MFCS

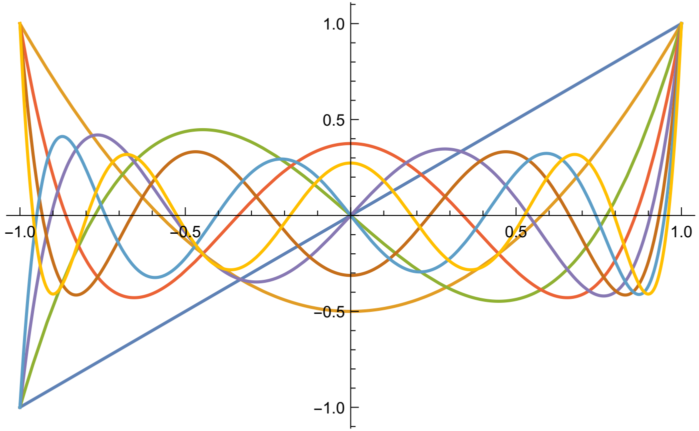
Functions

Klaus Sutner

Carnegie Mellon University

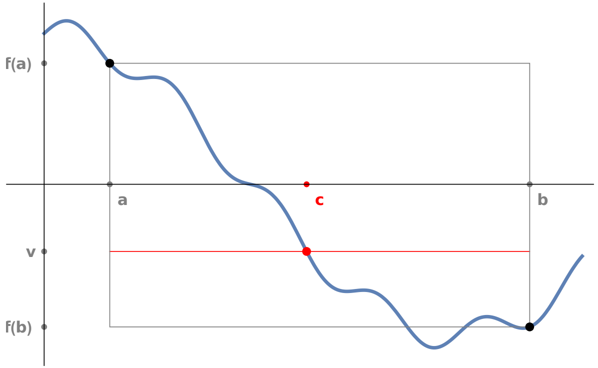
Fall 2022

Pretty Pictures



Legendre polynomials you might know from calculus.

IVT



A beautiful picture of some rando continuous function. Supposedly this picture is useful to understand why the *intermediate value theorem* holds.

Too Narrow

Unfortunately, this concept of a function familiar from calculus is far too narrow for our purposes. We need to work a bit harder and figure out what the essential ideas behind the concept of a function really are.

First off, we can take inspiration from computable functions, functions that can be described by an algorithm.

To specify such a function, we need three pieces of information:

- the type of the allowed inputs,
- the type of the corresponding outputs,
- a mechanical method to get from input to output.

Abstraction

Input/output types are a perfectly good general idea, that we will steal. Alas, requiring all functions to be defined by an algorithm is not terribly useful.

For us, the "method" can be as general as imaginable: it can be just a collection of input/output pairs.

More precisely, suppose A and B are two sets. We want to define functions from A to B. To this end, consider some input/output pairs:

$$F \subseteq A \times B$$

When would F make sense as a function?

Key Ideas

We need two key properties.

Totality	For every $x \in A$, there is a $y \in B$ such that $(x, y) \in F$.
Single-Valuedness	If $(x, y), (x, z) \in F$, then $y = z$.

In other words, we want the function to be defined on all of A[†], and we want the output to be unique.

That's all, folks.

[†]Actually, it does make sense to consider so-called partial functions that may not be defined on all of A, but not right now.

Formal Definition

Definition

A function (or map, mapping) from A to B is a triple (F, A, B) where $F \subseteq A \times B$ is total and single-valued.

We write $f : A \to B$ for a function from A to B and call A the domain of f, and B its codomain. $F = \operatorname{grf} f$ is called the graph of the function[†].

f is an endofunction if $\operatorname{dom} f = \operatorname{cod} f$.

We write $f(a) = b$ instead of $(a, b) \in F$.

We write $A \to B$ or B^A for the collection of all functions from A to B.

[†]In set theory, a function is often defined to be its graph. That won't work for us.

Example: Legendre Polynomial #5

For P_5 we have:

- the domain is $\mathbb{R}$
- the codomain is $\mathbb{R}$
- the graph is

$$\{ (x, (15x - 70x^3 + 63x^5)/8) \mid x \in \mathbb{R} \}$$

So this function is “computable” in a way (once you have figured out what it means to compute with reals, currently more or less an open problem).

