Title:	Category Theory
Speaker:	Tom LaGatta
Typist:	Ray Puzio
Date:	#mar2014

Video of talk: `http://bit.ly/1mOfyE1`

A category consists of a collection of objects and morphisms. The morphisms are structure-preserving maps between objects.

(We will talk about "collections" rather than "sets" or "classes" because we don't want to get hung up on set-theoretic subtleties and connotational baggage .)

Structure is given by the category.

(We are not defining "structure", just using the intuitive notion as a guide.)

Formally, every morphism maps its source objects to its target objects.

Every object has an identity morphism. ("Hello — you're me ;)")

Morphisms are closed under composition in the case where the target of the first morphism equals the source of the second morphism.

Composition is an associative operation.

———————————————————————-

Pragmatic value: Unifying language for math (50% of mathematicians speak it.)

It's a convenient abstract language with lots of syntactic sugar. That makes it possible to reuse structure in different areas of mathematics.

Prototypical example is Set, the category of sets. objects: sets morphisms: maps between sets structure: membership, cardinality, no duplicates, no order

Sets are convenient for studying the foundations of mathematics because they come with a bare minimum of structure.

Example: Category of manifolds objects: smooth manifolds morphisms: smooth maps structure: calculus

Example: Category of finite dimensional vector spaces (over some given field) objects: vector spaces morphisms: linear transformations structure: linearity

Example: Topological vector spaces objects: vector spaces endowed with topology morphisms: continuous linear transformations structure: both topological and linear — they play together nicely

Example: Category of Hilbert spaces objects: Hilbert spaces morphisms: distance-preserving linear transformations structure: linear, topological, length, angle

Example: Haskell objects: Haskell types morphisms: Haskell functions structure: Cartesian closed (According to a result of Seeley, all typed lambda calculi can be interpreted in terms of Cartesian closed categories)

————————————————————————-

Interlude on LISP types

In pure lisp, there is only one type* $T :: T \to T$

However, this may not be the right way to think about this. Lisp is dynamically typed so we should be asking about the types of data as opposed to the types of variables. Put another way, when we consider the functor from programs to effects, the types show up on the right-hand side of the arrow as opposed to the left-hand side.

* "type" is a homonym in this domain

(Originally, were planning to return to this topic afterwards but time ran out.)

————————————————————————-

Example: Category of categories

objects: categories morphisms: functors structure: source, target, composition

Functors are structure-preserving maps between categories. They map objects into objects and morphisms into morphisms which preserve the structure of source, target, and composition.

Example: Category of a diagram objects: nodes morphisms: paths structure: how diagram is wired up

Note that there is some potential for ambiguity here — is a path which goes from A to B by following an arrow from A to C, then an arrow C to A to be considered the same as following the arrow from A to B? Both possibilities are valid; to be precise and rigorous, we will need to state what convention we choose. Also, for convenience, we might leave out some obvious arrows from the diagram, e.g. not indicate the identity morphisms explicitly.

This sort of thing is familiar from non-mathematical contexts. For instance, at business presentations, one often sees such diagrams used to illustrate manufacturing processes, cash flows, and the like. Category theory allows us to understand them as mathematical constructs.

Examples of functors can be shown by mapping such diagrams into other categories, such as the category of vector spaces.

Category theory is a convenient lingua franca for mathematics. It formalizes various notions: structure, stuff, property, natural, canonical, universal

References:

The n-cat lab: `http://ncatlab.org`

MacLane, S., 1997, Categories for the Working Mathematician, 2nd edition, New York: Springer-Verlag.

F W Lawvere and S H Schnauel Conceptual Mathematics: a first introduction to categories Buffalo Workshop Press, 1991.

David I. Spivak, Category Theory for Scientists `http://arxiv.org/abs/1302.6946`

Example: Category of database schemas objects: tables morphisms: foreign keys structure:

A filled database is a functor from this category to the category of whatever you are storing in your database.

Yoneda Lemma: Conceptually, we understand an object by its action on all other objects. Dually, we understand an object by the action of all other objects on it.

We have a locally small category C.

We take its opposite category C^{op}

We have the functor category $\mathrm{Fun}(C^{\mathrm{op}}, Set)$

Here, the objects are functors and the morphisms are natural transformations.

For every object A of C, let $H_A(B) = \mathrm{Hom}(B, A)$. Then $H_A = B \mapsto \mathrm{Hom}(B, A)$ is a functor from C^{op} to set.

The structure associated to A is captured by this functor H_A.

$H = (A \mapsto H_A)$ is the Yoneda embedding. It is full and faithful.