(Why) Should You Know Category Theory?

Vijay Anant & Raghu Ugare

Lambda Matters

Functional Conf, 2018

About Us

- Raghu Ugare
- Vijay Anant

Agenda

What This Talk Is *NOT* About

- YACTT — Yet Another Category Theory Tutorial!

What We Would Like To Talk About...

- How did we end up here?
- Brief intro to CT
- What's in it for U(s)?

How Did We End Up Here?

Abstractions

- Math is everywhere!
- Math is full of abstractions
- Programming/CS also deals with abstractions
- Math is pure!
- FP inspired from Math

Evolution Of FP

- Lambda Calculus (Alonzo Church, 1930)
- Category Theory (Mac Lane & Eilenberg, 1950s)
- LISP (John McCarthy, 1958)
- Many other FP languages
- Can Programming Be Liberated From The Von Neumann Style? (John Backus, 1977)
- Why Functional Programming Matters (John Hughes, 1990)

Evolution of Haskell

- Birth — Haskell 1.0 (1990)
- Teething problem — Purity Vs Side-Effects
- Solution ?
- Notions of Computations And Monads (Eugenio Moggi, 1991)
- Monads in Haskell (Philip Wadler, Simon Peyton Jones et al)

Category Theory

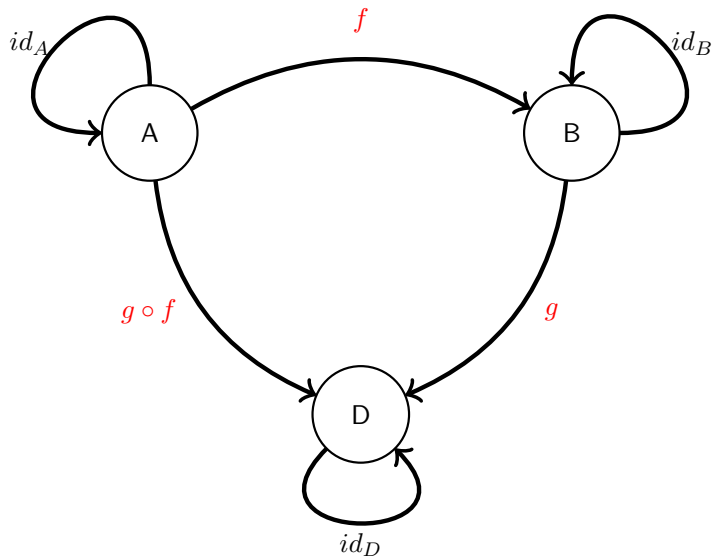
A Brief Intro. . .

Category Theory

Basics

- Category : objects + morphisms
- Morphisms should "compose"
- Category Laws:

$$(f \cdot g) \cdot h = f \cdot (g \cdot h) \quad \rightarrow \text{Associativity}$$
$$id \cdot f = f \quad \rightarrow \text{Left Identity}$$
$$f \cdot id = f \quad \rightarrow \text{Right Identity}$$