Notation

Usually we do not utilize the triple notation for functions, it's a bit too fusty. Instead we write things like

$$f : A \to B \qquad f(x) = blahblahblah$$

If domain and codomain are completely obvious from context, we may get lazy and write

$$f(x) = blahblahblah$$

It's best to avoid this for a while, though, to get used to specifying domain and codomain precisely.

Forward

When $b = f(a)$, one often calls b the image of a under f.

Definition

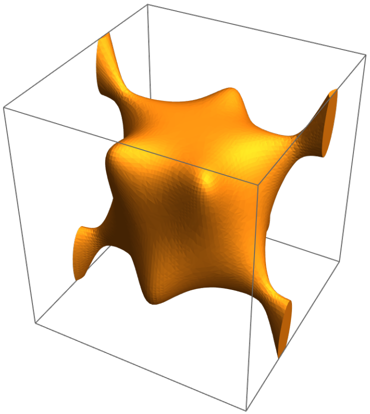
Consider a function $f : A \to B$. The image of $X \subseteq A$ under f is

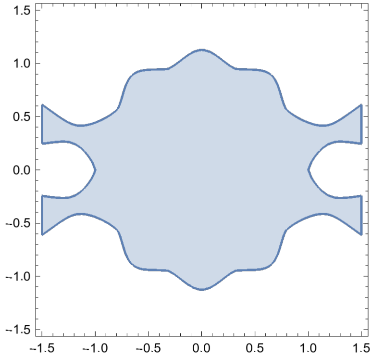
$$f(X) = \{\, f(x) \mid x \in X \,\}$$

The range or image of f is $\operatorname{rng} f = f(\operatorname{dom} f)$.

For example, $\sin : \mathbb{R} \to \mathbb{R}$ has range $[-1, 1]$, $\exp : \mathbb{R} \to \mathbb{R}$ has range $\mathbb{R}_+$ and $\log : \mathbb{R}_+ \to \mathbb{R}$ has range $\mathbb{R}$.

What is f?





Notation

If you want to be very precise, you can introduce a new function

$$\widehat{f} : \mathfrak{P}(A) \to \mathfrak{P}(B) \qquad \widehat{f}(X) = \{\, f(x) \mid x \in X \,\}$$

In other words, we could keep track of the change in domain and codomain and change the symbol for the function accordingly.

In programming this is critical, code expecting input of type A will not accept input of type $\mathfrak{P}(A)$.

But in math and TCS it's up to the reader to figure out what exactly is meant by f[†]. You are supposed to adjust the types depending on context.

[†]This tends to drive beginners nuts, but in the end it is the only feasible approach. Otherwise you drown in symbols

Backward

Definition

Consider a function $f : A \to B$.
The preimage or fiber of $b \in B$ under f is

$$f^{-1}(b) = \{\, a \in A \mid f(a) = b \,\}$$

The preimage of $X \subseteq B$ under f is $f^{-1}(X) = \bigcup_{b \in X} f^{-1}(b)$.

Warning: f^{-1} is **not** a function $B \to A$, it has the type $B \to \mathfrak{P}(A)$ or $\mathfrak{P}(B) \to \mathfrak{P}(A)$[†].

[†]But see bijections below.

Example

Consider $\sin : \mathbb{R} \to \mathbb{R}$.

For $|x| > 1$, the fibers $\sin^{-1}(x)$ are all empty.

The fiber $\sin^{-1}(0)$ is $\mathbb{Z}\pi$.

And $\sin^{-1}(\mathbb{R}_{\geq 0})$ is $[0, \pi] + 2\mathbb{Z}\pi$.

Functions versus Sets

Suppose we have some ambient set $\mathcal{U}$ and we need to deal with subsets $A \subseteq \mathcal{U}$. We can express these subsets in terms of functions.

Definition

The characteristic function of $A \subseteq \mathcal{U}$ is defined by $\chi_A : \mathcal{U} \to \mathbf{2}$ where

$$\chi_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise.} \end{cases}$$

Here we write $\mathbf{2} = \{0, 1\}$.

We will shamelessly identify 0 with False, and 1 with True, whenever convenient. Or we think of them as integers, it all depends[†].

[†]This kind of skulduggery drives type theory purists nuts.

Bitvectors

So χ_A is just a bitvector that answers membership questions.

