

CDM

Functions

Klaus Sutner
Carnegie Mellon University
www.cs.cmu.edu/~sutner

Battleplan

- Functions, defined
- Basic Properties
- Kernel Relation
- Functional Digraphs

Defining 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 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 1.

Let $\rho : A \rightarrow B$ 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 xfy .
- $A \rightarrow B$ is the set of all functions from A to B .
- This is also written B^A on occasion.

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 are 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 than the more narrow definition.

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

$$f : A_1 \times \dots \times A_k \rightarrow B.$$

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

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

can also be construed as

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

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

We refer to the section on cardinality for a proof.

For example, there are uncountably many functions $\mathbb{N} \rightarrow \mathbb{B}$, 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 $f : A \rightarrow B$ and $g : B \rightarrow C$ we can form the relational composition $f \bullet g$.

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

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

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. Alas, it is well-established . . .

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

The Range of a Function

It is often important to single out the elements in the codomain of a function that are images of some element in the domain:

$$\text{rg}(f) = \{ y \in B \mid x \in A, y = f(x) \} \subseteq B$$

This set is called the *range* or *image* of f .

Note that the codomain of a function is simply 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 codomain with the range, always check the definition.

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

Classification

Types of Functions

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

Definition 2. *Let $f : A \rightarrow B$.*

- 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.

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.

Examples

Example 1. *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 2. *Let $f : A \rightarrow B$ be a function. The following are equivalent:*

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

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 \rightarrow 2)

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

(2 \rightarrow 3)

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

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

And Surjectivity

Lemma 3. *Let $f : A \rightarrow B$ be a function. The following are equivalent:*

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

Exercise 2. *Give a proof for lemma ??.*

Decomposing Functions

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

Theorem 1. *Every function is a composition of a surjective function and an injective function.*

Proof.

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

$$\begin{aligned}g(x) &= [x]_\rho \\h([x]_\rho) &= f(x)\end{aligned}$$

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

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

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

Likewise, \sin has infinitely many piecewise inverses.

Characteristic Functions

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

Definition 3. The **characteristic function** of $A \subseteq \mathcal{U}$, $\chi_A : \mathcal{U} \rightarrow \mathbb{B}$ 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 functions are referred to as **bitvectors** in computer science. Consider the universe $\mathcal{U} = \{0, \dots, n-1\}$. 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 3. *Implement a bitvector library.*

YABA

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

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

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

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

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

Functions and Equivalence Relations

Definition 4. Let $f : A \rightarrow B$ be a function. Define the **kernel relation** of f by

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

Usually written K_f .

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

Example 3.

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

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

Kernels Everywhere

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

Theorem 2. *Every equivalence relation ρ on A is a kernel relation. In fact, we can choose a function $f : A \rightarrow A$ such that $K_f = \rho$.*

Proof.

WE have to produce a function $f : A \rightarrow A$ 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 $z \in [x]_\rho$.

□

There are other ways to define f , but this solution is particularly simple.

A Minor Foundational Problem

But, there actually 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 .

But, we have the Axiom of Choice.

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

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

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

Define

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

Can check that f really is a function and that $\rho = K_f$.

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* $f : [n] \rightarrow [n]$, i.e., by a simple integer array of length n . For example, we could set

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

This is the *canonical selector function*.

Example 4. *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 Inflationary 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 4. *Show that a function $f : [n] \rightarrow [n]$ is a canonical selector for some equivalence relation if, and only if, it is idempotent and deflationary.*

Orbits

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 5. Degrees

Let $v \in V$ be a vertex.

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 ?

