CDM

Functions

Klaus Sutner
Carnegie Mellon University

40-functions 2017/12/15 23:22

1 Defining Functions

- Classification
- Orbits
- Closures
- Higher-Order Functions

Classical Functions

In calculus, functions are usually defined like so:

$$f(x) = x^2 + \sin(x)$$
$$g(x,y) = \frac{x^2 + 3y^2}{x - y}$$

There is an expression, which we can construe as a simple program, that determines the output value $f(x)$ given x as input. Note that the user has to figure out the proper input values. E.g., $x \neq y$ is required for g.

This intensional style is very close to computer science: to define a function we specify a method to compute its values. Note that we have to to stretch the notion of computation a bit, $\sin(x)$ is not computable in the narrow sense.

Functions à la Dirichlet

Alas, this approach to functions is too narrow. It turns out to be better to abstract away from the computational process and to focus on the input/output behavior alone.

Definition

Let $\rho : A \to B$ be a relation. ρ is a function (or map, mapping) if

- ρ is single-valued: $x\ \rho\ u \wedge x\ \rho\ v \Rightarrow u = v$.
- ρ is total: $\forall x \in A\, \exists y \in B\, (x\ \rho\ y)$.

A relation that is only single-valued is also called a partial function.

Notation:

- It is customary to write $f(x) = y$ rather than $x\ f\ y$.
- $A \to B$ is the set of all functions from A to B.
- This is also written B^A or $\mathrm{Fct}(A, B)$.

I/O

This is the input/output relation definition; it is very general and relatively new (1830), and it is purely extensional: a function is just a set of pairs (input,output), but we don't necessarily have any way to compute the output given the input (even if we a generous about what constitutes computability and admit things like $\sin(x)$).

Indeed, one of the strengths of this definition is that it allows for non-computable functions.

This may sound bizarre from the computer science perspective, but it turns out to be a much more useful platform that the more narrow definition.

Note that the domain is entirely unconstrained. In particular we can also accommodate k-ary functions of the form

$$f : A_1 \times \ldots \times A_k \to B.$$

Currying

In a sense, k-ary functions for $k > 1$ are superfluous: we can always rewrite a k-ary function in terms of unary ones.

The main idea is that

$$f : A \times B \to C$$

can also be construed as

$$F : A \to (B \to C)$$

where $F(a)(b) = f(a, b)$.

Still, it is useful and more natural to have both options. Also note that many programming language have great difficulties with the second version; functions tend not to be first class citizens.

Cardinality

The notation B^A for the collection of all functions from A to B is quite natural in the light of the following lemma.

Lemma

The cardinality of B^A is $|B|^{|A|}$.

Of course, for this to make sense we first have to explain cardinal arithmetic, in particular exponentiation κ^λ; see the section on cardinality for a proof.

For example, there are uncountably many functions $\mathbb{N} \to \mathbb{B}$, i.e., infinite binary sequences. For finite sets we get ordinary exponentiation.

Mental Health Warning: Insane Notation

Since functions are just special relations, composition of relations also makes sense for functions. Given two functions $f : A \to B$ and $g : B \to C$ we can form the relational composition $f \bullet g$.

Unfortunately, this is traditionally written backwards as $g \circ f$ (read: g after f). Thus, in most texts

$$(f \bullet g)(x) = (g \circ f)(x) = g(f(x)).$$

This notation does make some (minor) sense if function application is written on the left, but it violates diagrammatic order and, of course, directly clashes with relational composition. So, for the sake of sanity, we will write things the natural way:

$$(f \bullet g)(x) = (f \circ g)(x) = g(f(x)).$$

Incidentally, in computer science you will also find notations such as $f;g$ to indicate sequential composition.

Recall: Relations

Recall that a relation ρ from A to B is a subset of $A \times B$.

Here A is the domain of the relation and B its codomain.

