

CDM

Iteration and Computation

Klaus Sutner

Carnegie Mellon University

Fall 2009

Outline

- 1 Finding Cycles
- 2 Fast Iteration
- 3 Iteration and Decidability
- 4 Theory of Fixed Points

Calculating Transients and Periods

How do we compute the transient t and period p of the orbit of $a \in A$ under $f : A \rightarrow A$ for finite carrier sets A ?

The obvious brute force approach is to use a container to keep track of everything we have already seen:

$$a, f(a), f^2(a), \dots, f^i(a)$$

and then to compare $f^{i+1}(a)$ to all these previous values.

In most cases, the data structure of choice is a hash table or tree: we can check whether $f^{i+1}(a)$ is already present in expected constant time or logarithmic time, respectively. Memory requirement is linear in the size of the orbit assuming the elements in A require constant space (a fairly safe assumption, if the elements are big use pointers).

Floyd's Trick

A classical problem from the early days of Lisp: Suppose we have a pointer-based linked list structure in memory and we want to check if there are any cycles in the structure (as opposed to having all lists end in `nil`).

We can think of this as an orbit problem:

- A is the set of all nodes of the structure,
- $f(x) = y$ means there is a pointer from x to y .

The Problem:

Suppose further the structure consumes 90% of memory, so we cannot afford to build a large hash table or tree.

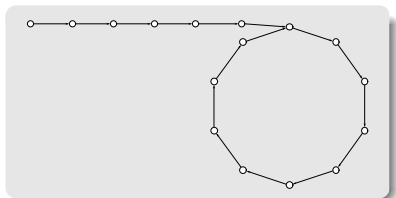
Can we compute transients and periods on $O(1)$ space?

A Memoryless Approach

At first glance, this may seem quite impossible: if we forget already discovered elements we *obviously* cannot detect cycles.

Nonetheless, the following code finds a point on the cycle in the orbit of a .

```
u = f(a);  
v = f(u);  
while( u != v )  
{  
    u = f(u);  
    v = f(f(v));  
}
```



Claim

Upon termination, $u = v$ is a position on the cycle.

Moving Pebbles

Think of two pebbles u and v , moving at speed 1 and 2, respectively.

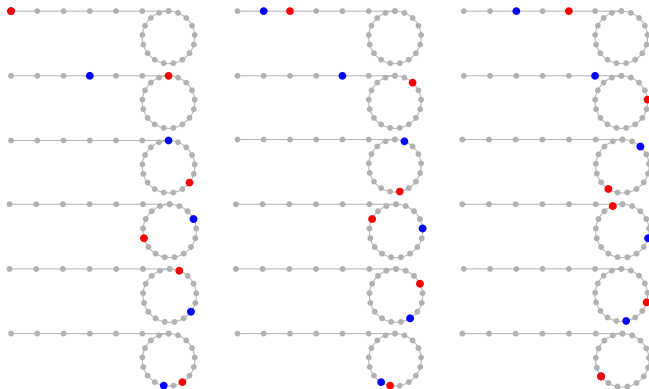
The slow pebble u enters the limit cycle at time t , the transient, when the fast pebble v is already there. From now on, v gains one place on u at each step. So pebble v must catch up at time s where $s \leq t + p$, where p is the period.

Also note that once we have our foothold on the cycle, we can compute the period: run around the cycle one more time, counting.

One can make a nice movie out of this. OK, it is pretty boring after all, but what do you expect.

example

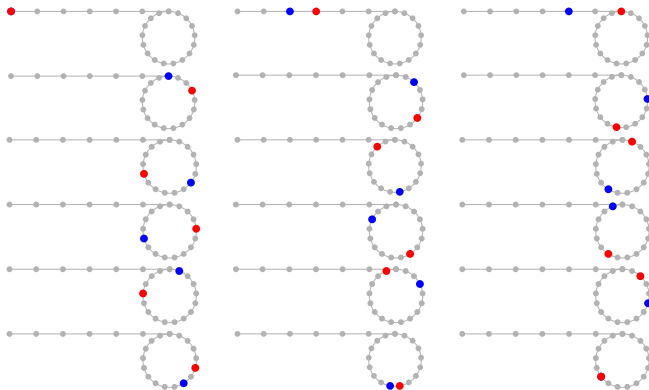
Here the transient is 6, and the period 17.



