# Finite Inverse Categories as a Data Structure

Dimitris Tsementzis

June 7, 2018

Section 1

# Introduction

# What is category theory?

Category theory is the mathematical study of mathematical structure.

A category can be thought of as a collection of nodes (structures) and arrows between nodes (transformations).

Categories, the objects of study of category theory, are themselves mathematical structures.

**Example**

- The category **Set** with sets as nodes and functions as arrows
- The category **Graph** with graphs as nodes and graph homomorphisms as arrows
- The category $\omega$ with the natural numbers $n, m \in \mathbb{N}$ as nodes and with an arrow $n \to m$ iff $n \leq m$

# The Problem: Data Sets are too specific

Data is formalized as a mathematical structure.

Usually it is formalized as a table $T$ which we can think of as a set of rows $r_1, ..., r_n$ and a set of columns $C_1, ..., C_m$ such the $C_j$ are sets and $r_i \in C_1 \times ... \times C_m$

## Example

| Name | Age | Gender | Team |
|---|---|---|---|
| John | 13 | M | Bulls |
| Jane | 23 | F | Bucks |
| Pia | 28 | U | Bucks |
| John | 88 | M | Bulls |

**PROBLEM**: This formalism for data is often too fine-grained.

## Example

- What is the average age of John?

# Proposed Solution: Data Shapes

The usual approach to data is to formalize and store it in a maximally **fine-grained** way, and then introduce abstractions on top of this maximally fine-grained layer: **Sand Paradigm**

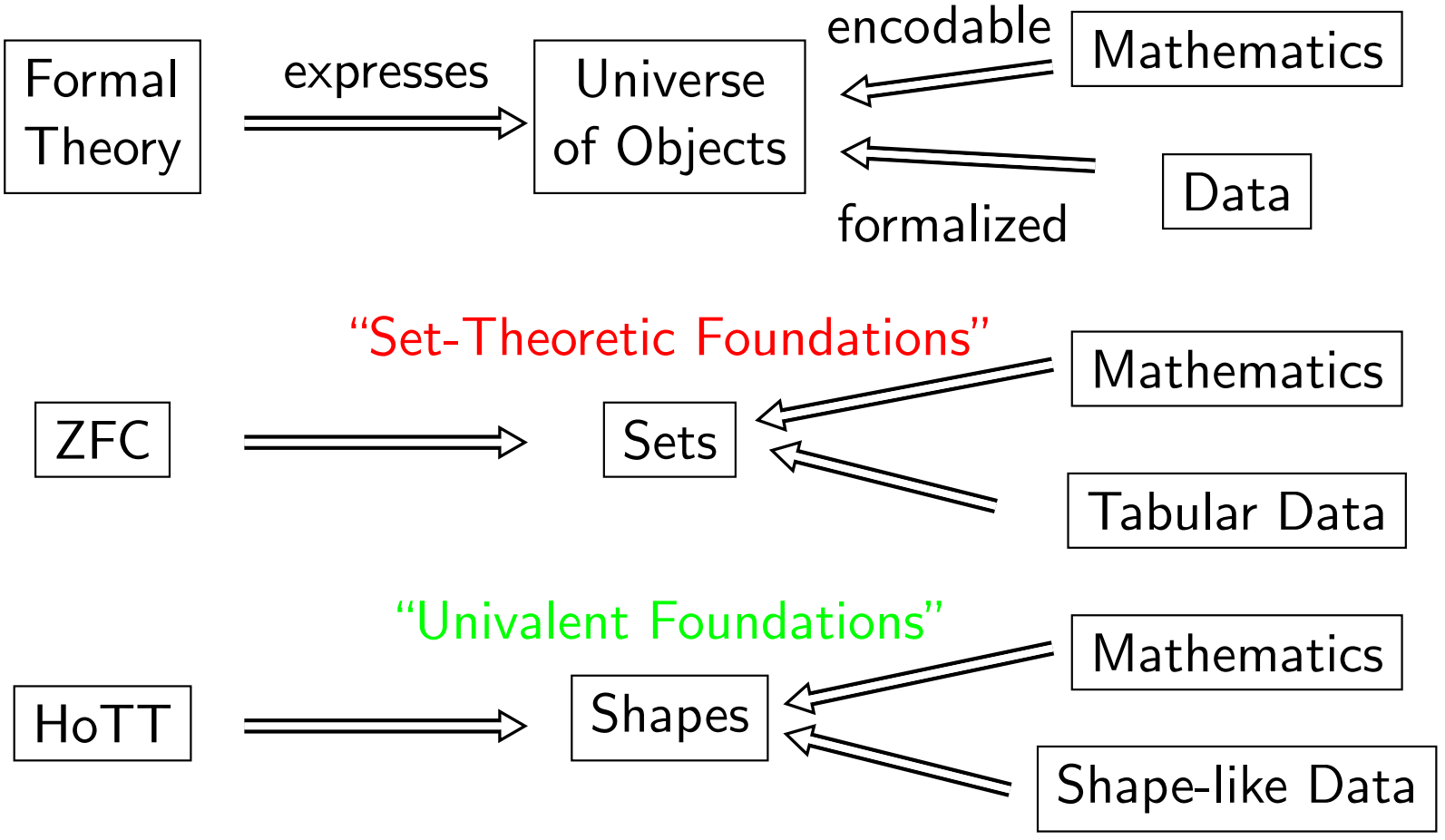
This works very well in some cases (e.g. reusability) but not so well in others (e.g. large databases, functional queries).

