Why is so much of FP about Types instead of Functions?

Donovan Crichton

October 2023

Preliminaries

- Slides and Examples available at:
  https://github.com/donovancrichton/Talks
- This talk: BFPG/WhyIsFPAboutTypes

About Me

Australian National University

- PhD Candidate
- Computing Foundations
- School of Computing

- Visiting Scholar
- Trusted Systems Lab
- IIIS

- ASD Co-Lab Scholar

What is a Function?

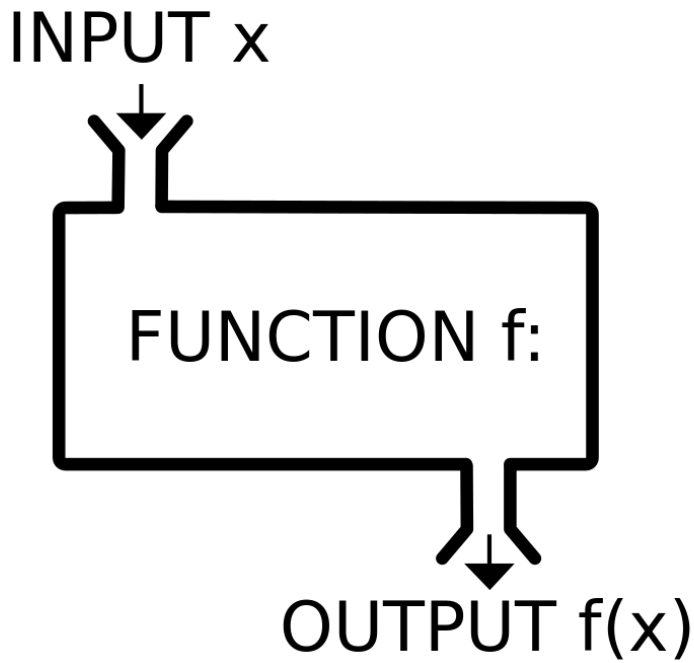

Figure: A "black-box" depiction of a function[1]

[1] Source https://en.wikipedia.org/wiki/Function_(mathematics)#/media/File:Function_machine2.svg

What can we do with a function?

- Give it its argument.

What can we do with a function?

- Give it its argument.
- Look at its result.

What can we do with a function?

- Give it its argument.
- Look at its result.
- ...

What can we do with a function?

- Give it its argument.
- Look at its result.
- ...
- ...

What can we do with a function?

- Give it its argument.
- Look at its result.
- ...
- ...
- Maybe we need to think about this some more...

What is a function, Georg Cantor?

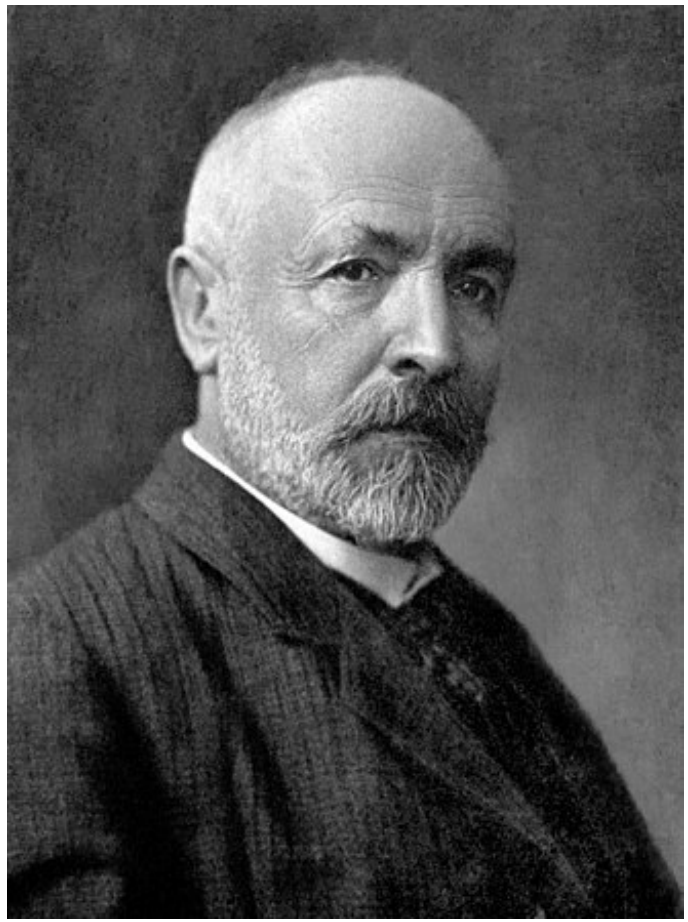

Figure: Georg Cantor (1845 - 1918)[1]

[1] Source https://en.wikipedia.org/wiki/Georg_Cantor#/media/File:Georg_Cantor_(Portr%C3%A4t).jpg

Sets

- Sets are concerened with collections of discrete mathematical objects.
  e.g $A = \{1, 2, 3, ...\}$.