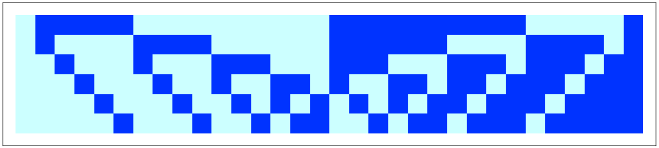
In particular when $\mathcal{U} = [n]$ and n is reasonably small, this provides an excellent implementation for subsets.

Consider the universe $\mathcal{U} = [10]$ and $A = \{2, 3, 6, 8\}$, then

```
bool  A[] = {0,0,1,1,0,0,1,0,1,0}
```

For $n = 64$ this means we can perform some operations in one step.

Picture



Bitvectors for all even-cardinality subsets of $[6]$.

Set Operations

Characteristic functions translate set operations into logical connectives:

$$\begin{aligned}
\chi_{A\cup B} &= \chi_A \vee \chi_B \\
\chi_{A\cap B} &= \chi_A \wedge \chi_B \\
\chi_{A\oplus B} &= \chi_A \oplus \chi_B \\
\chi_{A^c} &= \neg\chi_A
\end{aligned}$$

Here all the logical operations are supposed by applied pointwise: $\chi_{A\cup B}(x) = \chi_A(x) \vee \chi_B(x)$.

Composition of Functions

The most important property of functions is that they can be composed, we can execute one after the other. Given

$$f : A \to B \qquad g : B \to C$$

Definition

The composition of f and g, in symbols $g \circ f$, is given by

$$h : A \to C \qquad h(x) = g(f(x))$$

Note that the codomain of f and the domain of g has to match, otherwise composition is undefined.

A Pipe Dream

In 1976, the category theorist Charles Wells wrote[†].

> Observe that I write functions on the right and functional composition from left to right. This is undoubtedly the *Wave of the Future*. It makes functional diagrams easier to read and corresponds to the natural order of doing things on a pocket calculator.

Hewlett-Packard was no doubt pleased with Wells's comments: their calculators used postfix notation: argument 1, argument 2, operator. Works much better than infix.

Alas, nothing happened, zilch, nada, null.

[†]In a paper titled "Some Applications of the Wreath Product Construction," no less.

But Why?

There is a clash between composition of functions and diagrams

$$A \xrightarrow{f} B \xrightarrow{g} C$$

Reading from left-to-right we get $f \circ g$, but according to our definition this is $g \circ f$[†].

Also, the standard convention for sequential composition of programs is $F;G$: first run F, the run G.

Alas, the composition convention is firmly entrenched, we won't fight.

[†]Worse, it clashes with relational composition, see there.

Iteration

Definition

Let $f : A \to A$ be an endofunction. The kth power of f (or kth iterate of f) is defined by induction as follows:

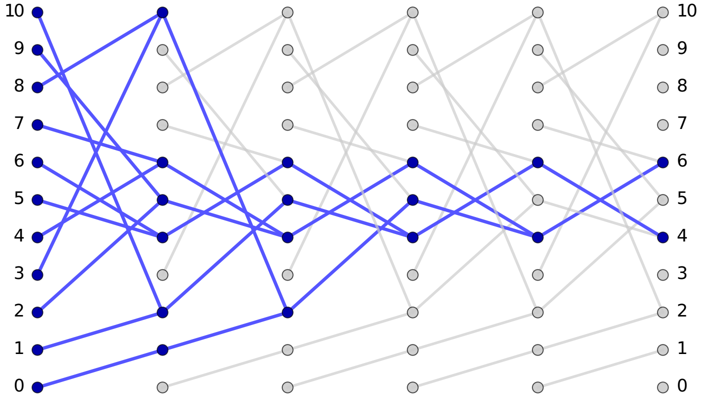
$$f^0 = I_A$$
$$f^k = f \circ f^{k-1}$$

Here I_A denotes the identity function on A.

Informally, this just means: compose function f $(k-1)$-times with itself.