The support (or domain of definition) of ρ is the set

$$\mathsf{supp}(\rho) = \{\, a \in A \mid \exists\, b \in B\, a\ \rho\ b \,\} \subseteq A$$

Dually, the range or image of ρ is the set

$$\mathsf{rng}(f) = \{\, b \in B \mid \exists\, x \in A\, a\ \rho\ b \,\} \subseteq B$$

More Mischief

For a function $f : A \to B$ the domain of definition is all of A, the same as its domain, and is given as part of the specification.

However, determining the range can be difficult. For example, for a multivariate integer polynomial the (undecidable) question of whether the polynomial has any integer roots is the same as asking whether the range of the corresponding function contains 0.

Note that some texts confuse the domain with the domain of definition, and the codomain with the range. Always check the definition.

The Graph of a Function

Definition

The graph of a function $f : A \to A$ is the relation

$$\mathsf{grf}(f) = \{\, (a, b) \mid f(a) = b \,\}$$

Thus the graph is the collection of all input/output pairs of f.

This may seem a bit bizarre, but there is method to the madness: according to our definition the specification of a function involves not just the graph but also the domain and codomain.

The domain of f can be retrieved from $\mathsf{grf}(f)$ but not the codomain. For partial functions not even the domain can be obtained.

Extension versus Intension

If we follow Dedekind and specify a function as its graph, i.e., a set of pairs, possibly augmented by domain and codomain we lose anything resembling the ability to compute the function value $f(x)$ given input x. This is the extensional view of functions.

Alas, in particular in computer science, it is often important to understand in a constructive sense how $f(x)$ and x are related.

Unfortunately, an intensional view of functions is quite a bit more complicated. For example, are $\lambda x.x + x$ and $\lambda x.2x$ intensionally the same? Even worse: is an intensional function the same as an algorithm?

Congruences

Definition

Let $\approx$ be an equivalence relation on A and $f : A \to A$. Then $\approx$ is a congruence for f if $x \approx y$ implies $f(x) \approx f(y)$.

The definition naturally generalizes to functions of arbitrary arity and to collections of functions.

Congruences are important in algebraic structures: taking the quotient with respect to a congruence (rather than an arbitrary equivalence) produces similar algebraic structures.

Converse and Inverse

Another touchy point is the converse of a function, considered as a relation. Instead of f^c one often writes f^{-1}, and calls it the inverse.

That's OK, but realize that f^{-1} is in general not a function, just a relation.

E.g, $f = \{(1,1),(2,1)\}$.
Then $f^{-1} = \{(1,1),(1,2)\}$ and therefore not single-valued.

Exercise

When is f^{-1} again a function? Or at least a partial function? What role does the range play?

- Defining Functions
- Classification
- Orbits
- Closures
- Higher-Order Functions

Types of Functions

Functions in general are complicated, but there is a simple, general classification of functions that is eminently useful in many places.

The terminology here is due to Bourbaki and has become totally standard.

Definition

Let $f : A \to B$ be any function.

- f is surjective if $\forall\, y \in B\, \exists\, x \in A\, (f(x) = y)$,
- f is injective if $f(u) = f(v) \Rightarrow u = v$,
- f is bijective if f is injective and surjective.

Again

Surjective (onto) means every element in the codomain has a preimage.

Injective (one-one): no one in the codomain has more than one preimage. Another way of thinking about this is that no information is lost during the application of f: we can reconstruct x from $f(x)$, the operation is reversible.

Bijective: there is a perfect correspondence between the domain and the codomain. Bijections are the key element in cardinality arguments.

Example

Here are some real-valued functions familiar from calculus.

$x \mapsto x^2$	is neither injective nor surjective.
$x \mapsto x^3 - x$	is surjective but not injective.
$x \mapsto e^x$	is injective but not surjective.
$x \mapsto x^3$	is bijective (even an order isomorphism).

Characterizing Injectivity

Lemma

Let $f : A \to B$ be a function. The following are equivalent:

1. *f is injective.*
2. *$f \bullet f^c = I_A$.*
3. *$\exists\, F : B \to A\ (F \circ f = I_A)$.*
4. *f has the left-cancellation property:*
   $\forall\, g, h : C \to A\ (f \circ g = f \circ h \Rightarrow g = h)$.

