A computer-checked library of Category theory with definitions of Functors and F-Algebras

Rado Todorov
Author

Benedikt Ahrens
Responsible Professor

Lucas Escot
Supervisor

1 Background

Category theory addresses mathematical structures and allows us to formally describe their relations. An example of a category is presented in Figure 1 and consists of:

- a collection of objects
- arrows between objects (called morphisms)
- an arrow for each object to itself (identity morphism)
- composition between morphisms

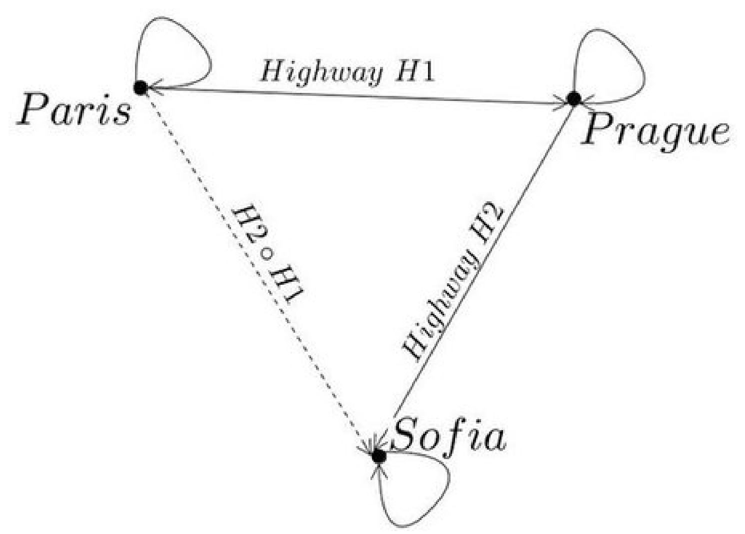

Figure 1. Category of road-connected cities. [1]
Arrows indicate whether 2 cities are connected.

Category theory is embedded in concepts of computer science:

- Type classes
- Polymorphic functions
- Recursion over recursive data types

The goal of this project is to create a library of category theory equipped with examples that is tailored towards newcomers to the field of category theory.This poster covers the notions of functor algebras

2 Methods

- Notions are defined based on notes by Ahrens et al. [2]
- The proof-assistant Lean [3] is utilized in to define and prove properties of the concepts in category theory
- Each definition is accompanied by examples that adhere to its laws
- Initially a basis of the library is implemented, followed by definition and examples of individual topics.

3 Implemented Concepts

Functors map objects and morphisms from one category to another. Endofunctors map from and to the same category. Implementation is shown in Figure 2.

```
structure functor (C D : category) :=
  (map_obj : C → D)
  (map_hom : Π {X Y : C} (f : C.hom X Y), D.hom (map_obj X) (map_obj Y))
  ...
```

Figure 2. Implementation of the functor definition

Algebras - presented in Figure 3.

- allow us to define and evaluate expressions by a given endofunctor *F*

$$F(X) \xrightarrow{\phi} X$$

Figure 3. Diagram representation of an algebra (X, φ). *F*(*X*) represents the value mapped by the endofunctor *F*.

$$\begin{tikzcd} F(X) \arrow[r, "\phi"] \arrow[d, "F(f)"'] & X \arrow[d, "f"] \\ F(Y) \arrow[r, "\psi"'] & Y \end{tikzcd}$$

Figure 4. Commutative diagram between algebras (X, φ) and (Y, ψ)

In Figure 4, f is a **homomorphism**, if the 2 paths from *F*(*X*) to *Y* are equal.
The category of algebras *Alg(F)* has algebras defined by *F as objects and homomorphisms as morphisms.*

Let *F*(*X*) map to *1 + A × X*.
Alg(F) defines list objects and their transformations from functional programming.
Figure 5 shows (*List A*, [*nil, cons*]) as the **initial** algebra and **fold** is the unique function from it to any other algebra defined by *F*.

$$\begin{tikzcd} 1 + (A \times ListA) \arrow[r, "{[nil,cons]}"] \arrow[d, "F(fold(\psi))"'] & ListA \arrow[d, "fold(\psi)"] \\ 1 + (A \times Y) \arrow[r, "{\psi=[\psi_a,\psi_b]}"'] & Y \end{tikzcd}$$

Figure 5. Commutative diagram between algebras (*List A*, [*nil, cons*]) and (Y, [ψ_a, ψ_b])

$$map\ (g) \circ map\ (f) = map\ (g \circ f)$$

Figure 6. Example of the fusion property with maps.The second traversal of the list can be omitted.

Composition in *Alg(F)* can be applied in functional programming by the name of **fusion** to exclude intermediate products as visualized in Figure 6.

Lambek's theorem
For any ***initial*** algebra (X, φ) exists a morphism *ψ such that* the composition of *φ* and *ψ* is the identity morphism - Figure 7.
Can be used to give recursive definition for fold-like morphisms for any **initial** algebra.

$$\begin{tikzcd} F(X) \arrow[r, shift left, "\phi"] \arrow[d, "F((\!|\psi|\!))"'] & X \arrow[l, shift left, "(\!|F(\phi)|\!)"] \arrow[d, "(\!|\psi|\!)"] \\ F(Y) \arrow[r, "\psi"'] & Y \end{tikzcd}$$

Figure 7. Lambek's theorem and visual proof of $(\!|\psi|\!) = \psi \circ F((\!|\psi|\!)) \circ (\!|F(\phi)|\!)$

4 Discussion

- The implementations prioritize using fields over arguments to avoid long type signatures

```
structure category :=
  (C₀      : Sort u)
  (hom     : Π (X Y : C₀), Sort v)
  (id      : Π (X : C₀), hom X X)
  ...
structure category (C₀:Sort u) (hom:Π (X Y : C₀), Sort v)
(id:Π (X : C₀), hom X X):=
  ...
```

Figure 8. Fields vs. Arguments

- Concepts are encoded as structures where possible to assemble useful data for future proofs.
- "Mirror image" concepts are implemented as standalone structures instead of applying the concept of duality.

5 Conclusions

We have :

- successfully implemented a library of category theory in the proof assistant Lean
- Provided example of well-known categories and concepts from computer science
- showed how algebras allow us to construct and reason about inductive types
- provided a generalized framework for recursion

6 References

[1] N. Grozev, "Functional Programming and Category Theory [Part 1] - Categories and Functors ," Nikolay Grozev,
[2] B. Ahrens, K. Wullaert, "Category Theory for Programming,"
[3] L. de Moura, S. Kong, J. Avigad, F. Van Doorn,and J. von Raumer, "The lean theorem prover (sys-tem description)," in Automated Deduction-CADE-25: 25th International Conference on Automated Deduc-tion, Berlin, Germany, August 1-7, 2015, Proceedings25, pp. 378–388, Springer, 2015.

TU Delft

R.A.Todorov@student.tudelft.nl