**IDEA**: Formalize, store and manipulate data in an **abstract** way, and fine-grain it at will: **Slab Paradigm**

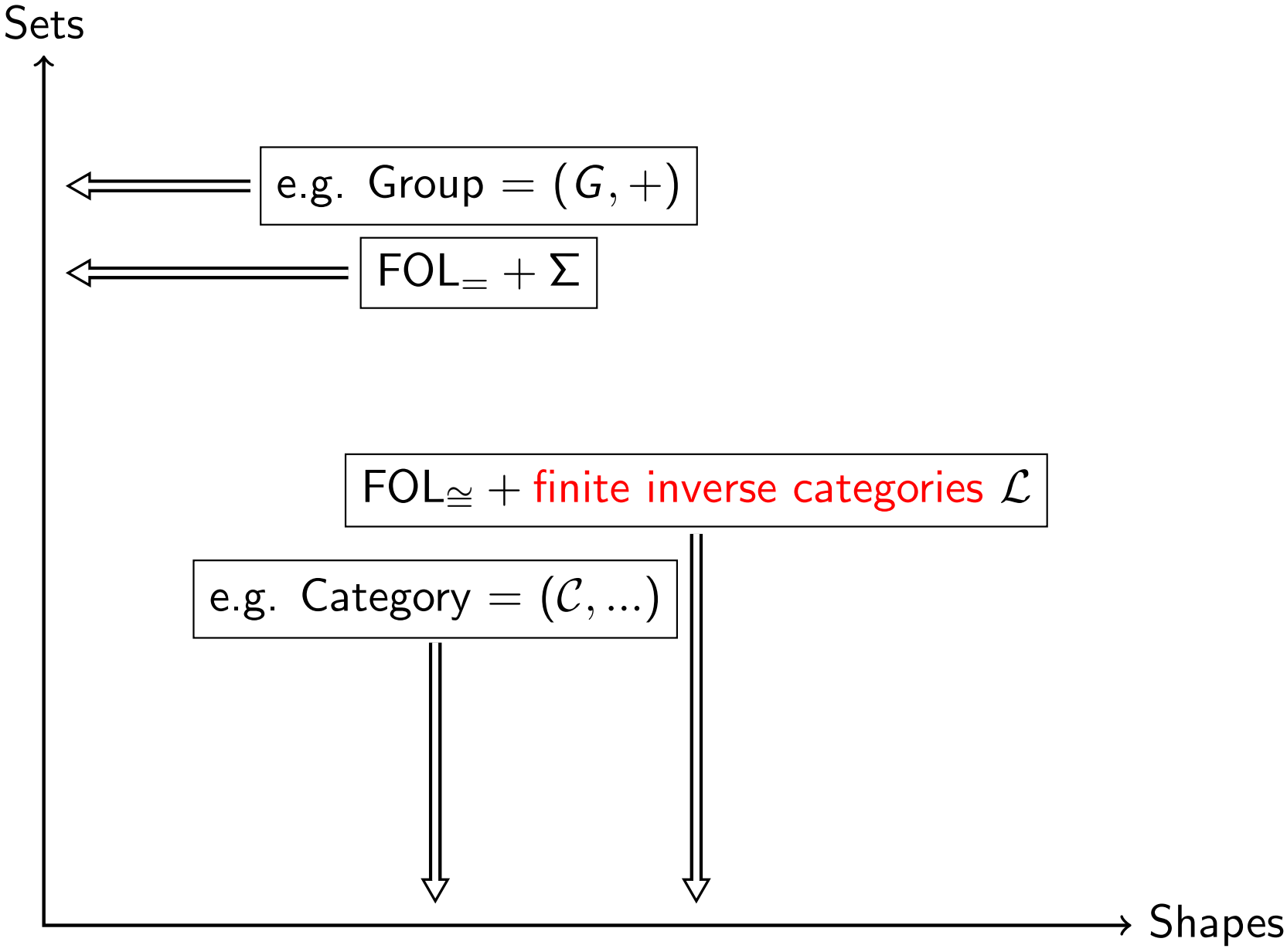
**IMPLEMENTATION:** A two-layer (abstract/concrete) framework for data.