The last property may seem a bit strange and unreasonably complicated, but it is particularly useful when one needs to generalize injectivity in an algebraic context.

Proof

Proof. (of lemma 1)

$(1 \to 2)$

$a\ (f \bullet f^c)\ b$ iff $f(a) = c = f(b)$. Since f is injective, iff $a = b$.

$(2 \to 3)$

Let $F = f^c$ (and recall the relationship between $\circ$ and $\bullet$).

$(3 \to 4)$

Let F, g and h as given. Then

$g \circ f = h \circ f$ implies $(g \circ f) \circ F = (h \circ f) \circ F$.

By associativity, $g \circ (f \circ F) = h \circ (f \circ F)$, $g = h$.

More Proof

$(4 \to 1)$

Pick $a, b \in A$ such that $f(a) = f(b)$.

Set $C = \{\diamond\}$ and let $g(\diamond) = a$, $h(\diamond) = b$.

Then $g \circ f = h \circ f$, hence $g = h$ and thus $a = b$.

□

For example, $f : \mathbb{R} \to \mathbb{R}$, $f(x) = e^x$ is injective since the function $F : \mathbb{R} \to \mathbb{R}$, $F(x) = \ln x$ for $x > 0$ and $F(x) = 0$, otherwise, has the property in part (3) of the lemma.

And Surjectivity

Lemma

Let $f : A \to B$ be a function. The following are equivalent:

1. *f is surjective.*
2. *$I_B \subseteq f^c \bullet f$.*
3. *$\exists F : B \to A\ (f \circ F = I_B)$.*
4. *f has the right-cancellation property:*
   $\forall g, h : B \to C\ (g \circ f = h \circ f \Rightarrow g = h)$.

Exercise

Give a proof for lemma 9.

Decomposing Functions

How special are injective and surjective functions? According to the following theorem, not very.

Theorem

Every function is a composition of a surjective function and an injective function.

Proof.

Let $\rho = K_f$ be the kernel equivalence relation for $f : A \to B$. Define $g : A \to A/\rho$ and $h : A/\rho \to B$ by

$$g(x) = [x]_\rho$$
$$h([x]_\rho) = f(x)$$

Then $f = h \circ g$, h is injective, and g is surjective.

Touchy point: must check that h is well-defined. What if $[x]_\rho = [y]_\rho$?

□

Inverting Functions

- If $f : A \to B$ is bijective then $f^{-1} : B \to A$ is a function (even a bijection).
- If $f : A \to B$ is injective then $f^{-1} : \text{rng}(f) \to A$ is a function. Can be extended to a function $f^{-1} : B \to A$ by picking random values for points in $B - \text{rng}(f)$.

But for non-injective f, f^{-1} is not even a partial function, just a relation.

Nonetheless, in calculus, one often tries to find inverses for some restrictions of f.

Example

$f(x) = x^3$ has an inverse as a function $f : \mathbb{R} \to \mathbb{R}$.

$f(x) = x^2$ has no inverse as a function $f : \mathbb{R} \to \mathbb{R}$, but $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ has an inverse: $\sqrt{x}$.

Likewise, $\sin$ has infinitely many piecewise inverses.

Characteristic Functions

Consider subsets of some fixed universe $\mathcal{U}$.

Definition

The characteristic function of $A \subseteq \mathcal{U}$, $\chi_A : \mathcal{U} \to \mathbb{B}$ is

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

χ_A is just another representation of A, and sometimes more useful.

Characteristic function are referred to as bitvectors in computer science. Consider the universe $\mathcal{U} = \{0, \ldots, n-1\}$. Then $A = \{2, 3, 6, 8\}$ for $n = 10$ corresponds to

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

Set Operations

Set operations translate into logical connectives:

$$\chi_{A\cup B}(x) = H_{\text{or}}(\chi_A(x), \chi_B(x))$$
$$\chi_{A\cap B}(x) = H_{\text{and}}(\chi_A(x), \chi_B(x))$$
$$\chi_{A\Delta B}(x) = H_{\text{xor}}(\chi_A(x), \chi_B(x))$$
$$\chi_{A^c}(x) = H_{\text{not}}(\chi_A(x))$$

While bitvectors are simple they do provide a perfectly good implementation for small universes. Note that one can pack the bits and exploit bit-parallelism to speed up all the operations above Of course, for large universes this method fails miserably.

