Differentiating Data Structures

Mitch Stevens

March 21, 2018

What is an Algebra

An **algebra** is a pair of monoids over some type.

Examples of Algebras include:

- $\mathbb{B}$: (False, True, or, and)
- $\mathbb{N}, \mathbb{R}, \mathbb{C}$: $(0, 1, +, \times)$
- Sets: $(\varnothing, U, \cup, \cap)$
- Polynomials of Algebras
- Square Matrices: $(0_n, I_n, +, \times)$
- Musical Notes: $\mathbb{Z}_{12}$
- Musical Sequences (Euterpea)
- Pretty Printing
- Image Composition

```
class Alg a where
  zero :: a
  one :: a
  (+) :: a -> a -> a
  (*) :: a -> a -> a
```

Laws:

- Monoid Laws
- $(a + b) * c = (a * c) + (a * c)$

Types form an Algebra

A **sum type** takes a value from A **or** from B

- `Either a b`
- `data Bool = True | False`
- `data Maybe a = Nothing | Just a`

A **product type** takes a value from A **and** from B

- `(a, b)`
- `data Person = Person Name Age`
- `data Time = Time Hour Minutes Seconds`

```
instance Algebra Type where
  zero = Void
  one  = ()
  (+)  = Either
  (*)  = (,)
```

Does this satisfy the algebra law from earlier?

$$(a+b)*c = a+c*b+c$$

Traffic light example

- How many values inhabit `Traffic`?
- How many values inhabit `Intersection`?
- How many values inhabit `ComplexInter`?

```haskell
data Traffic = R | Y | G

data Intersection = Inter
    { north :: Traffic
    , east  :: Traffic
    , south :: Traffic
    , west  :: Traffic
    }

type ComplexInter lane =
    lane -> Traffic
```

Type Isomorphism

There is a difference between the set of Types and the equivalence class of types ($\mathbb{T}$)

The size of a type is the number of values that can inhabit it. Lets call this size computing funtion

$$\sigma :: \mathbb{T} \to \mathbb{N}$$

Name	Constructor/Type	Size
Void	N/A	0
Unit	()	1
Boolean	`False | True`	2
Int	$-2^{31} \ldots 2^{31} - 1$	2^{32}
String	`[Char]`	∞

Traffic light example

- How many values inhabit `Traffic`?
- How many values inhabit `Intersection`?
- How many values inhabit `ComplexInter`?

```haskell
data Traffic = R | Y | G

data Intersection = Inter
    { north :: Traffic
    , east  :: Traffic
    , south :: Traffic
    , west  :: Traffic
    }

type ComplexInter lane =
    lane -> Traffic
```

How many values inhabit List?

$$
\begin{aligned}
\sigma(\texttt{List a}) &= \sigma(\texttt{Nil | Cons a (List a)}) \\
&= \sigma(\texttt{Nil}) + \sigma(\texttt{Cons a (List a)}) \\
&= 1 + \sigma(a) \times \sigma(\texttt{List a}) \\
&= 1 + \mathsf{a} \times (1 + a \times \sigma(\texttt{List a})) \\
&= 1 + \mathsf{a} + \mathsf{a}^2 \times \sigma(\texttt{List a}) \\
&= 1 + \mathsf{a} + \mathsf{a}^2 \times (1 + a \times \sigma(\texttt{List a})) \\
&= 1 + \mathsf{a} + \mathsf{a}^2 + a^3 + a^4 + \dots
\end{aligned}
$$

So we have:

$$\sigma(\texttt{List a}) = \sum_{n=0}^{\infty} a^n = \frac{1}{1-a}$$

Which strongly resembles the geometric sum ($|r| \geq 1$):

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r}$$

Refactoring

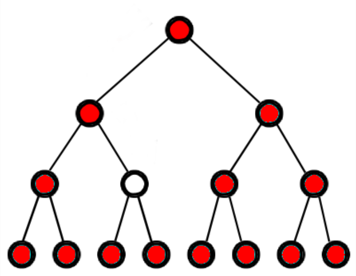
Give that σ is a isomorphism, we know that $id_{\mathbb{T}} = \sigma \circ id_{\mathbb{N}} \circ \sigma^{-1}$.

$$\begin{tikzcd} \mathbb{T} \arrow[r, "id_{\mathbb{T}}"] \arrow[d, "\sigma"] & \mathbb{T} \\ \mathbb{N} \arrow[r, "id_{\mathbb{N}}"] & \mathbb{N} \arrow[u, "\sigma^{-1}"'] \end{tikzcd}$$

Example: Refactoring the type `Either (b, a, [b]) a`

One-Hole Contexts and Zippers

A 'One-hole Context' is a data structure with a 'hole' where a value can be inserted.



If you include a value, you can use a one-hole context to recreate the original data-structure. The product of a value and a one-hole context is called a zipper.

Zipper of a List

The List Zipper is defined `type ListZipper a = ([a], a, [a])`. It provides fast modification of adjacent elements in a list.

$$
\begin{array}{lcccl}
(& [a_0, \ldots, a_{i-2}], & a_{i-1} & ,[a_i, \ldots, a_n] &) \\
(& [a_0, \ldots, a_{i-1}], & a_i & ,[a_{i+1}, \ldots, a_n] &) \\
(& [a_0, \ldots, a_i], & a_{i+1} & ,[a_{i+2}, \ldots, a_n] &)
\end{array}
$$

Rules of Differentiation

Usually only differentiation is only defined over functions $\mathbb{R}^m \rightarrow \mathbb{R}^n$. But the rules that come from differential calculus can be applied to any algebra.
This is not usually meaningful, however.

Sum Rule:

$$\partial_x(A+B) = \partial_x A + \partial_x B$$

Product Rule:

$$\partial_x(A \times B) = \partial_x A \times B + A \times \partial_x B$$

Chain Rule:

$$\partial_x(A \circ B) = \partial_x A(B) \times \partial_x A$$

Differentiating Types

The dervivative of a Type is meaningful, it is **exactly** the type of its one-hole context!

It is easier to differentiate a function in $\mathbb{N}$ than a function in $\mathbb{T}$. We apply the same trick as before.

$$\begin{tikzcd}
\mathbb{T} \arrow[r, "\partial_{\mathbb{T}}"] \arrow[d, "\sigma"] & \mathbb{T} \\
\mathbb{N} \arrow[r, "\partial_{\mathbb{N}}"] & \mathbb{N} \arrow[u, "\sigma^{-1}"']
\end{tikzcd}$$

Computing the One-Hole Context of a List

Other Questions

- Can we perform more than one differentiation?
  Yes.
- Is there a method for computing anti-differentiation?
  No idea.
- Is there a reasonable interpretation for a fractional or negative type?
  Kind of.