The pebbles meet at time 17.

Example

Same transient and period, but this time the pebble speeds are 2 and 3, respectively.



Again, the pebbles meet at time 17.

How about the Transient?

Suppose we already know p , the period.

```
t = 0;
u = a;
v = iterate( f, a, p ); // v = f^p(a)
while( u != v ) {
    u = f(u);
    v = f(v);
    t++;
}
```

Claim

Upon completion, t is the transient length.

Proof. v is exactly p places ahead of u . So, when u first enters the cycle, v has just gone around once, and they meet. □

Floyd's Trick

Let us assume f to be computable in time $O(1)$ and elements of the carrier set A to take space $O(1)$.

Theorem

One can determine the transient t and period p of a point in A under f in time $O(t + p)$, and space $O(1)$.

Linear time cannot be avoided in general (why?), so this is optimal.

Exercise

The choice of speeds 1 and 2 for the pebbles in Floyd's algorithm is natural, but there are other possibilities. Discuss other choices.

Beware of Permutations

Floyd's cycle finding algorithm is an excellent general purpose tool in particular when the evaluation of the function in question is cheap.

But note that in the special case where the function is known to be a permutation on a finite domain there is, of course, no need to use Floyd's or similar cycle finding algorithms: since the components of the diagram are all cycles we can simply trace a cycle once to determine its length. So the natural method to compute cycle length is automatically memoryless (if we assume the objects in question can be stored in constant space).

Incidentally, determining cycle lengths of permutations is very important for some advanced counting methods, more later.

The Real Challenge

Typically we are not interested in just one map $f : A \rightarrow A$ but in a whole family of maps. For example, we may want

- $A = \{0, 1, \dots, n - 1\}$
- $A = \{1, 2, \dots, n\}$
- $A = 2^n$
- $A = \Sigma^n$
- $A =$ full binary trees on n leaves

We need a description that holds for all values of n , preferably in closed form.

Here are some simple examples.

Modular Addition

Consider the additive function

$$\begin{aligned}f_k &: \mathbb{Z}_n \rightarrow \mathbb{Z}_n \\x &\mapsto x + k \pmod n\end{aligned}$$

f_k is clearly injective, so the orbits are all cycles.

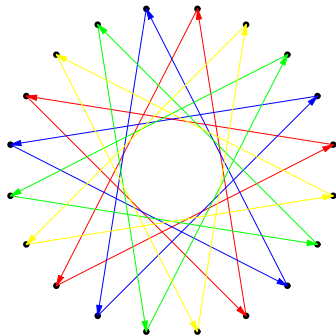
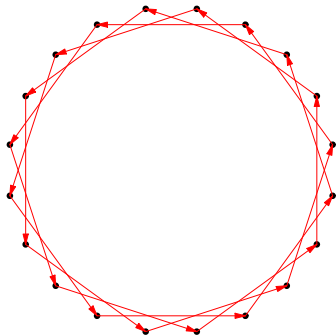
Moreover, since $f_k(x + d) = f_k(x) + d \pmod n$ all the cycles must have the same length.

So how many orbits are there?

Proposition

f_k has $\gcd(n, k)$ distinct orbits, each of length $n / \gcd(n, k)$.

$$n = 20$$



The stride is $k = 3$ on the left, $k = 8$ on the right.

How About Multiplication?

Can we come up with a similar analysis for the multiplicative function

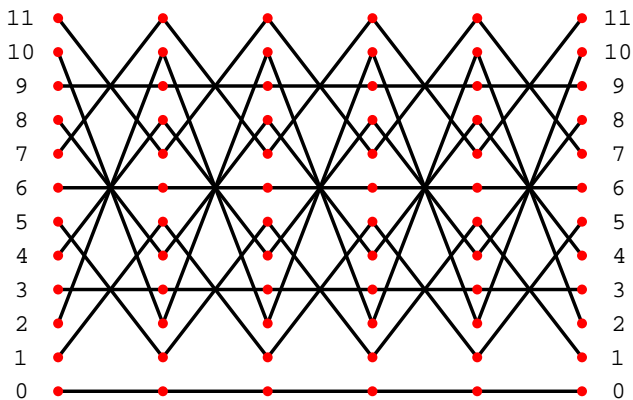
$$g_k(x) = k \cdot x \bmod n$$

One should expect somewhat greater difficulties here: g_k is not injective in general, so the orbits will have transients.