Exercise

Implement a bitvector library.

YABA

So, a subset of any fixed universe can be identified with a sequence of true/false values:

$$\mathfrak{P}(\mathcal{U}) \text{ corresponds to } \mathcal{U} \to \mathbb{B}$$

The logical operations on $\mathbb{B}$ can naturally be extended to sequences of true/false values: apply them point-wise to corresponding elements in the sequences. Note that point-wise operations on (short) bit sequences are useful enough to be incorporated in many programming languages such as C.

Hence it is not too surprising that

$$\langle \mathfrak{P}(\mathcal{U}), \cup, \cap, ^-, \emptyset, \mathcal{U} \rangle$$

is yet another Boolean algebra, we a just dealing with vectors of Boolean values.

Functions and Equivalence Relations

Definition

Let $f : A \to B$ be a function. Define the kernel relation of f by

$$x \,\rho\, y \iff f(x) = f(y)$$

The kernel relation is usually written K_f or $\ker(f)$.

It is easy to check that ρ is an equivalence relation on A.

Example

Let A be the set of all polygons, $f(x)$ the area of x. Yields the "same-area" relation K_f.

Let $f : \mathbb{Z} \to \mathbb{Z}$, $f(x) = x \bmod m$. Then K_f is congruence modulo m.

Kernels Everywhere

Question: Could it be that every equivalence relation is already a kernel relation?

In other words, given an arbitrary equivalence relation ρ on A, can we produce a function $f : A \to B$ such that $\rho = K_f$.

If we don't care much about the codomain this is easy:

$$\begin{array}{rccc} f: & A & \longrightarrow & \mathfrak{P}(A) \\ & x & \longmapsto & [x] \end{array}$$

Better Kernels

Theorem

Every equivalence relation ρ on A is a kernel relation. In fact, we can choose a function $f : A \to A$ such that $K_f = \rho$.

Proof.

We have to produce a function $f : A \to A$ such that $\rho = K_f$.

For each equivalence class $[x]_\rho$ pick one representative $x_0 \in [x]_\rho$.

Set $f(z) = x_0$ for all $z \in [x]_\rho$.

□

A Minor Foundational Problem

But, there is a small problem: how do we actually pick $x_0 \in [x]$?

If $[x] \subseteq A$ is just an abstract set, there is no mechanism to select x_0.

As B. Russell pointed out, given an infinite collection of pairs of shoes one can select one from each pair; given an infinite collection of socks one is stumped.

Choice

But, we have the celebrated Axiom of Choice.

Consider the partition $P \subseteq \mathfrak{P}(A)$ corresponding to ρ.

By (AC), there is a choice set C:

$$\forall\, X \in P\,(|X \cap C| = 1).$$

Hence we can define

$$f(x) = y \qquad \text{iff} \qquad x\ \rho\ y \wedge y \in C$$

It is easy to check that f really is a function and that $\rho = K_f$.

Well-Orders

Alternatively, we can follow Zermelo and use (AC) to construct a well-order $\prec$ of A: for each non-empty subset X of A we can use choice to pull out an element which we will declare to be the $\prec$-minimal element of X.

But then we can define

$$f(x) = \min_{\prec} [x]_\rho$$

This idea is extremely useful in practice when $A = [n]$.

Kernel Schmernel

So who should worry about this general abstract nonsense? Admittedly, it seems that computer science does not require the Axiom of Choice, but the kernel idea provides a very nice data structure for equivalence relations.

Suppose $A = [n]$ for simplicity.

Then we can represent any equivalence relation ρ on A by a selector function $f : [n] \to [n]$, i.e., by a simple integer array of length n. For example, we could set

$$f(a) = \min\big(\, x \in [n] \mid x\ \rho\ a \,\big)$$

This is the canonical selector function.

