Algebraic Structures

CS 152 (Spring 2025)

Harvard University

Thursday, April 2, 2025

Today, we will learn about

- Type constructors
  - Lists, Options
- Algebraic structures
  - Monoids
  - Functors
  - Monads
- Algebraic structures in Haskell

Algebraic Structures

- An abstract set of things (the *carrier*)
- An abstract set of operations over those things
- A set of laws that govern those operations

Algebraic Structures: Examples

- Numbers over addition and multiplication
- Lists over append
- Types over type constructors

Type Constructors

- A *type constructor* creates new types from existing types

Type Constructors

- A *type constructor* creates new types from existing types
  - E.g., product types, sum types, reference types, function types, ...

Lists

- Assume CBV λ-calc with booleans, fixpoint operator $\mu x : \tau.\ e$

Expressions	e	$::= \cdots \mid [\,] \mid e_1 :: e_2 \mid \text{isempty? } e \mid \text{head } e \mid \text{tail } e$
Values	v	$::= \cdots \mid [\,] \mid v_1 :: v_2$
Types	τ	$::= \cdots \mid \tau\ \textbf{list}$
Eval contexts	E	$::= \cdots \mid E :: e \mid v :: E \mid \text{isempty? } E \mid \text{head } E \mid \text{tail } E$

List inference rules

$$\frac{}{\text{isempty?}\ [\,] \longrightarrow \textbf{true}} \qquad \frac{}{\text{isempty?}\ v_1 :: v_2 \longrightarrow \textbf{false}}$$

$$\frac{}{\text{head}\ v_1 :: v_2 \longrightarrow v_1} \qquad \frac{}{\text{tail}\ v_1 :: v_2 \longrightarrow v_2}$$

$$\frac{}{\Gamma \vdash [\,] : \tau\ \textbf{list}} \qquad \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau\ \textbf{list}}{\Gamma \vdash e_1 :: e_2 : \tau\ \textbf{list}}$$

$$\frac{\Gamma \vdash e : \tau\ \textbf{list}}{\Gamma \vdash \text{isempty?}\ e : \textbf{bool}} \qquad \frac{\Gamma \vdash e : \tau\ \textbf{list}}{\Gamma \vdash \text{head}\ e : \tau} \qquad \frac{\Gamma \vdash e : \tau\ \textbf{list}}{\Gamma \vdash \text{tail}\ e : \tau\ \textbf{list}}$$

$$\text{append} \triangleq \mu f : \tau\ \textbf{list} \to \tau\ \textbf{list} \to \tau\ \textbf{list}.\ \lambda a : \tau\ \textbf{list}.\ \lambda b : \tau\ \textbf{list}.$$
$$\textbf{if}\ \text{isempty?}\ a\ \textbf{then}\ b\ \textbf{else}\ (\text{head}\ a) :: (f\ (\text{tail}\ a)\ b)$$

Options

Expressions	$e ::= \cdots \mid \text{none} \mid \text{some } e$
	$\mid \text{case } e_1 \text{ of } e_2 \mid e_3$
Values	$v ::= \cdots \mid \text{none} \mid \text{some } v$
Types	$\tau ::= \cdots \mid \tau \textbf{ option}$
Eval contexts	$E ::= \cdots \mid \text{some } E \mid \text{case } E \text{ of } e_2 \mid e_3$

Option as syntactic sugar

Option as syntactic sugar

- the type τ **option** as syntactic sugar for the sum type **unit** $+ \tau$

Option as syntactic sugar

- the type τ **option** as syntactic sugar for the sum type $\mathbf{unit} + \tau$
- none as syntactic sugar for $\mathsf{inl}_{\mathbf{unit}+\tau}\ ()$

Option as syntactic sugar

- the type τ **option** as syntactic sugar for the sum type $\mathbf{unit} + \tau$
- none as syntactic sugar for $\mathsf{inl}_{\mathbf{unit}+\tau}\ ()$
- some e as syntactic sugar for $\mathsf{inr}_{\mathbf{unit}+\tau}\ e$

Zoo of Generic Structures

- We like to deal with generic structures when possible, as a means of abstraction
- This is true both in proofs and in programs!
- A few common structures come up a lot

Monoids

A *monoid* is a set T with a distinguished element called the *unit* (which we will denote u) and a single operation $\mathit{multiply} : T \to T \to T$ that satisfies the following laws.

$$\forall x \in T.\ \mathit{multiply}\ x\ u = x \quad \text{Left id.}$$

$$\forall x \in T.\ \mathit{multiply}\ u\ x = x \quad \text{Right id.}$$

$$\forall x, y, z \in T.\ \mathit{multiply}\ x\ (\mathit{multiply}\ y\ z) = \mathit{multiply}\ (\mathit{multiply}\ x\ y)\ z \quad \text{Assoc.}$$

Monoid examples

- Integers with multiplication.
- Integers with addition.
- Strings with concatenation.
- Lists with append.

Do Types Form a Monoid?

- What is the carrier?
- What is the multiply operation?
- What is the unit?

Do Types Form a Monoid? Yes!

- What is the carrier? *The set of all types*
- What is the multiply operation? *The product type constructor*
- What is the unit? *The unit type*

Functors

A functor associates with each set A a set T_A; has a single operation $map : (A \to B) \to T_A \to T_B$ that takes a function from A to B and an element of T_A and returns an element of T_B

$$\forall f \in A \to B, g \in B \to C.$$

$$(map\ f); (map\ g) = map\ (f; g) \quad \text{Distributivity}$$

$$map\ (\lambda a : A.\ a) = (\lambda a : T_A.\ a) \quad \text{Identity}$$

Functor examples

- Options.
- Lists.

Monads

Monads

A monad associate each set A with a set M_A. Two operations:

- $return : A \to M_A$
- $bind : M_A \to (A \to M_B) \to M_B$

Monad laws

$$\forall x \in A, f \in A \to M_B.$$

$$bind\ (return\ x)\ f = f\ x \quad \text{Left id.}$$

$$\forall am \in M_A.\ bind\ am\ return = am \quad \text{Right id.}$$

$$\forall am \in M_A, f \in A \to M_B, f \in B \to M_C.$$

$$bind\ (bind\ am\ f)\ g =$$

$$bind\ am\ (\lambda a\!:\!A.\ bind\ (f\ a)\ g) \quad \text{Assoc.}$$

Option monad

$$\mathsf{return} : \tau \to \tau \ \textbf{option}$$

$$\mathsf{bind} : \tau_1 \ \textbf{option} \to (\tau_1 \to \tau_2 \ \textbf{option}) \to \tau_2 \ \textbf{option}$$

Algebraic structures in Haskell

- https://www.haskell.org/
- Pure functional language
- Call-by-need evaluation (aka lazy evaluation)
- Type classes: mechanism for ad hoc polymorphism
  - Declares common functions that all types within class have
  - We will use them to express algebraic structures in Haskell

Why Monads?

- Monads are *very* useful in Haskell
- Haskell is pure: no side effects
- But side effects useful!
- **Monadic types cleanly and clearly express side effects computation may have**
- Monads force computation into sequence
- Monads as type classes capture underlying structure of computation
  - Reusable readable code that works for any monad

Further Reading

- Monadic Parsing in Haskell (Functional Pearl)
  `https://www.cs.nott.ac.uk/~pszgmh/pearl.pdf`
- Free Monads
  `https://okmij.org/ftp/Computation/free-monad.html`