Lecture 21: Adjunctions, Algebras of a Monad

Lecturer: Max S. New
Scribe: Ayan Chowdhury

November 11, 2025

Recall that our model for call by value semantics is a BiCartesian Closed Category $\mathcal{C}$ with a strong monad T with some of the interpretations being:

$$\begin{aligned}
\Gamma, A \text{ (contexts and types)} &\to \mathcal{C}_0 \\
\Gamma \vdash M : A &\to \mathcal{C}(\Gamma, TA) \\
\Gamma \vdash M : A \text{ for } M \text{ a value} &\to \mathcal{C}(\Gamma, A)
\end{aligned}$$

where the value semantics and usual semantics align when M is a value (that is $[\![M]\!] = \eta([\![M]\!]^V)$).

One concrete example of Call-By-Value semantics is the category of sets with the Maybe monad, $\text{Maybe}\, A = A \uplus 1$. This lets us model crashing or uncatchable errors in our language. Let us consider how to model these semantics in Call-By-Name.

1 Call By Name

Let us first define our semantics concretely for the Maybe monad. We give the following interpretations:

$$\begin{aligned}
\Gamma, A \text{ (contexts and types)} &\to \text{Pointed Sets} \\
\Gamma \vdash M : A &\to f : |\Gamma| \to |A|
\end{aligned}$$

Where $|P|$ for a pointed set P denotes the underlying set. The intution for these interpretations is the base points provides the semantics for failing (or the program crashing). Note however that the function f is a function of underlying sets, and not a base-point preserving function. We interpret products in the following way:

$$[\![A \times B]\!] = [\![A]\!] \times [\![B]\!]$$

taking products in the category of pointed sets. We interpret a context Γ in the same way. The interpretation of the termianl object is given by

$$[\![1]\!] = 1$$

where 1 is taken to be the unit in the category of pointed sets (a set with just a base point and no other elements).

We also impose that for $\Gamma \vdash M : A$ when M is strict in x we have the function associated with the term M preserves the base point associated with x. This follows our intution, as if M is strict in x in Call-By-Name semanatics, then if x results in a crash, then our program will crash. Similarly, as we are strict in x, we map the error result, being the basepoint of x, to the error result of M.

This motivates the reason the function associated with $\Gamma \vdash M : A$ need not be base point preserving. We could return a non-error output even when all inputs are errors. It follows that

$$[\![A \Rightarrow B]\!] = \{\text{all (not necessarily basepoint preserving) functions } |[\![A]\!]| \to [\![B]\!] \}$$

The basepoint of this set of functions is the constant function which always returns the basepoint of B.

Consider the interpretation of 0. The empty set is not pointed, and thus cannot be the interpretation of 0. Instead we take

$$[\![0]\!] = (\varnothing)_*$$

where $(A)_*$ for any set A denotes freely adjoining a basepoint. Formally $(A)_* = A \uplus \{*\}$. Note that in our model it holds $[\![0]\!] \cong [\![1]\!]$.

The interpretion of sums is given by

$$[\![A + B]\!] = (|[\![A]\!]| \uplus |[\![B]\!]|)_*$$

We can observe that the semantics of $\cdot \vdash M : 1 + 1$ is just a function from $\{*\}$ to a three element set in both Call-By-Name and Call-By-Value.

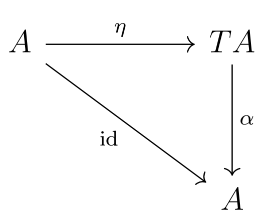
While seemingly different, both Call-By-Name and Call-By-Value can be treated as arising from a monad T. We have this directly for the interpretations for Call-By-Name. For Call-By-Value we consider Algebras of Monads.

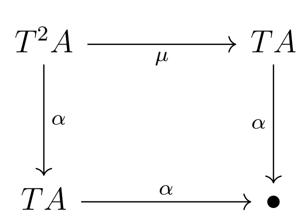
2 Algebra Of A Monad

An algebra of a Monad T over a category $\mathcal{C}$ consists of

- A "Carrier" $A \in \mathcal{C}$.
- An "Algebra" $\alpha : TA \to T$

such that the following diagrams commute:





As an example we have that an Algebra of the Maybe monad is a pointed set. We have from our first diagram that $\alpha : A \uplus \{\text{err}\}$ maps A to A via the identity. It follows that α is uniquely defined by where α sends err, so α exactly gives the data of a pointed set. This lets us redfine our semantics of Call-By-Name with the Maybe monad using algebras of the Maybe monad explicitly.