Example

Congruence modulo 4 on $[10]$ produces

x	1	2	3	4	5	6	7	8	9	10
$f(x)$	1	2	3	4	1	2	3	4	1	2

Idempotent Deflationary Functions

Then

$$a\ \rho\ b \iff f(a) = f(b)$$

Note that this function has the following properties:

- idempotent: $f \circ f = f$
- deflationary: $f(x) \leq x$

On the other hand, any function with these properties defines an equivalence relation.

So we can implement equivalence relations in $\Theta(n)$ space with $O(1)$ lookup.

Exercise

Show that a function $f : [n] \to [n]$ is a canonical selector for some equivalence relation if, and only if, it is idempotent and deflationary.

- Defining Functions
- Classification
3. Orbits
- Closures
- Higher-Order Functions

Functional Digraphs

Recall that every binary relation on V can be represented by a digraph $G = \langle V, E \rangle$. What's special about these graphs when the relation is a function?

Definition (Degrees)

Let $v \in V$ be a vertex in a digraph G.
The indegree of v is $|\{\, u \in V \mid (u, v) \in E \,\}|$.
The outdegree of v is $|\{\, u \in V \mid (v, u) \in E \,\}|$.

Since the edges in a functional digraph are given by $(x, f(x))$, every vertex must have outdegree 1.

However, the indegree may vary from 0 to $|A|$.

Transient and Period

What happens if we trace a path in a functional digraph starting from some vertex a?

$$a, f(a), f(f(a)), \ldots, f^n(a), \ldots$$

Since f is single-valued, we can follow exactly one path.

If A is finite, then the path must ultimately wind up on a cycle (the limit cycle)

$$f^t(a) = f^{t+p}(a)$$

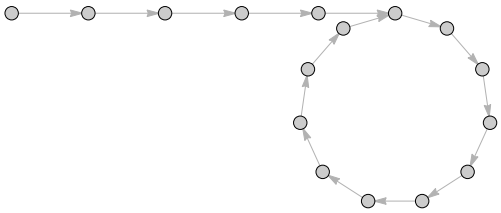
for some $t \geq 0$, $p > 0$, which depend on x.

Definition

The least t and p such that $f^t(a) = f^{t+p}(a)$ is the transient and the period of a (wrt. f).

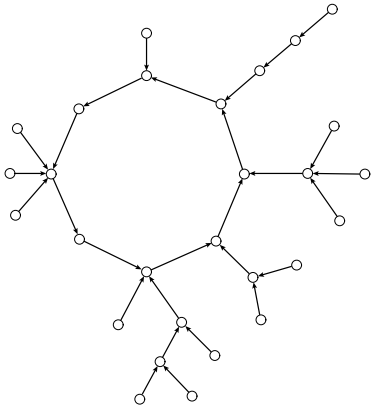
The Lasso

Tracing a function on a finite domain:

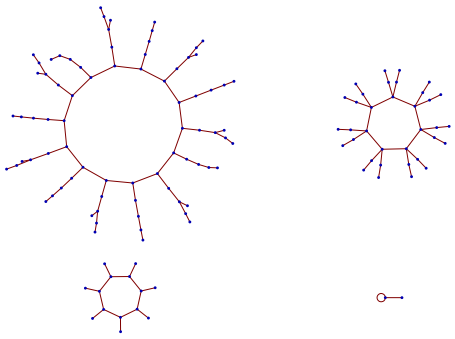


Note that we always get exactly this type of picture; the only thing that can change is the length of the handle and the cycle.

A Typical Limit Cycle



A Complete Diagram



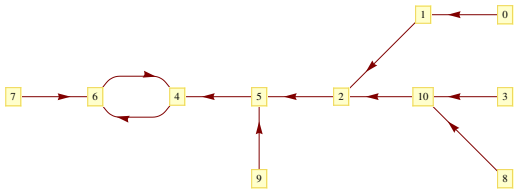
The whole diagram may have several limit cycles (this is ECA 74 on 7-bit configurations).