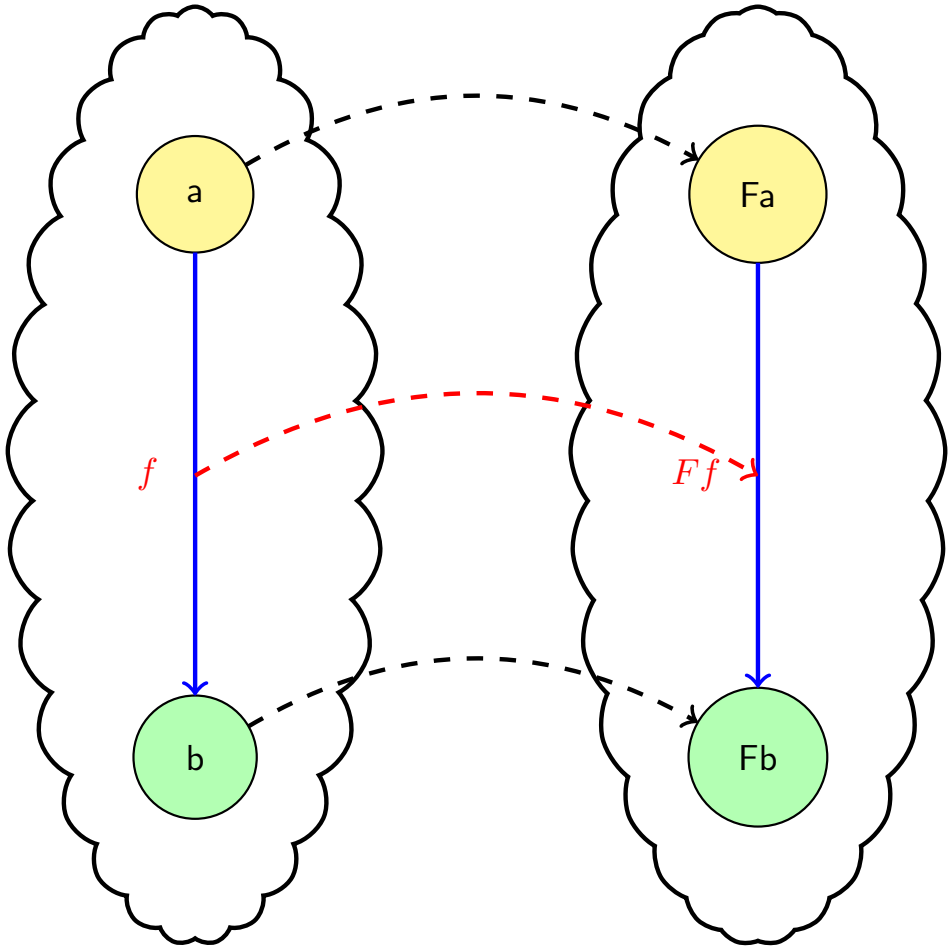
Examples

- Category **Set**: objects = sets & morphisms = mappings
- Category **Hask**: objects = types & morphisms = functions

Functors in Category Theory

Basics

- Functor is a morphism in the Category Cat.
- Category **Cat**: objects = Categories, morphisms = functors



Applying Category Theory...

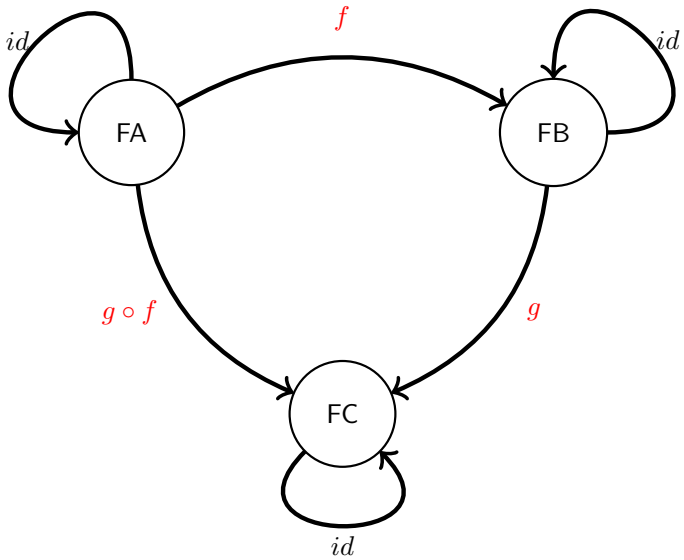
Modeling Tree Growth

- A tree "grows" every year, if **alive**
- Likelihood of being **hit**
- A **fallen** tree does not grow
- Need to calculate the probability of the tree being alive/hit/fallen after `n` years
- A Probabilistic Modeling Problem

Modeling Tree Growth (contd...)

- Possible states?
- `{Alive, Hit, Fallen}`
- Distribution
- `[(State, Probability)]`
- Events?
- `{ grow, hit, fall}`
- Each "transition"?
- $State \rightarrow Distribution$

What do we See ??




```
id     ::                  (a ->   a)
return :: (Monad m) => (a -> m a)

(.)    ::                  (b ->   c) -> (a ->   b) -> (a ->   c)
(<=<)  :: (Monad m) => (b -> m c) -> (a -> m b) -> (a -> m c)
```

What ELSE did we achieve?

The same $Distribution$ type can be used in modeling. . .

- Independent Events – Rolling a dice, Coin Toss
- Dependent Events – Monty Hall problem, Tree Growth year-after-year

For more details. . .

FUNCTIONAL PEARLS

Probabilistic Functional Programming in Haskell

MARTIN ERWIG and STEVE KOLLMANSBERGER

School of EECS, Oregon State University, Corvallis, OR 97331, USA

(*e-mail:* [erwig, kollmast]@eecs.oregonstate.edu)

1 Introduction

At the heart of functional programming rests the principle of referential transparency, which in particular means that a function f applied to a value x always yields one and the same value $y = f(x)$. This principle seems to be violated when contemplating the use of functions to describe probabilistic events, such as rolling a die: It is not clear at all what exactly the outcome will be, and neither is it guaranteed that the same value will be produced repeatedly. However, these two seemingly incompatible notions can be reconciled if probabilistic values are encapsulated in a data type.

To Conclude...

- Math provides abstractions
- Abstractions are powerful tools for dealing with Complexity!
- Design Patterns, Algorithms, Data-Structures
- Synergy needed between Theory and Practice
- But do we need to know it all?
- Well, We do not know what we do **not** know!

Thank You