Geb Reading Group Syllabus

Purpose

This document is intended to summarize the background, especially in category theory and polynomial functors, required to understand how Geb works. It will serve as a guide to topics for the weekly reading group, and for attendees on how to prepare.

Practically none of the following is required just to write code in Geb! But it is required to understand how the language works under the hood, and that would also enable a programmer to use more of it.

Learning outline

The two main threads of theory underlying Geb are category theory and polynomial functors. To some degree I am treating the former as the theory of (programming) languages and the latter as the theory of data structures, but in Geb they are intertwined, defined through mutual recursion: category theory defined by data structures, data structures given semantics by category theory, and “(programming) language” emerging as a notion of its own (broader than “category”).

The big picture

The end goal of the language definition of Geb – the point at which the language itself is finished and all else is libraries – is what I have been calling “programming languages à la carte”. That is a reference, to “Data Types à la Carte”, so the founding paper there (see the Bibliography) is one place you might start. It illustrates how to define data structures in ways which allow for more combinators than traditional explicitly-recursive ADTs. Geb aims to extend this notion to programming languages – defining them in terms of individual language features and combinators on languages themselves.

That possibility emerges from founding category theory itself on polynomials.

Track: category theory

Fortunately, but not just coincidentally, the aspects of category theory underpinning Geb are mainly the foundational ones – the first few that are typically presented in books on category theory. One of my aims in the reading group will be to communicate these foundations in terms that will be most familiar to programmers – each of them has a clear analogue in and application to programming.

Here is a rough order of topics I’d recommend.

Definition

- The axioms which define a category
- The notion of isomorphism
- Question: what aspects of programming languages could be expressed as categories?

The category of categories

- Functors
- Natural transformations
- Question: which programming language constructs could be viewed as functors? As natural transformations?

Slice categories and (co)presheaves

Slice categories are the standard way of expressing dependent types in category theory, so they are particularly relevant to programming with formal verification. They are also a step on the way to understanding the more general notion of (co)presheaf.

Adjunctions, universal properties, representable functors, and the Yoneda lemma

Riehl is a superb reference on these intertwined topics, but also quite dense. They are perhaps best learned by example, of which several follow below. Geb turns their use somewhat on its head: instead of defining “universal property” in terms of categories, Geb defines polynomials without explicit reference to (although with antipication of) category theory, then universal properties in terms of polynomials, then categories in terms of universal properties. This sequence allows the intermediate definition of “programming language” in terms of universal properties, distinct from, though closely related to, that of “category”.

Examples of universal properties

The following examples should help to absorb the notion of “universal property” itself, but are also precisely the ones which underlie Geb, and programming languages in general. As such, they also underlie category theory

itself – they are the ones in terms of which category theory can be homoiconic (can write itself).

For each of these examples, therefore, the question is: what programming language concept(s) do these universal properties correspond to?

- Terminal objects
- Initial objects
- Products
- Coproducts
- Hom-objects (exponentials)
- Initial algebras
- Terminal coalgebras
- Free monads
- Cofree comonads
- Equalizers
- Coequalizers
- Pullbacks
- Pushouts
- Limits
- Colimits
- Ends
- Coends
- Kan extensions

Track: polynomial functors

By far the most comprehensive, up-to-date reference that I know of on this topic is from the Topos Institute – a book which is still being updated at least as of the last few months, and an accompanying video series (with more relevant videos frequently produced). The book was originally called Polynomial Functors: A General Theory of Interaction

; “General” has since been changed to “Mathematical”. The book’s website is in the Bibliography.

The book is large and contains many examples and applications; many of the formulas and theorems have been coded in Geb’s Idris implementation, and in many cases I’ve ported them to the dependently-typed context (that is, polynomial functors on slice categories). The most important thing to take from it is that there are many ways of viewing polynomial functors – if you can understand each and how to translate among them, then you will understand how to use them to write data structures, programming languages, category theory, and Geb in particular.

Bibliography

Category theory

- The Dao of Functional Programming by Bartosz Milewski
- The Joy of Abstraction by Eugenia Cheng
- Category Theory in Context by Emily Riehl

- Paranatural Category Theory
- Paranatural Category Theory Slides
- Paranatural Category Theory Video 1 (“(Co)ends and (co)structure”)
- Paranatural Category Theory Video 2
- A Categorical Semantics for Inductive-Inductive Definitions
- Coend Calculus
- Generic Programming With Adjunctions
- Adjoint Folds
- Conjugate Hylomorphisms
- Kan Extensions for Program Optimization
- Data Types à la Carte
- Richard Southwell: Category Theory for Beginners (videos)
- Bartosz Milewski video playlists
- Recursion schemes without the barbed wire (part 1)
- Recursion schemes without the barbed wire (part 2)

Polynomial functors

- Polynomial Functors: A Mathematical Theory of Interaction
- Dirichlet Functors are Contravariant Polynomial Functors
- From Algebras and Coalgebras to (Polynomial) Dialgebras
- nLab: polynomial functor
- nLab: parametric right adjoint