**SLOGAN:** Data **is** Shape.

# The Big Picture: Foundations of Mathematics



# Big Picture: Role of Category Theory



# Related Work: Category Theory and Data

The idea of applying category-theoretic approaches to data, databases, data management etc. has been explored...

- Spivak, Kent. *Olog: A Categorical Framework for Knowledge Representation.* (2012)
- Wisnesky, Breiner, Jones, Spivak, Subrahmanian. *Using Category Theory to Facilitate Multiple Manufacturing Service Database Integration.* (2017)
- Rosebrugh, Wood. *Relational Databases and Indexed Categories.* (1992)

...and there is even, already, some work involving HoTT

- Chu, Weitz, Cheung, Suciu. *HoTTSQL: Proving Query Rewrites with Univalent SQL Semantics.*

# Section 2

## Finite Inverse Categories

# Finite Inverse Categories as damgs with extra structure

## Definition

A **directed acyclic (multi)graph** (da(m)g) is a directed (multi)graph $G = (O, A)$ with no paths from any node to itself (acyclic).

## Definition

A **finite inverse category** (fic) is a pair $(\mathcal{L}, \circ)$ where:

- $\mathcal{L}$ is a transitive damg
- $\circ$ is a partial associative binary operation on the edges of $\mathcal{L}$ which is defined only for "composable" pairs of arrows.

## Definition

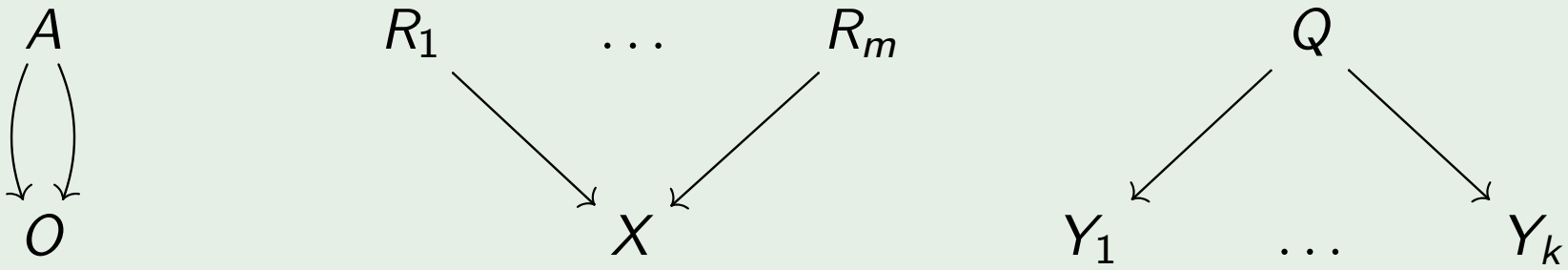
A **finite inverse category** (fic) is a finite skeletal category with no non-identity endomorphisms.