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

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*)

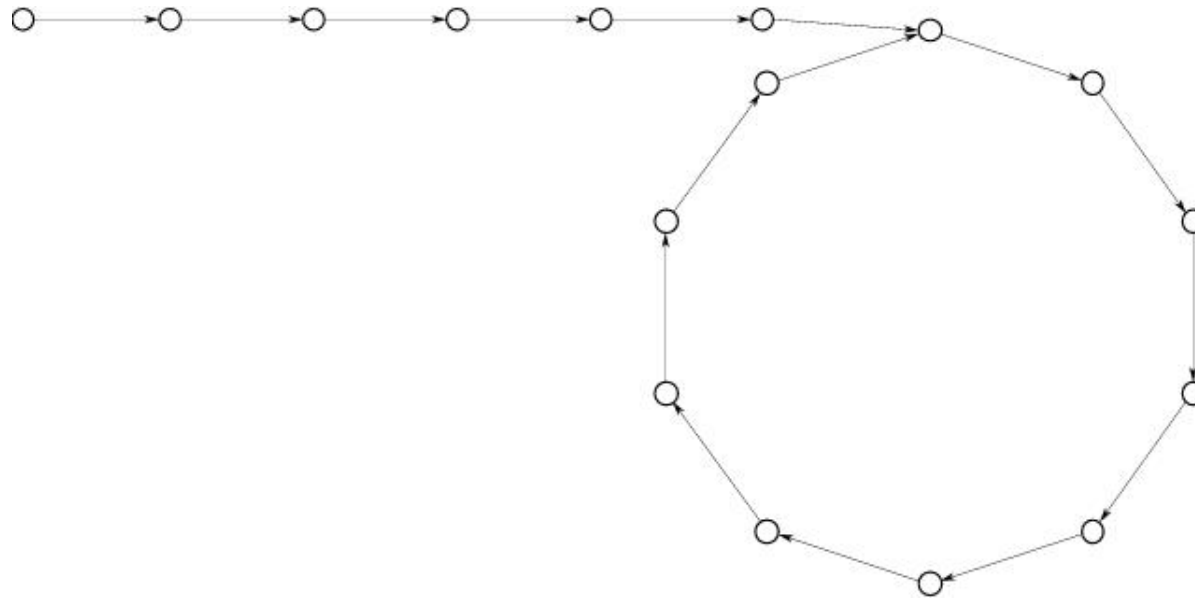
$$f^t(a) = f^{t+p}(a)$$

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

Definition 6. The least t and p such that $f^t(a) = f^{t+p}(a)$ is the **transient** and the **period** of a (wrt. f).

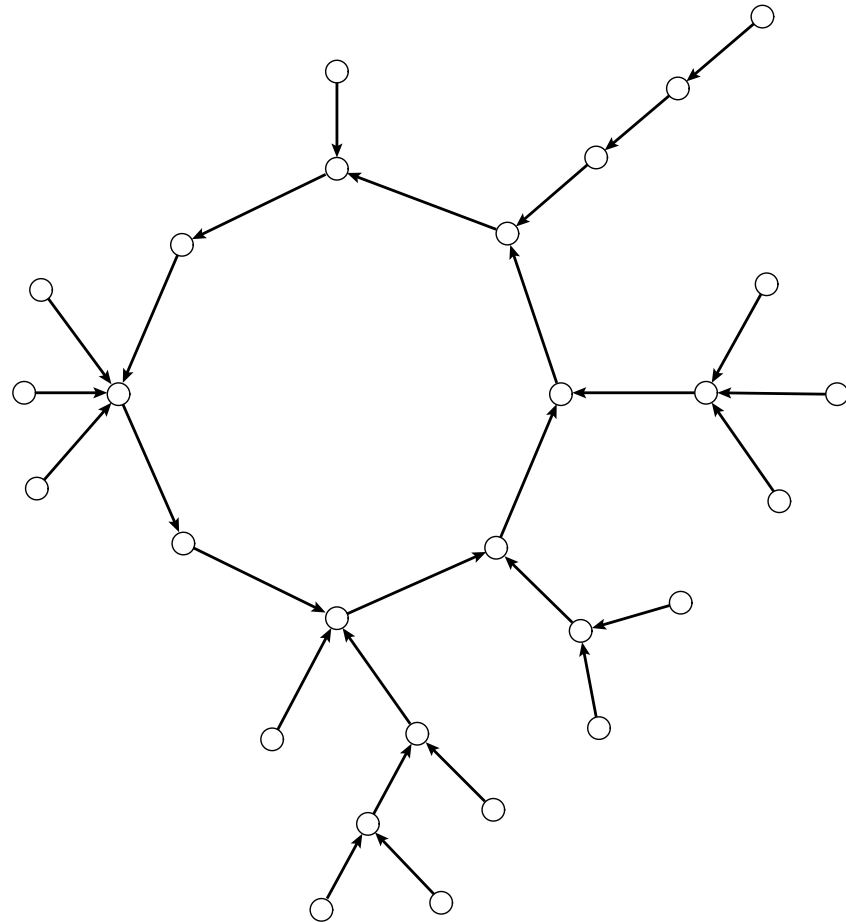
The Lasso

Tracing a function on a finite domain:



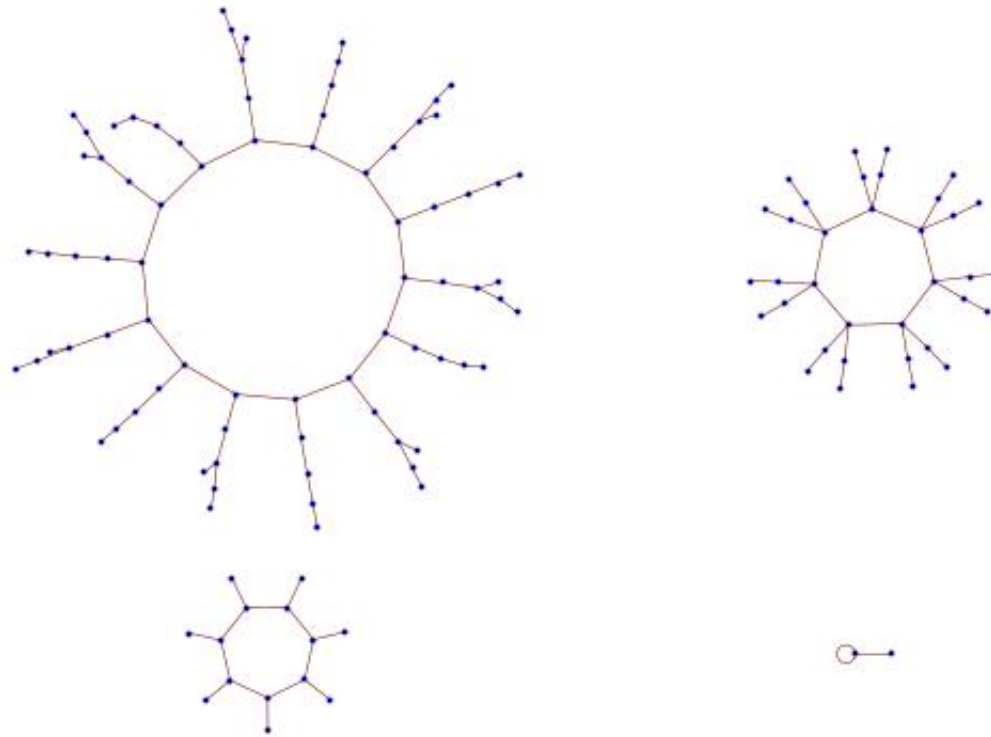
Note that we always get exactly this picture; the only thing that can change is the length of the handle and the cycle.

A Typical Limit Cycle



Several lassos can share the same limit cycle.