Example 1

Here is a simple example: the function

$$f(x) = x^2 + 1 \bmod 11$$

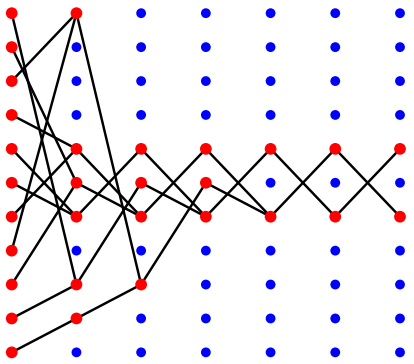
with domain and codomain $\{0, 1, \ldots, 10\}$. Here is the diagram:



There is one limit cycle of length 2, the maximum transient is 4.

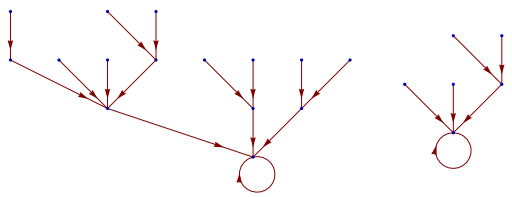
Another Picture

Iterating the function a few times provides another way to visualize the same function.



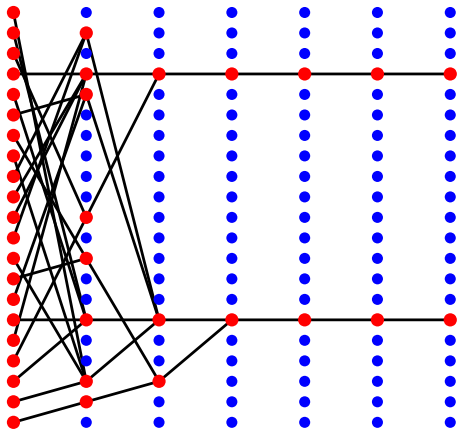
Example 2

Here is the same function $f(x) = x^2 + 1 \bmod 21$ but with modulus 21.



This time there are two fixed points, maximum transient is 3.

And Iterating



- Defining Functions
- Classification
- Orbits
4. Closures
- Higher-Order Functions

Closure

Let S be some ambient set and $f : S \to S$ on operation on S.

$X \subseteq S$ is f-closed if $f(X) \subseteq X$.

For a list of function $\boldsymbol{f} = f_1, \ldots, f_k$ we say that X is $\boldsymbol{f}$-closed if X is f_i-closed for all i.

The $\boldsymbol{f}$-closure of $X_0 \subseteq S$ is the least X such that $X_0 \subseteq X$ and X is $\boldsymbol{f}$-closed.

We write $\mathsf{cl}_{\boldsymbol{f}}(X_0)$ or $\mathsf{cl}(X_0, \boldsymbol{f})$.

In terms of set-theory we have

$$\mathsf{cl}(X_0, \boldsymbol{f}) = \bigcap \{\, Y \mid X_0 \subseteq Y \wedge Y \,\boldsymbol{f}\text{-closed} \,\}$$

Examples

Example

$S = \mathbb{N}$, $x_0 = 0$, $f(x) = x + 1$.

Closure: $\mathbb{N}$.

Example

$S = \mathbb{Z}$, $x_0 = 0$, $f_1(x) = x + 1$, $f_2(x) = x - 1$.

Closure: $\mathbb{Z}$.

Example

$S = \mathbb{N}$, $x_0 = 0$, $f_1(x) = x + a$, $f_2(x) = x + b$ where a, b coprime.

Closure: $F \cup \{\, x \in \mathbb{N} \mid x \geq (a-1)(b-1) \,\}$ where F is finite.

Exercise

$S = \mathbb{N}$, $f_1(x) = \lfloor x/2 \rfloor$, $f_2(x) = \lceil x/2 \rceil$.

Determine $\mathsf{cl}(x_0, \boldsymbol{f})$.