# Examples of fics

## Example

The one-object,one-morphism category $\bullet$ is a fic

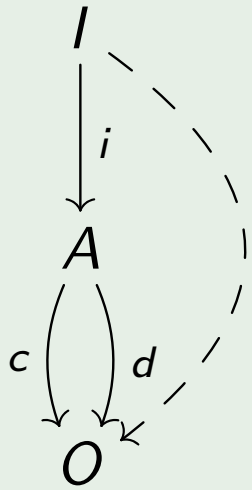
## Example



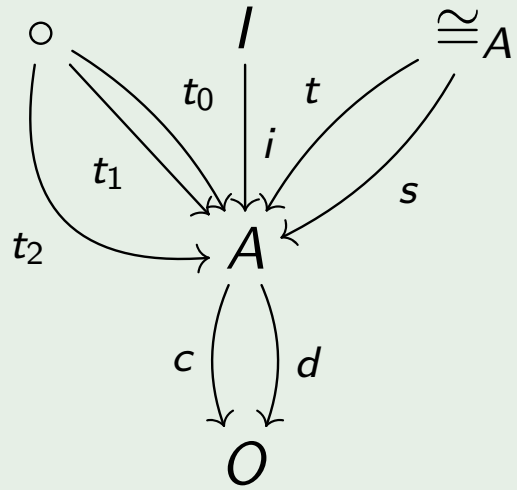

## Example

$\mathcal{L}_{\mathsf{rg}} :=$

$ci = di$

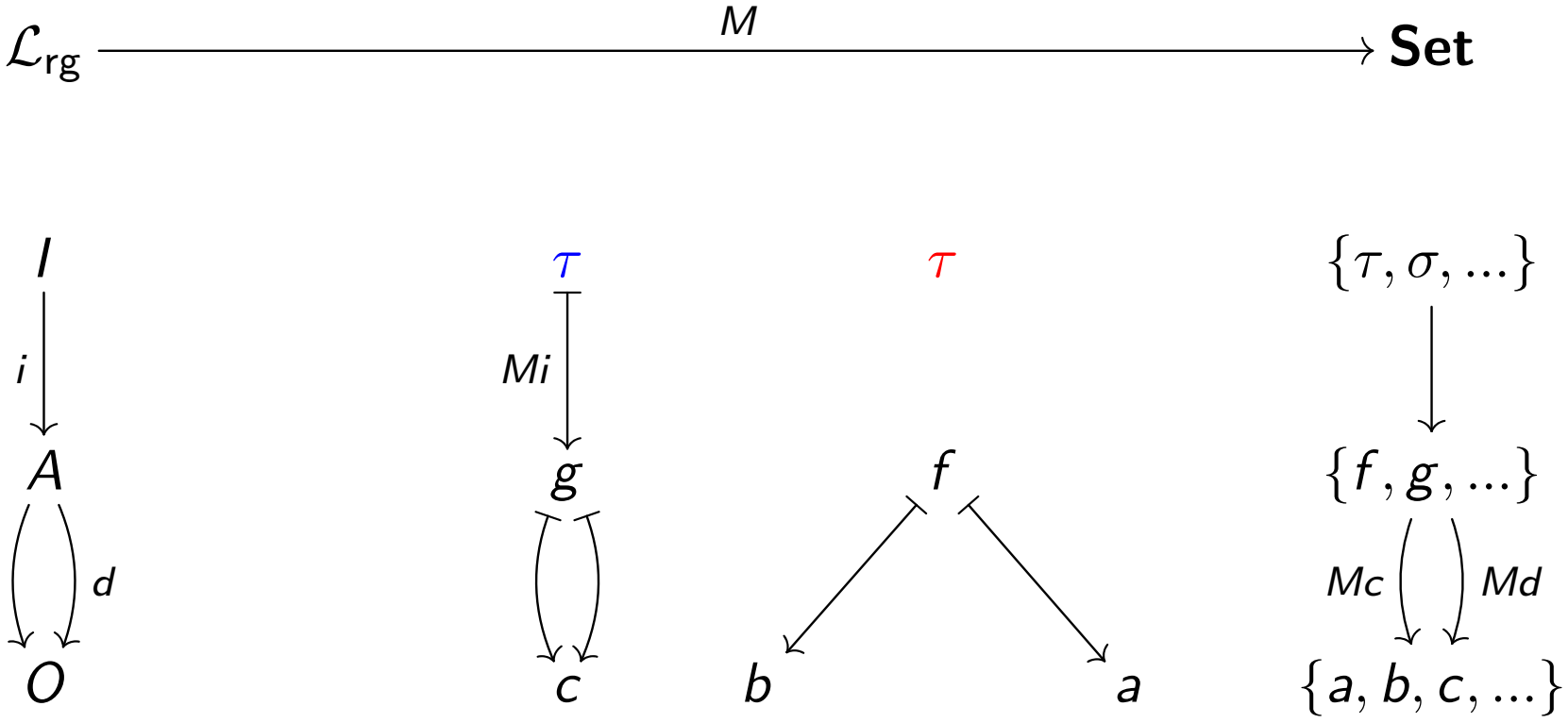



$\mathcal{L}_{\mathsf{cat}} :=$



