Cat Theo

June 2021

Contents

1 Introduction

1.1 Our Vision

Our project began in late April 2021, with the aim to create an interesting implementation of category theory, that could give a measure of interactivity with the end user through a graphical interface.

1.2 Our Idea

Our program is a tool for the representation of categories, by allowing the user to draw them as graphs, the program also includes a dynamic proofing algorithm, that procedurally checks the type of the morphisms, and is able to determine if the category is commutative.

2 The Implementation

2.1 Project Management

Since the beginning, we have made use of the Maven build tool to manage our directory structure and our dependencies, automatically managing versions and linking libraries to the IDE, allowing an easier management of the workplace environment on the different devices that we used

2.2 Graphical Interface

Our GUI adheres to the M.V.C (Model - View - Controller) paradigm, that allowed us to split our workload efficiently.
The framework that has been chosen is JavaFX, both because it is modern, so it is more aesthetically pleasing, and because it's philosophy is already in accordance with the M.V.C paradigm.

2.3 The Model

The model of our architecture corresponds to a model of a category, simulating in a completely independent manner the arrows and the objects; the model in itself therefore provides a CLI through which to interact with our program.

2.4 The GUI

The graphical interface is realized without the use of libraries other than JavaFX itself, the graph visualization is done through a series of trigonometrical calculation that make use of JavaFX's properties to recalculate the position of the arrow's endpoint and the line.
The line themselves are movable by the dragging the morphism's label, this is possible since the line is a Bezier's curve, and the label acts as a control point.

3 The Model

3.1 The Category

The mathematical model is based on the `Category` class, which could be regarded as a data structure embedding 2 different maps:

- The relationships between objects, their arrows and the spaces onto which they go or come from. (So the actual thing the user sees on the screen)
- The relationships between the arrows and the arrows which compose from them. (any change on an original arrow has to be reflected onto its composites)

3.2 The Objects

Even though it doesn't do much (and there indeed exist some implementations which don't use the concept of Objects) in our program the `Obj` class is used to store all the arrows which share the feature of incoming to this object (or outcoming from it) and all the spaces which reside in it. Computationally they pretty much do nothing (except from changing the name of their identities and spaces).

3.3 The Arrows

The `Arrow` class is one which represents the actual concept of Arrows and, indeed, has the main feature of being able to answer queries on its own properties (whether it is Epic, Monic, etc.).

3.4 Checking commutativity

A category commutes whether for each possible path from one object to another the result (in our case represented as the space of the final arrow) is the same. Indeed the method `commutes()` does this exact same thing, it finds each possible path (a thing, we know, is pretty inefficient), of course while taking into account cycles, and checks if the results would be the same.