Clearly,

$$\mathsf{cl}(2^k, \boldsymbol{f}) = \{0, 1, 2, 4, 8, \ldots, 2^k\}$$

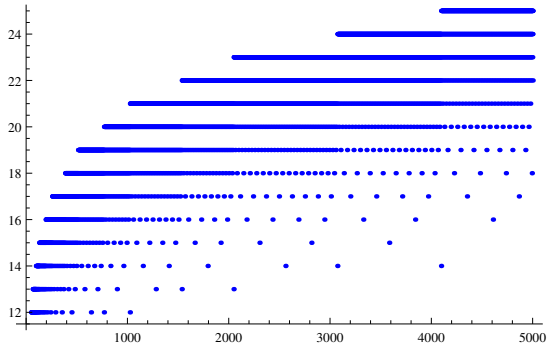
but other values are not so clear. For example,

$$\mathsf{cl}(560, \boldsymbol{f}) = \{0, 1, 2, 3, 4, 5, 8, 9, 17, 18, 35, 70, 140, 280, 560\}$$

and

$$\mathsf{cl}(561, \boldsymbol{f}) = \{0, 1, 2, 3, 4, 5, 8, 9, 17, 18, 35, 36, 70, 71, 140, 141, 280, 281, 561\}$$

Here is a plot of the size of these sets:



Fixed Points

The set-theoretic description of the closure is impredicative and thus not particularly useful from an algorithmic perspective. We can express the problem as a fixed point computation below and stops when the least fixed point of the operation Consider the operation

$$\Phi : Y \mapsto Y \cup \bigcup f_i(Y)$$

Then the closure X of X_0 is the least fixed point of Φ containing X_0.

But we can approximate the closure from below by starting at X_0 and applying Φ till a fixed point is reached:

$$\mathsf{cl}(X_0; \boldsymbol{f}) = \bigcup_{k \geq 0} \Phi^k(X_0)$$

Closure Algorithms

Here is a prototype algorithm along these lines.

Let us say that $x \in S$ requires attention (via f_i) if $x \in X$ but $f_i(x) \notin X$.

```
X = X0
while( some x requires attention via fi )
       add f(x) to X
```

For a real algorithm (at least for the finite case), we need to explain what data structures are used and how we search for an element requiring attention.

The good news is that this is old hat: this is precisely the the problem of graph exploration.

The graph in question here has $[k]$-labeled edges

$$u \xrightarrow{i} f_i(u)$$

and we are trying to find the points reachable from X_0.

The only difference with standard algorithms such as DFS and BFS is that we don't have an explicit adjacency list representation, we compute edges when we need them.

Spanning Trees

Closure algorithms organized like DFS or BFS produce a spanning tree in the subgraph reachable from X_0:

whenever u requires attention via f_i we compute $v = f_i(u)$.

If v has not yet been encountered we place the edge $u \to v$ into the spanning tree.

If v has been encountered we can record an "equation" $u \cdot i = v$ (see the section on algebra).

Higher Arities

Note that our definitions of closure also make sense for functions of higher arity.

For example, given a binary function $f : S \times S \to S$ we can say that $X \subseteq S$ is f-closed if $f(X \times X) \subseteq X$.

- Defining Functions
- Classification
- Orbits
- Closures

5 Higher-Order Functions

Associated Functions

Suppose we have a function

$$f : A \to B$$

with domain A and codomain B.

We can naturally associate this function with several others that operate only indirectly on the elements of A.