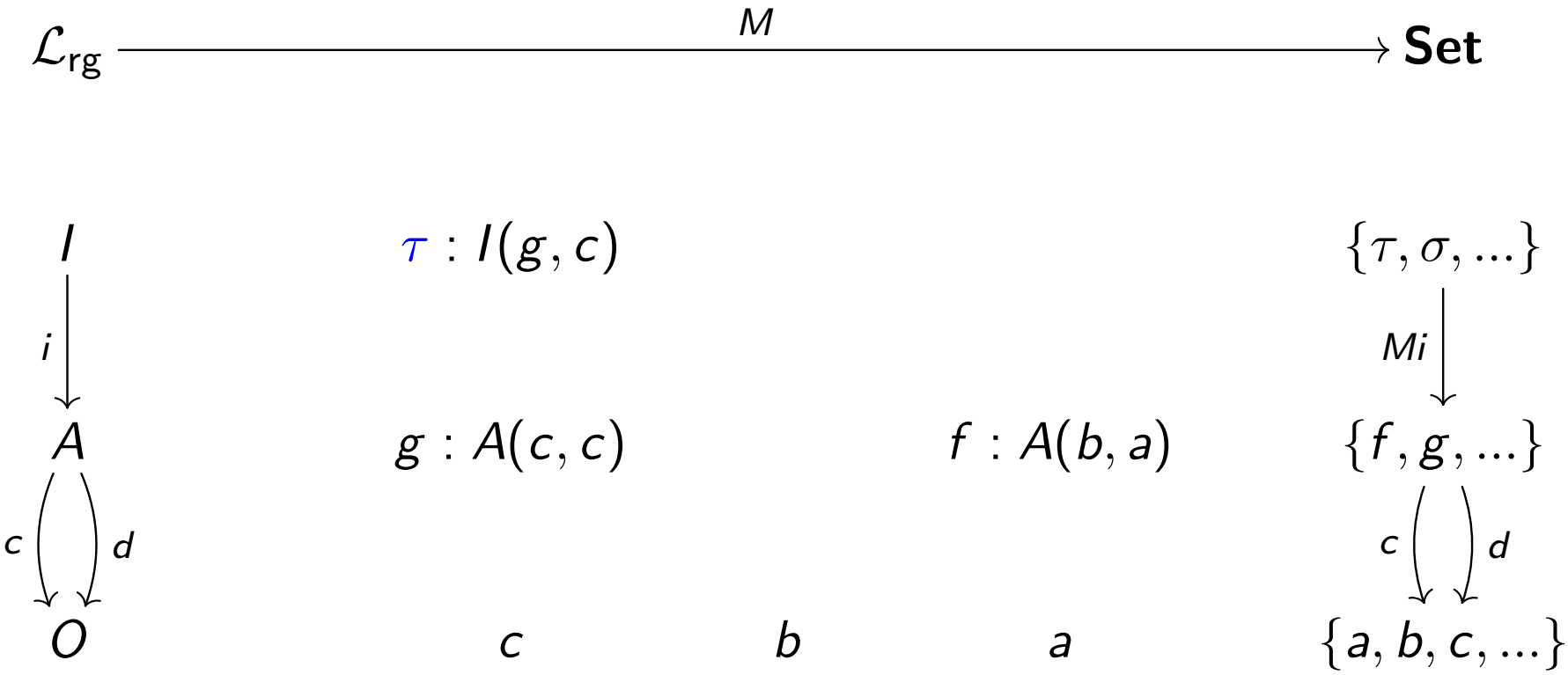
# How to think of fics

- A **schema** for dependently typed data
- A **syntax** for a certain logic (FOL$_{\cong}$)
- But **not** as ologs.



# How to think of fics

- $O$ : **Type** ..... $a, b, c : O$
- $A : O \to O \to$ **Type** ..... $g : A(c, c), f : A(b, a)$
- $I : \Sigma(x : O)A(x, x) \to$ **Type** ..... $\tau : I(g, c)$




**TAKE-AWAY:** fic data exists ONLY with its dependencies

# fics as a data structure

Fix a fic $\mathcal{L}$.

**Definition ("Data Shape")**

A **data shape** is a (Reedy fibrant) functor $M : \mathcal{L} \to \mathbf{Set}$.

