Data Visualization From a Category Theory Perspective

Davi Sales Barreira, Asla Medeiros e Sá

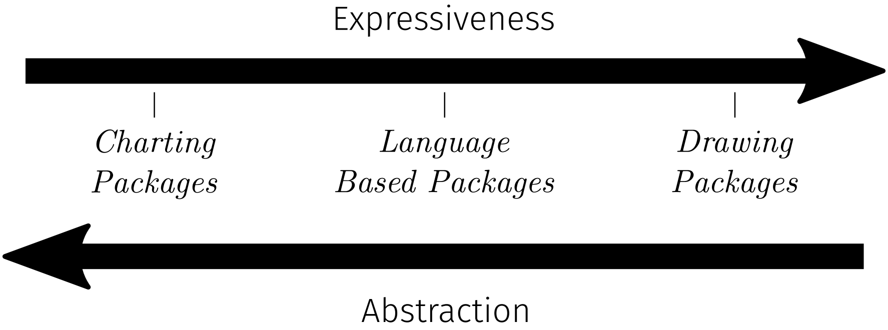
FGV - EMAp, IMPATech

February 8, 2025

Table of contents

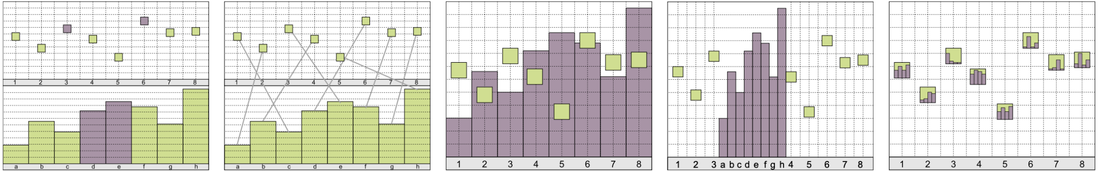
Introduction

Balance expressiveness and abstraction in data visualization frameworks.



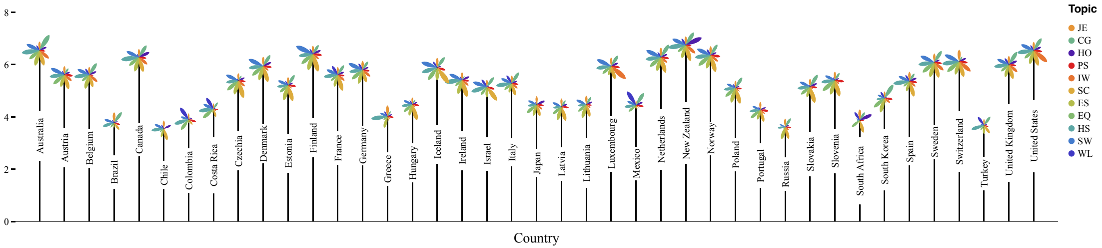
Introduction

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



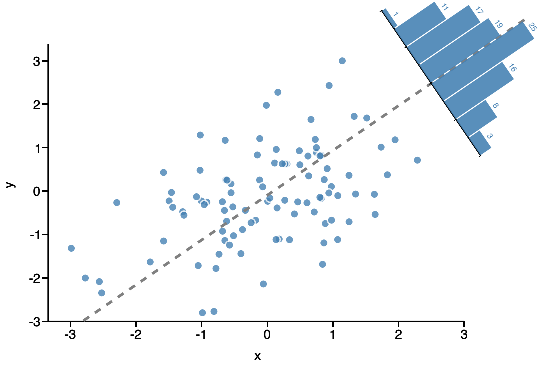
(a) Juxtaposed views. (b) Integrated views. (c) Superimposed views. (d) Overloaded views. (e) Nested views.