However, rather than defining pointed sets as being an algebra of a Monad, we can study algebraic structures directly and see how we can derive a monad from such structures.

3 Abstract Algebra

We define an Algebraic Theory to be an STLC-signature (where we take STLC to have no connectives) consisting of

- One base type X
- Operations of the form $\text{op} : X^n \to X$
- Axioms denoting relations between our operations

For example we can consider the theory of monoids given by a base type X, the operations multplication, $m : X, X \to X$ and identity, $e : \cdot \to X$, and the axioms

$$X \vdash m(x, e) \qquad X \vdash m(e, x) \qquad X, Y, Z \vdash m(x, m(y, z)) = m(m(x, y), z)$$

For a algebraic theory Σ we define a Σ-algebra to be an interpretation of Σ in Set. We then have that an algebra in our theory of monoids is exactly a monoid.

Many of our computation effects arise from algebraic theories. We could consider the following examples:

- Pointed set is an algebraic structure with just one operation, $\text{crash} : \cdot \to X$.
- Printing strings in the alphabet A^* is given by an operation $\text{print}_a : X \to X$ for all $a \in A^*$.
- Logging with a monoid W can be given by an operation $\text{act}_w : X \to X$ for all $w \in W$ satisfying $\text{act}_w(\text{act}_{w'}(x)) = \text{act}_{ww'}(x)$ and $\text{act}_e(x) = x$.
- Idempotent commutative monoid is a monoid such that $m(x, x) = x$ and $m(x, y) = m(y, x)$ and it gives a theory of finitary non-determinism.
- We can define the theory of state given a finite set of states S by defining operations $\text{put}_s : X \to X$ for all $s \in S$ and $\text{get} : X^S \to X$ satisfying

$$\text{put}_s(\text{get}(x_0, \cdots)) = \text{put}_s(x_s) \qquad \text{put}_s(\text{put}_{s'}(x)) = \text{put}_{s'}(x)$$

$$\text{get}(\text{put}_{s_0}(x_0), \text{put}_{s_1}(x_1), \cdots) = \text{get}(x_0, x_1, \cdots)$$

$$\text{get}(\text{get}(x_{0,0}, x_{0,1}, \cdots), \text{get}(x_{1,0}, x_{1,1}, \cdots), \cdots) = \text{get}(x_{0,0}, x_{1,1}, \cdots)$$

4 Category of Algebras

We can consider Σ-algebras to form a category. Given algebras (α, X) and (β, Y) we can define a homomorphism $\varphi : (\alpha, X) \to (\beta, Y)$ where φ maps X to Y such that for all opereations op we have

$$\varphi(\alpha_{\text{op}}(x_1, \cdots)) = \beta_{\text{op}}(\varphi(x_1), \cdots)$$

There then exists a functor U from $\text{Alg}(\Sigma) \to \text{Set}$ by mapping an algebra to its underlying set, and mapping a morphism to the underlying function (note that this is a forgetful functor). With this we can define the Cokleisli category given by

$$(\text{Cokleisli}\,\Sigma)_0 = \text{Alg}(\Sigma)$$

$$(\text{Cokleisli}\,\Sigma)((X, \alpha), (Y, \beta)) = \text{Set}(X, Y)$$

Taking U, we can go from $\text{Alg}(\Sigma)$ to Set. If we are also given a monad T, we can then generate a free algebra from Set.

5 Free Algebras

We have the following universal property. For all sets A, and given algebras (X, α) we have that

$$\text{Set}(A, U(X, \alpha)) \cong \text{Alg}(FA, X)$$

where FA denotes the free algebra. Intuitively we have that defining a homomorphism out of a free-algebraic structure of A is equivalent to defining a function from A.

We can explicitly construct

$$|FA| : [\left\{ \begin{array}{r} \cdot \vdash M : X \text{ generated by } \Sigma \text{ extended with} \\ \text{operations } \; : \cdot \to X \text{ for all } a \in A \end{array} \right\}]$$

where the brackets denotes equivalence classes up to equality from our axioms. Then for an operation $\text{op} : X^n \to X$ we can define

$$\text{op}_{FA}([M_1], \cdots, [M_n]) = [\text{op}(M_1, \cdots, M_n)]$$

However, note that such a definition of a free-algebra is very syntactic and not nice in general for proving theorems we want out of them. However, we can more explicitly construct free algebras given a particular theory.

For example, it holds that the free-algebra on A for monoids is the set of finite lists on A, where multplication is given by concatenation, and the identity corresponds to the empty list. We can similarly define free-algebra structures on all of the algebraic theories we had defined previously which give rise to computational effects.