**Definition ("Data Tranformation")**

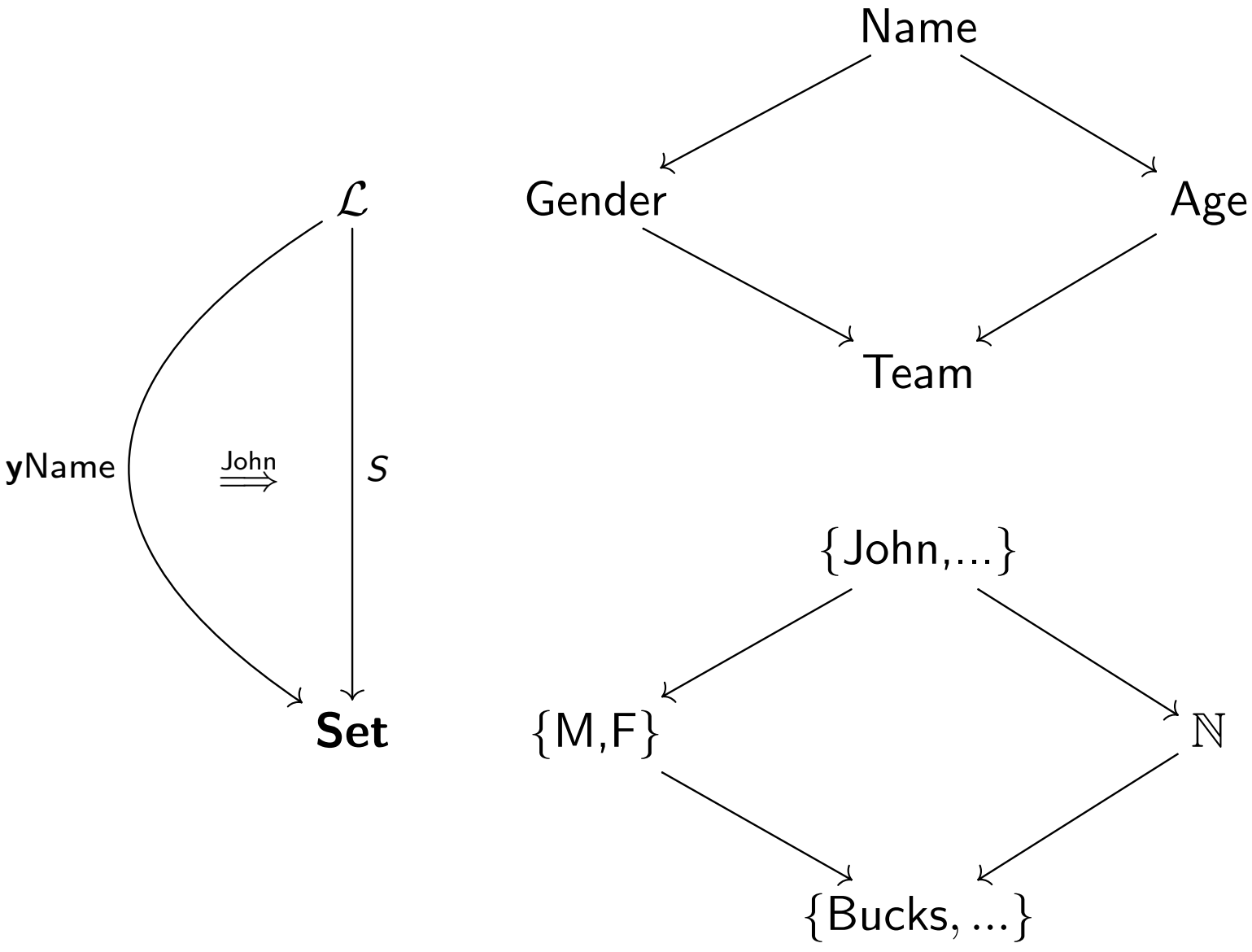
A **data transformation** is a natural transformation $\alpha : M \Rightarrow N$.

**Definition ("Data Point")**

If $M$ is a data shape and $C$ an object in $\mathcal{L}$, then a **data point** is a transformation $r : \mathbf{y}C \Rightarrow M$.

One can think of $r$ as a "row", and of $\mathbf{y}C$ as a "column" for our the "table" given by the data shape $M$.