A Complete Diagram



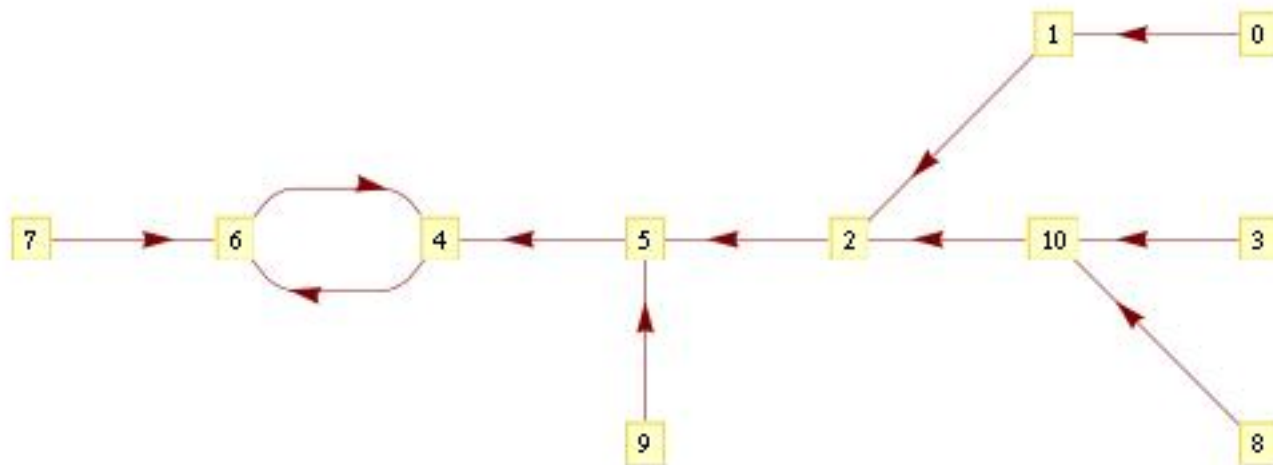
The whole diagram may have several limit cycles (this is ECA 74 on 7-bit configurations).

Example 1

Here is a simple example: the function

$$f(x) = x^2 + 1 \pmod{11}$$

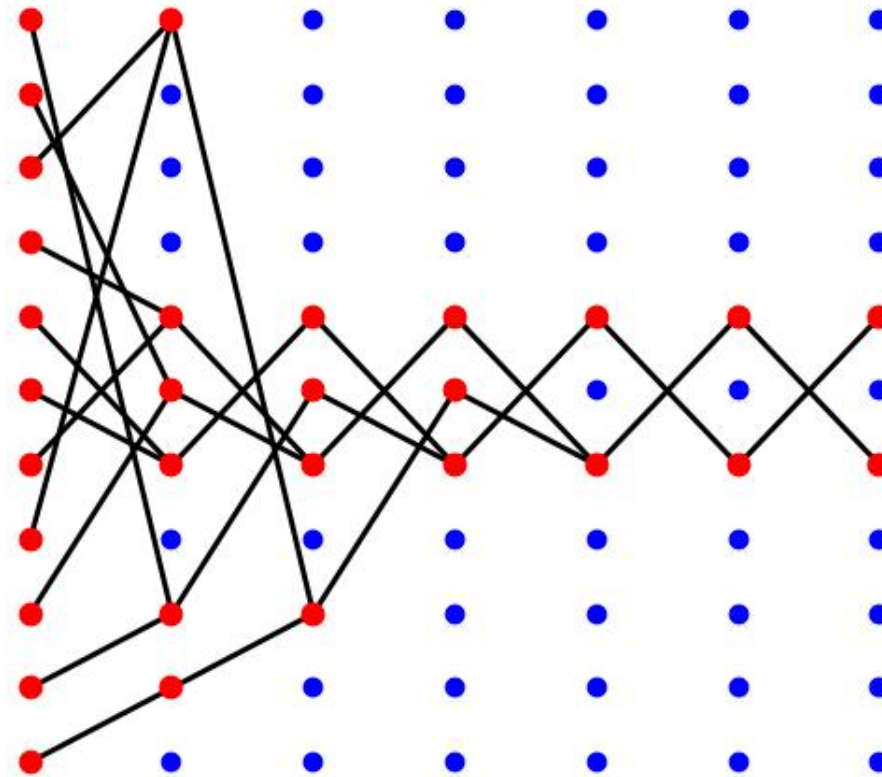
with domain and codomain $\{0, 1, \dots, 10\}$. Here is the diagram:



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

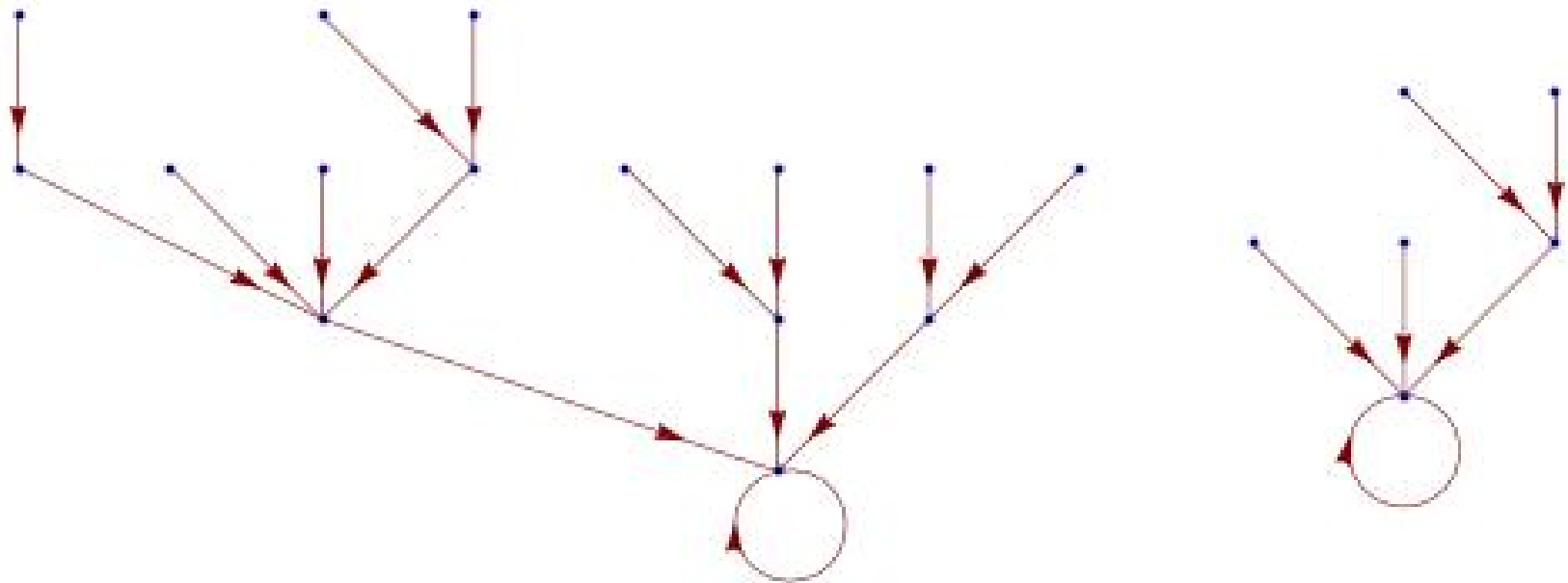
Another Picture

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



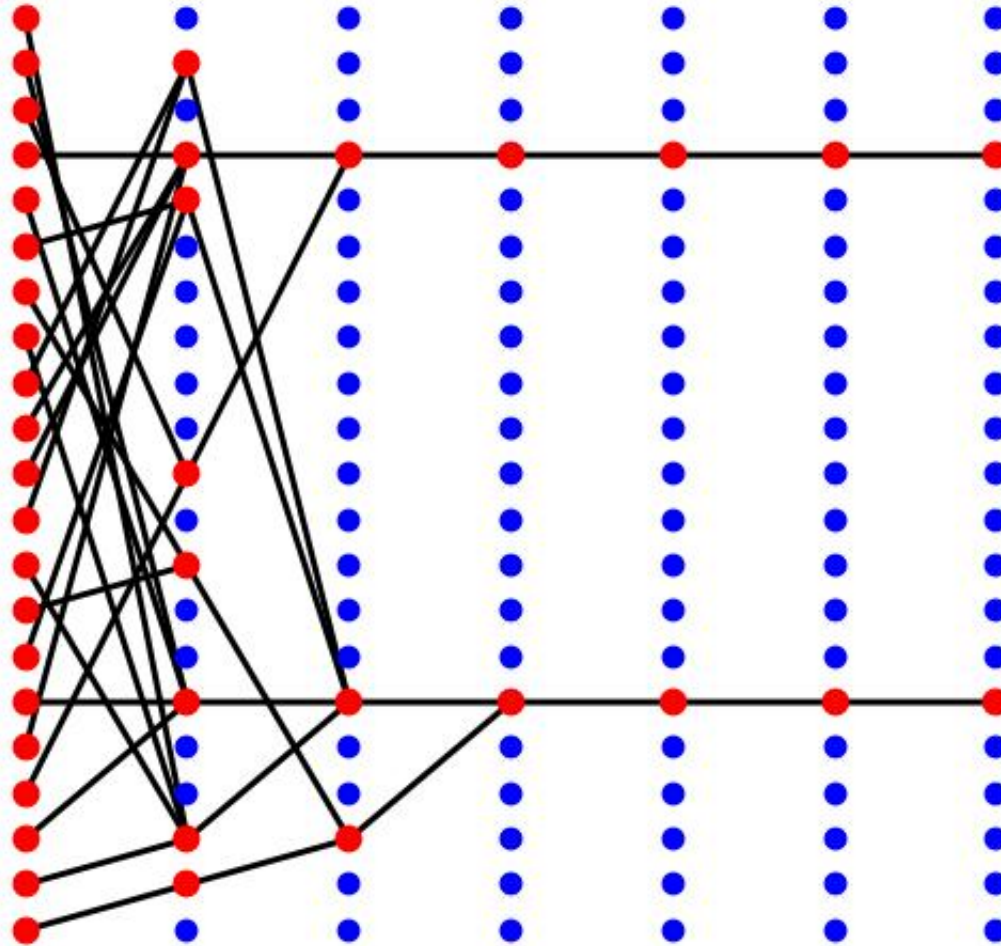
Example 2

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



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

And Iterating



Higher-Order Functions

Complicated Domains

Functions are obviously crucial in programming. In fact, a C program is essentially a (usually vast) collection of functions calling each other.

For a function $f : A \rightarrow B$, it is often convenient to apply f not just to elements of its actual domain A , but also to subsets thereof.

Let $X \subseteq A$ and $Y \subseteq B$. Define

$$f(X) = \{ f(x) \mid x \in X \} \subseteq B,$$
$$f^{-1}(Y) = \{ x \in A \mid f(x) \in Y \} \subseteq A.$$

In the forward direction, this is similar to the map command in Lisp or Mathematica.

Example 5. Let $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x) = x^2$. Then $f([-2, 2]) = f([0, 2]) = [0, 4]$,
 $f^{-1}([0, 1]) = [-1, 1]$, $f^{-1}([-1, 1]) = [-1, 1]$, $f^{-1}([-2, -1]) = \emptyset$

Extending Functions

More formally, we can explain application of f to sets of arguments as lifting the function to the power set.

$$f : A \rightarrow B$$

gives rise to the two functions

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

$$f^{-1} : \mathfrak{P}(B) \rightarrow \mathfrak{P}(A)$$

Note that f^{-1} in this sense is a bonified function, even if f is not injective.

If $Y \cap \text{rg}(f) = \emptyset$, we have $f^{-1}(Y) = \emptyset$, but that's OK.

