A functional programming approach to mathematical analysis

Simplifying mathematical analysis

Satrajit Chatterjee

Canadian Undergraduate Mathematics Conference, July 2022

Computer Science and Mathematics
University of Ottawa

Why are we here

~~To learn functional programming!~~

jk, uOttawa had to make paradigms a part of my sequence to get me to care about that.

But then it got interesting, so who knows, you might like it too.

Why I care about this, and why you might

- I'm a CS Student.
- Real Analysis was hard.
- I want it to be not hard.

So what are paradigms

Programming language classification based on their features.

Some of the popular ones:

- Imperative $\rightarrow$ Tell a computer how to do something
- Declarative $\rightarrow$ Tell a computer what to do
- Object-oriented
- Logical
- Functional

Object Oriented Programming

```
public class Car {
    String name;
    public static Planet (String s) {
        this.name = s;
    }
}

public class Main {
    public static void main(String[] args) {
        String s = "Uranus";
        Planet p = new Planet();
        System.out.println(p.name);
    }
}
```

```
p.name = "Pluto";
```

- You're allowed to do this.
- Pluto is now officially a planet!
- You did what NASA couldn't!!

Logic Programming

Prolog

- Atoms
- Predicates

```
planet(earth).
animal(mars).
planet(venus).
planet(venus).
neighbors(earth, mars).
neighbors(earth, venus).
% Find all neighbors of earth
neighbors(earth, X).
```

Logic Programming

- Propositional Logic
- Discrete Math
- Solving the Knapsack problem if you're familiar with it
- Natural Language Processing somehow, I never managed to get around to trying it out.

Finally, Functional Programming

- The exact opposite of imperative programming (where you change the state of something)
- Everything is a function. And there's some "variables".
- Immutable values.
- And apparently quite useful in math. We'll find out more in a second!

OCaml

```
type planet = {name: string; isplanet: bool} ;;
let pluto : planet = { name = "pluto"; isplanet = false } ;;

(* Will only output false *)
pluto.isplanet = true ;;
```

Cannot save Pluto from it's doom as a non-planet anymore :(

But that's okay! We have more important things to worry about.

So how is Domain Specific Languages different?

- Opposite to General Purpose Languages
- Language specific to a... domain
- We can treat Haskell and OCaml as a DSL for now... even though its not.
- No ideal Domain Specific Language for math at the moment, but that's something worth looking into.

So how is Domain Specific Languages different?

You might've heard of:

- HTML
- CSS
- SQL

Fun Fact

HTML + CSS is Turing complete.

So what now

You might be wondering, where is all the math...

Well we're here!

Types

- A lot of functional programming depends on user defined data types.
- Use this to our advantage to apply it to math.
- You must've heard of complex numbers...
- Most regular programming languages haven't heard of them.
- So let's teach it to 'em!

Types

A complex number has the form $a + bi$ or $a + ib$.

This can have an abstract functional representation as a datatype:

$$\begin{aligned} \text{data Complex} = &\ Plus_1\ \mathbb{R}\ \mathbb{R}\ I \\ &|Plus_2\ \mathbb{R} I\ \mathbb{R} \end{aligned}$$

If we account for the fact that complex numbers are isomorphic to pairs of real numbers, we can define it in terms of a new data type:

newtype Complex $= C(\mathbb{R}, \mathbb{R})$

Types

We can define addition and subtraction operations on complex numbers as

$$w + z = (a + x) + (b + y)i$$
$$w - z = (a - x) + (b - y)i$$

From a functional perspective,

```
(+) : Complex -> Complex -> Complex
(C (a, b)) + (C (x, y)) = C ((a + x), (b + y))
```

We can turn this into a functional datatype

```
data ComplexSyntax = i
    | ToComplex R
    | Plus ComplexSyntax ComplexSyntax
    | Times ComplexSyntax ComplexSyntax
    | ...
```

Welcome to Pattern Matching

```
let is_empty = function
  | [] -> true
  | _ :: _ -> false
;;
```

Types

Haskell

```
class Monoid m where
    mempty :: m
    mappend :: m -> m -> m
    mconcat :: [m] -> m
    mconcat = foldr mappend mempty

multi:: Int->Int
multi x = x * 1
add :: Int->Int
add x = x + 0

main = do
    print(multi 9)
    print (add 7)
```

Recap: Monoid $\rightarrow$ $Ia = aI = a$

Closed under an associative binary operation and has an identity element.

Lets bring it back to math

Every natural number is even or odd.

With the base case that 0 is even or 0 is odd, we can use an inductive property where n is even or n is odd, $n \in \mathbb{N}$.

```
datatype evenodd = Even | Odd;

fun test 0 = Even
  | test n = (case test (n-1) of Even => Odd
                               | Odd => Even);
```

Example in Analysis

Completeness property: if A is a set of real numbers with at least one number in it, there exists a real number y such that $x \leq y$ for every $x \in A$ (upper bound) and a smallest such number (least upper bound or **supremum** of A).

Functionally representing sup which is defined only for those subsets of $\mathbb{R}$ which are bounded from above,

$sup : \mathcal{P}^{+}\mathbb{R} \to \mathbb{R}$

$min : \mathcal{P}^{+}\mathbb{R} \to \mathbb{R}$
$min\ A = x \iff (x \in A) \wedge (\forall a \in A : x \leq a)$

Example in Analysis

Because sup A is similar to max A but is also the smallest element of a set, there is a connection to min. We can introduce the function:

$\textit{ubs} : \mathcal{P}\mathbb{R} \to \mathcal{P}\mathbb{R}$
$\textit{ubs}\ A = \{x | x \in \mathbb{R}, x \text{ upper bound of } A\}$

Example in Analysis - modularizing the problem

Because sup A is similar to max A but is also the smallest element of a set, there is a connection to min. We can define the function:

$ubs : \mathcal{P}\mathbb{R} \to \mathcal{P}\mathbb{R}$
$ubs\ A = \{x | x \in \mathbb{R}, \forall a \in A : a \leq x\}$

This function returns the upper bounds on A. The completeness axiom can now be stated as:

Assume an $A : \mathcal{P}^{+}\mathbb{R}$ with an upper bound $u \in ubs(A)$ Then $s = sup\ A = min(ubs(A))$ exists.

Lambda Functions

Anonymous functions that are either one time use or can be passed as a parameter.

```
(\x y -> x + y) 3 5
8 :: Integer
```

A more practical programming application:

```
addOneList lst = map addOne' lst
    where addOne' x = x + 1
```

But...

Is this easier to implement than simply writing a proof by hand?

no ;-; for the most part

but it does help modularize and simplify problems in analysis.

do you have any questions? Me too! ;-;

you can reach me at **satrajit314@gmail.com**