More precisely, we can lift f to

- lists over A
- subsets of A
- subsets of B

Lists

Write $\mathsf{List}(X)$ for the collection of all sequences over a ground set X. Then f gives rise to a map

$$\mathsf{List}(f) : \mathsf{List}(A) \to \mathsf{List}(B)$$

defined by

$$\mathsf{List}(f)(\mathsf{nil}) = \mathsf{nil}$$
$$\mathsf{List}(f)(a :: x) = f(a) :: \mathsf{List}(f)(x)$$

This lifting has several good properties:

$$\mathsf{List}(I_A) = I_{\mathsf{List}(A)}$$
$$\mathsf{List}(f \circ g) = \mathsf{List}(f) \circ \mathsf{List}(g)$$

Power Sets

Likewise we obtain

$$\mathfrak{P}(f) : \mathfrak{P}(A) \to \mathfrak{P}(B)$$

defined by

$$\mathfrak{P}(f)(X) = \{\, f(a) \mid a \in X \,\}$$

Alternatively, we can go in the opposite direction:

$$\mathfrak{P}'(f) : \mathfrak{P}(B) \to \mathfrak{P}(A)$$

defined by

$$\mathfrak{P}'(f)(Y) = \{\, a \in A \mid f(a) \in Y \,\}$$

Examples

In practice, one often write f instead of $\mathsf{List}(f)$ and $\mathfrak{P}(f)$. For $\mathfrak{P}'(f)$ one finds the notation f^{-1}.

Example

Let $f : \mathbb{R} \to \mathbb{R}$, $f(x) = x^2$.
$f((0, 1, 2, 3)) = (0, 1, 4, 9)$
$f([-2, 2]) = f([0, 2]) = [0, 4]$.
$f^{-1}([0, 1]) = [-1, 1]$, $f^{-1}([-1, 1]) = [-1, 1]$, $f^{-1}([-2, -1]) = \emptyset$

Functions as Input

Another important class of functions takes other functions as input, These are sometimes called higher-order functions or functionals.

- Integration (or differentiation) of real functions

$$\int : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$$

- Composition

$$\circ : (B \to C) \times (A \to B) \to (A \to C)$$

- Binding

$$\mathsf{bind1st} : (A \times B \to C) \times A \to (B \to C)$$

- Map

$$\mathsf{map} : (A \to B) \times \mathsf{List}(A) \to \mathsf{List}(B)$$

- Iteration

$$\mathsf{iterate} : (A \to A) \times \mathbb{N} \to (A \to A)$$

Example: Fold

Here is a well-known example of a higher order function: fold. Suppose we have

- a function $f : A \times B \to A$
- an starting element $e \in A$

By recursion, define

$$\begin{aligned} \mathsf{fold}(f, e, .) &: \mathsf{List}(B) \to A \\ \mathsf{fold}(f, e, \mathsf{nil}) &= e \\ \mathsf{fold}(f, e, L :: b) &= f(\,\mathsf{fold}(f, e, L), b\,) \end{aligned}$$

Note that we have used induction on the right in this definition. For example,

$$\mathsf{fold}(f, e, (a, b, c, d)) = f(f(f(f(e, a), b), c), d)$$

But Why?

A surprising number of list operations can be defined easily in terms of fold.

$f(a, b)$	e	$\mathsf{fold}(f, e, .)$
$a + 1$	0	length
$\mathsf{prep}(a, b)$	nil	reverse
$\mathsf{app}(a, g(b))$	nil	map g
$a \vee [b = b_0]$	tt	search
$a + [b = b_0]$	0	count

Here $[\alpha = \beta]$ follows Knuth's convention: it is 1 if indeed $\alpha = \beta$, and 0 otherwise.

And, of course, in a Boolean context we interpret 0 as false, and 1 as true. Overloading is fun, grin and bear it.

Implementing Removal

Here is another example for the versatility of fold.

Suppose we wish to implement an operation that removes some element b_0 from a list.

We could use $e = \mathsf{nil}$ and

$$f(x, y) = \begin{cases} x & \text{if } y \neq b_0, \\ \mathsf{app}(x, y) & \text{otherwise.} \end{cases}$$

Exercise

Explain in detail how this implementation of removal works.

Exercise

Verify the claims in the table above.

Summary

- Functions are often associated with computation, but the standard definition focuses on the input/output behavior only.
- Basic properties of functions are extensions of similar properties of binary relations.
- Kernels and equivalence relations are closely related.
- Iteration of endofunctions produces orbits, a central concept in discrete dynamics.
- Higher order functions arise naturally and are supported by some programming languages.