Sets

- Sets are concerened with collections of discrete mathematical objects.
  e.g $A = \{1, 2, 3, ...\}$.
- The order of elements does not matter.

Sets

- Sets are concerened with collections of discrete mathematical objects.
  e.g $A = \{1, 2, 3, ...\}$.
- The order of elements does not matter.
- Each element my appear only once.

Sets

- Sets are concerened with collections of discrete mathematical objects.
  e.g $A = \{1, 2, 3, ...\}$.
- The order of elements does not matter.
- Each element my appear only once.
- Given an set X and an index i we may chose $x_i \in X$. Think of this like a function $f : (Set(X), \mathbb{N}) \to X$.
  e.g $f(A, 2) = 2$.

Sets

- Sets are concerened with collections of discrete mathematical objects.
  e.g $A = \{1, 2, 3, ...\}$.
- The order of elements does not matter.
- Each element my appear only once.
- Given an set X and an index i we may chose $x_i \in X$. Think of this like a function $f : (Set(X), \mathbb{N}) \to X$.
  e.g $f(A, 2) = 2$.
- See [Halmos, 1960] for more!

Relations

Functions

What can we do with set-theoretic functions?

What can we do with functions, Alonzo Church?

Figure: Alonzo Church (1903 - 1995)[1]

[1] Source https://en.wikipedia.org/wiki/Alonzo_Church#/media/File:Alonzo_Church.jpg

The Untyped Lambda Calculus

The Grammar for LC.

$$
\begin{array}{rr}
M, N ::= x, y, z, ... & \text{Variables.} \\
\mid \lambda x.N & \text{Abstraction.} \\
\mid M\ N & \text{Application}
\end{array}
$$

An Idris Example (some liberties with types)
$\lambda x.\neg x$ True

```
module Lambda
f : Bool -> Bool
f = \x => not x

b : Bool
b = f True
```

What can we do with lambda calculi functions?

Functions are boring!

Can we apply any function to any argument?

Clearly we cannot, saying $(3 < 7) + 12$ does not make sense, nor does $(-4) \vee 7$.
Operations are often defined as being closed under a particular set.
Closures are really speaking about the type of operations.

Are there other ways we can catgorise functions besides the type of their input and output?

We can also classify functions based on properties of their behaviour. Sometimes we don't care what specific types a function operates on, so long as there is some valid operation defined for those types.

Diagrams of types.

Diagrams of behavioural properties.

To functional programming

Question for the pub!

Why are functions the default 'building-blocks' in most programming languages? Why not relations?

Is everything really a function? (Really?)

The argument goes that a pure functional language is just an elaboration of some variant of a typed lambda calculus. Let's briefly investigate this.

Values as functions.

Are these functions the same?

```
module Values
%default total

seven :: Int
seven = 7

seven' :: () -> Int
seven' = \x => 7
```

Values as functions.

Are these functions the same?

```
module Values
%default total

seven : Int
seven = 7

seven' : () -> Int
seven' = \x => 7
```

The humble unit type.

The unit type is special. Denoted `()` or $\top$, the unit type is defined as having only one single element, also `()` or sometimes $*$. The unit element can *always* be constructed. This implies that `seven` is the same as `seven'` as we can always construct the argument to `seven'`.

Data types as functions.

Can we do the same with data types?

We can see this with church encoding, and have a small example below, interested readers should see [Pierce, 2002, p. 58-68].

Data types as functions.

Can we do the same with data types?

We can see this with church encoding, and have a small example below, interested readers should see [Pierce, 2002, p. 58-68].

Church encoding of the Boolean type

```
module ChurchBools
data Boolean : Type where
  True : Boolean
  False : Boolean

true : a -> a -> a
true = \t => \f => t

false : a -> a -> a
false = \t => \f => f
```

Algerbraic Data Types

Sum Types

The sum type is analogous to disjoint union in set theory. $A + B$ can be construct from either $x \in A$ or $y \in B$ provided we claim which set was the original one.

Product Types

References

P. R. Halmos. *Naive set theory*. van Nostrand, 1960.

B. C. Pierce. *Types and programming languages*. MIT press, 2002.