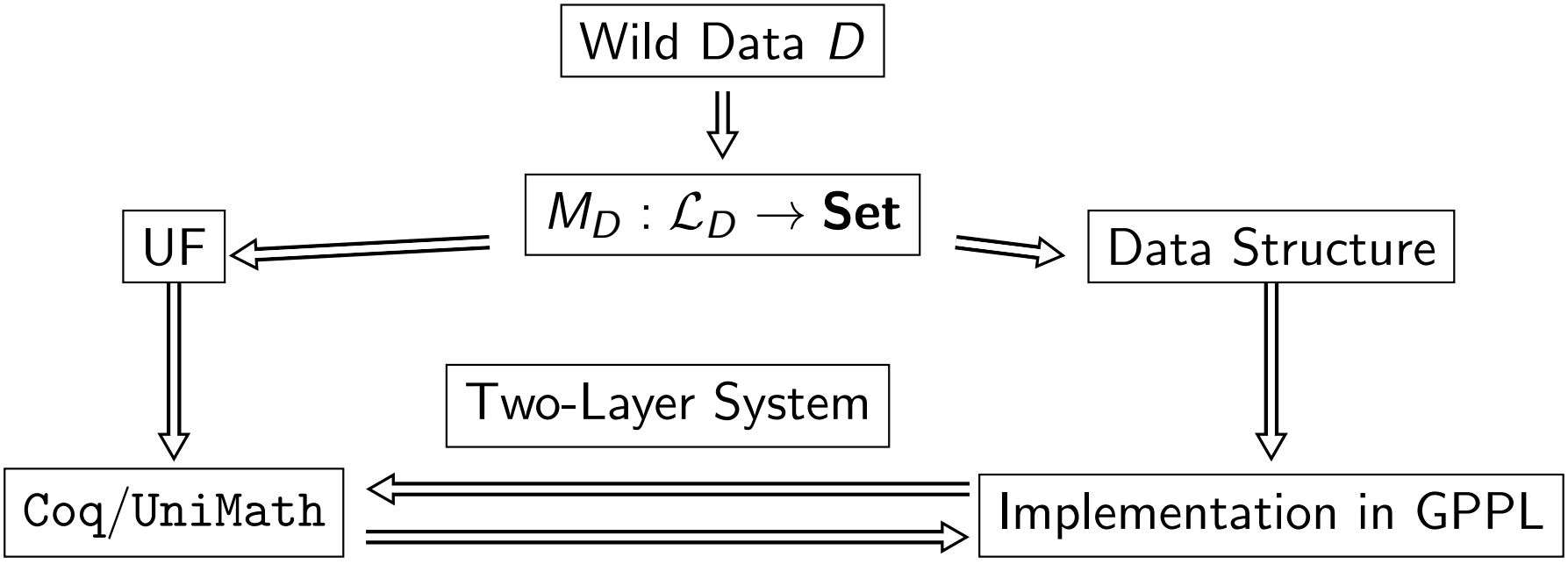
# Example of a data shapes and its points



Section 3

# Implementing Data Shapes

# Zoom out: How is this all relevant to the problem?




**Theorem (T.,2016)**

*Every fic $\mathcal{L}$ corresponds to a well-formed type $S_{\mathcal{L}}$ in MLTT and every $\mathcal{L}$-structure $M$ corresponds to a well-formed context.*

**Theorem (T., Weaver, 2017)**

*There exists a "mini type theory" $\mathbb{T}_{fic}$ whose well-formed contexts are in bijective correspondence to fics.*

# Implementing Data Shapes: A Prototype

Following this way of thinking I have created an experimental implementation of fics in Python, together with an interpreter that produces a Coq file containing the definition of the fic.

This establishes a minimal functional pipeline for storing, querying and manipulating data as abstract shapes.

Let's see a demonstration.

# Section 4

## Future Directions

# Questions and Further Directions

There are several directions of immediate interest to pursue, both theoretical and applied:

**Theoretical:** For $a, b \in M(K)$, when is $a \cong b$?

- *A Higher Structure Identity Principle*, (T., B. Ahrens, P. North, M. Shulman)

**Theoretical:** Implement the whole system **inside** HoTT.

- *HoTT+I*, (T., M. Weaver)

# Questions and Further Directions

There are several directions of immediate interest to pursue, both theoretical and applied:

**Applied:** Develop the DataShape prototype into a usable platform, possibly with a UI

**Applied:** Develop DataShape database for a real-world use case.

- Engineering blueprints
- Blockchain Data
- Graphs defining neural nets

**WHY NOW?**

1. Category Theory combined with new foundations.
2. DTT is becoming a GPPL

Thank you

# References


1. Makkai, Michael. *First-Order Logic with Dependent Sorts with Applications to Category Theory*. (1997)
2. Tsementzis, Dimitris. *First-Order Logic with Isomorphism*. arXiv preprint arXiv:1603.03092 (2016).
3. Tsementzis, Dimitris, and Matthew Weaver. "Finite Inverse Categories as Dependently Typed Signatures, arXiv preprint." arXiv preprint arXiv:1707.07339 (2017).
4. "The Univalent Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013