And, of course, $f^{-1}(\{b\})$ need not be a singleton.

Functions as Input

Another problem in programming is functions that take other functions as input, these are sometimes called *higher-order functions*, *functionals*.

- Integration (or differentiation) of real functions

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

- Composition

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

- Binding

$$\text{bind1st} : (A \times B \rightarrow C) \times A \rightarrow (B \rightarrow C)$$

- Map

$$\text{map} : (A \rightarrow B) \times \text{List}(A) \rightarrow \text{List}(B)$$

- Iteration

$$\text{iterate} : (A \rightarrow A) \times \mathbb{N} \rightarrow (A \rightarrow A)$$

Example: Fold

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

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

By recursion, define

$$\begin{aligned} \text{fold}(f, e, \cdot) &: \text{List}(B) \rightarrow A \\ \text{fold}(f, e, L) &= \begin{cases} e & \text{if } L = \text{nil}, \\ f(\text{fold}(f, e, K), b) & \text{if } L = (K, b). \end{cases} \end{aligned}$$

For example,

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

But Why?

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

$f(a, b)$	e	fold($f, e, .$)
$a + 1$	0	length
pre(a, b)	nil	reverse
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 $e = \text{nil}$ and

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

Exercise 5. *Explain in detail how this implementation of removal works.*

Exercise 6. *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.