Introduction



Introduction

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



Introduction - Objective and Hypotheses

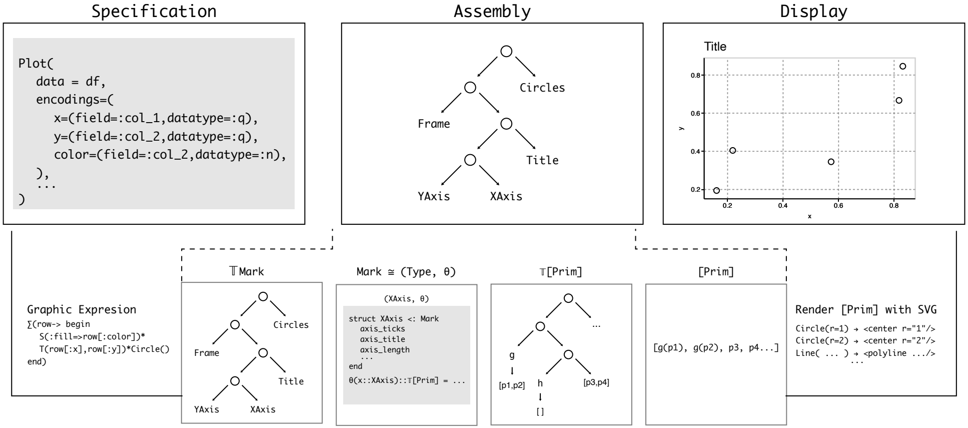
Main Objective: Develop a data visualization framework that extends the expressiveness of visualization grammars while preserving a high level of abstraction.

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.

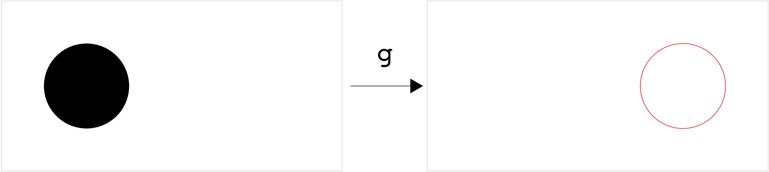
Introduction - Overview



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

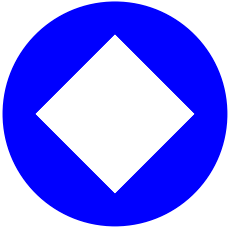
A diagram can be defined as an ordered list of primitives. $([\mathrm{Prim}], +, [\,])$ can be interpreted as a monoid.

```
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

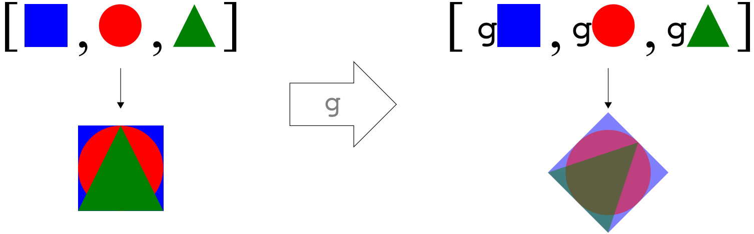
Diagrams are drawn by rendering the list of primitives on top of each other.

