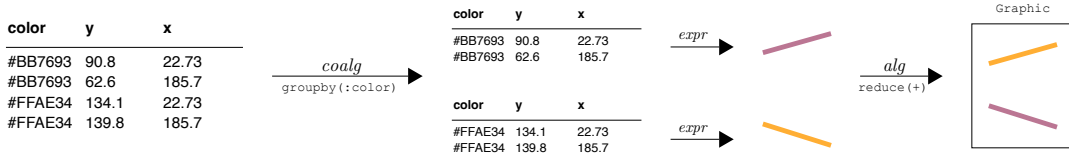
Data Visualization + Category Theory

The **graphic expression** is a function that takes as input the scaled data and returns a graphic. It iterates through the data, creating several graphical elements, which are then combined together to form the graphic. More formally, a graphic expression is a triple $(expr, alg, coalg)$, where:

$$\sum_{coalg}^{alg} expr \cong \left[\begin{array}{l} expr : D \rightarrow \mathbb{T}\{\text{Mark}\} \\ alg : [\mathbb{T}\{\text{Mark}\}] \rightarrow \mathbb{T}\{\text{Mark}\} \\ coalg : D \rightarrow [D] \end{array} \right]$$

Data Visualization + Category Theory

The graphic expression is very similar to the split-apply-combine pattern. The coalgebra does the splitting, the expression does the apply and the algebra does the combine.



Data Visualization + Category Theory

Here are some examples of graphic expressions:

$$\sum_{\text{row} \in D}^+ \text{T}(\text{row}[:x], \text{row}[:y]) * \text{Circle}(r = \text{row}[:\text{size}])$$

$$\sum_{\text{gdf} \in D_{\text{color}}}^+ \text{Line}(\text{gdf}[:x], \text{gdf}[:y], \text{color} = \text{gdf}[1, : \text{color}])$$

The End