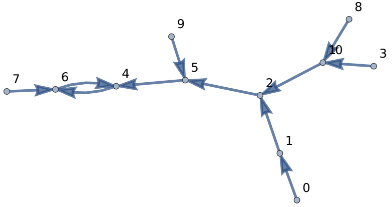
$$f^k = \underbrace{f \circ f \circ f \circ \ldots \circ f}_{k \text{ terms}}$$

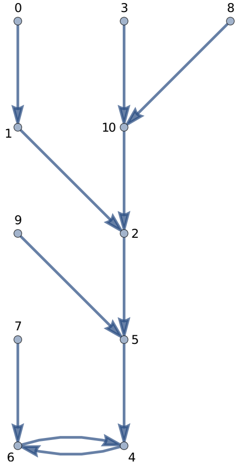
Example 1



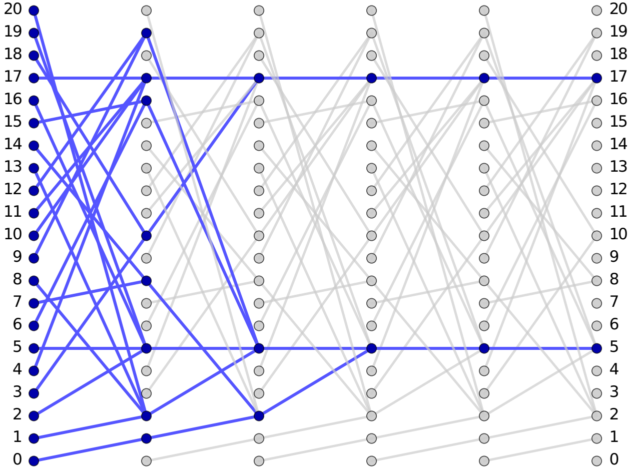
Iterating the map $x \mapsto x^2 + 1 \bmod 11$.

Phasespace



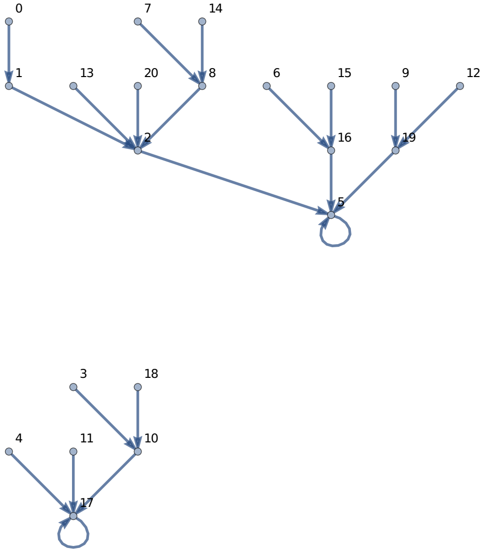


Example 2



Iterating the map $x \mapsto x^2 + 1 \bmod 21$.

Phasespace



General Laws

Without any further knowledge about f, there is not much one can say about the iterates f^k. But the following always holds.

Lemma (Laws of Iteration)

- $f^n \circ f = f^{n+1}$
- $f^n \circ f^m = f^{n+m}$
- $(f^n)^m = f^{n \cdot m}$

This should look very familiar ...

Binary Expansions

A point $a \in A$ is a fixed point of f if $f(a) = a$.

Here is an example. Write $a :: L$ for the list L with element a prepended and define the following operation on numbers and binary lists.

$$\begin{aligned}
&\beta : \mathbb{N} \times \mathsf{List}(\mathbf{2}) \to \mathbb{N} \times \mathsf{List}(\mathbf{2}) \\
&\beta(0, L) = (0, L) \\
&\beta(x, L) = (x \;\mathsf{div}\; 2, (x \bmod 2) :: L)
\end{aligned}$$

Now suppose we have an operator FP that, given β and a, looks for a fixed point among the iterates $a, \beta(a), \beta(\beta(a)), \ldots$

Of course, there may be none, just think about the first example from above.

Done

In our situation, a fixed point always exists (why?) and we have

$$\mathsf{FP}(\beta, (x, \mathsf{nil})) = (0, \mathrm{bin}(x))$$

where $\mathrm{bin}(x)$ is the binary expansion of x.

Thinking in terms of fixed point often produces very elegant descriptions of computations, often one-liners:

$$\mathsf{tobin}(x) = \mathsf{last}\big(\mathsf{FP}(\beta, (x, \mathsf{nil}))\big)$$

Exercise

Exercise

In your favorite programming language, implement a fixed point operation. Make sure to handle non-existent fixed points intelligently. Then implement the conversion to binary.

Exercise

Fixed points are quite important in calculus, for example in Newton's method. Typical application: to compute $1/a$ *where* $0 \neq a \in \mathbb{R}$ *we can find a fixed point of* $g(x) = 2x - ax^2$*.*
How does one have to modify the fixed point operator for this to make sense in an actual commputation?