```
S(:fill=>:blue)*Circle() + R(π/4)S(:fill=>:white)*Square()
```

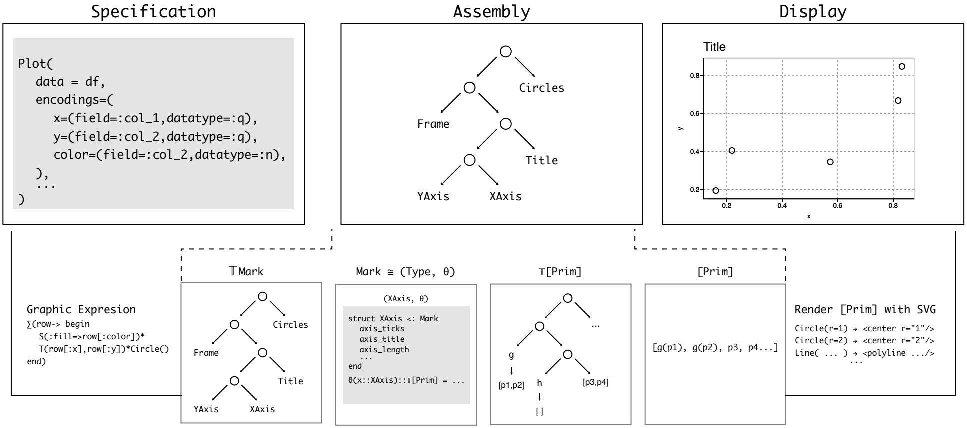


Data Visualization + Category Theory: Diagrams as Monoids

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



Data Visualization + Category Theory



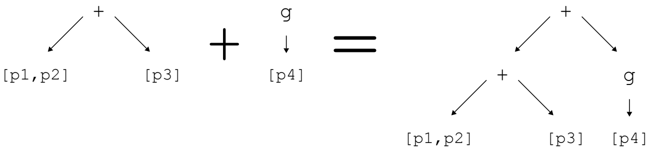
Data Visualization + Category Theory: Free Monads

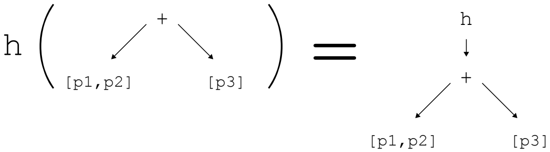
Let us improve our representation using expression trees. To do this, first we define a functor F that encodes diagram composition (i.e. $+$) and diagram transformations.

```
@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

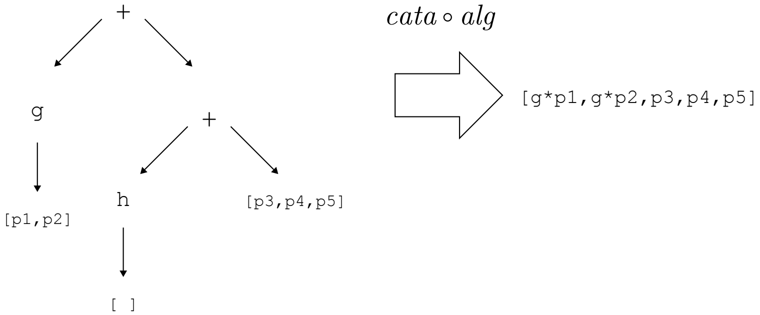
Using our functor F, we can define a free monad over such functor, i.e $\mathbb{T} := \text{Free}F$. Thus, $(\mathbb{T}, \eta, \mu)$ is our monad, where $\mathbb{T}$ is a parametric type representing trees with F as possible operations.
To ease the process of expressing trees, we overload the $+$ and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



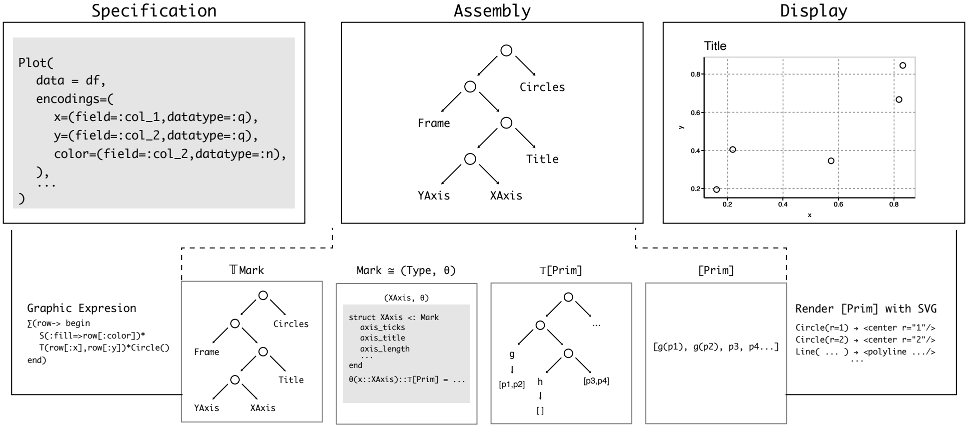


Data Visualization + Category Theory: Free Monads

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



Data Visualization + Category Theory



References