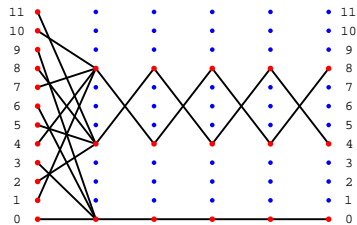
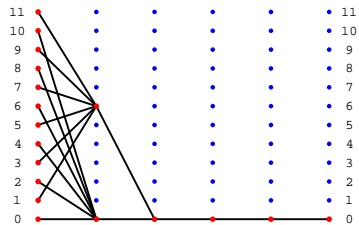
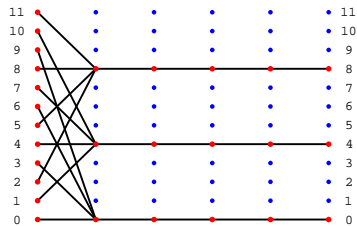
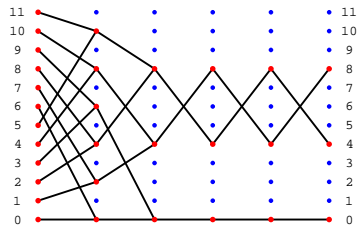
Moreover, the orbits cannot simply be translated into each other, not even the periodic parts.

A complete description of the digraph of g_k will be much more complicated than in the additive case.

$$n = 12, k = 5$$



There are 4 fixed points $(0, 3, 6, 9)$ and 4 2-cycles $((1, 5), (2, 10), (4, 8), (7, 11))$.

$n = 12, k = 2, 4, 6, 8$


The Easy Case

Suppose n and k are coprime. In this case g_k is injective, all orbits are periodic, all transients are 0.

That means that for all x there is some $p > 0$:

$$x = k^p \cdot x \pmod{n}$$

For this to hold it suffices that $k^p = 1 \pmod{n}$, so the multiplicative order of k in Z_n^* is an upper bound for the period p . But this bound is not tight, the choice of x also plays a role.

Exercise

Figure out the details.

- Finding Cycles

2 Fast Iteration

- Iteration and Decidability
- Theory of Fixed Points

Recall: Fast Exponentiation

Recall that the laws of iteration look very similar to exponentiation:

$$\textcircled{1} f^n \circ f^m = f^{n+m}$$

$$\textcircled{2} (f^n)^m = f^{n \cdot m}$$

Also recall the standard method for fast exponentiation based on squaring:

$$a^{2e} = (a^e)^2$$
$$a^{2e+1} = (a^e)^2 \cdot a$$

which allows us to compute a^e in $O(\log e)$ multiplications.

Is there an analogous way to perform “fast iteration?”

Fast Iteration

Suppose we want to compute f^{1000} . The obvious way requires 999 compositions of f with itself.

By copying the standard divide-and-conquer approach for fast exponentiation we could try

$$\begin{aligned}f^{2n} &= (f^n)^2 \\ f^{2n+1} &= f \circ (f^n)^2\end{aligned}$$

This seems to suggest that we can compute $f^n(x)$ in $O(\log n)$ applications of f .

After all, it's just like exponentiation, right?

Linear Maps

If the function f in question is linear it can be written as

$$f(x) = M \cdot x$$

where M is a square matrix over some suitable algebraic structure. Then

$$f^t(x) = M^t \cdot x$$

and M^t can be computed in $O(\log t)$ matrix multiplications.

So this is an exponential speed-up over the standard method.

Polynomial Maps

Another important case is when f is a polynomial map

$$f(x) = \sum a_i x^i$$

given by a coefficient vector $\mathbf{a} = (a_d, \dots, a_1, a_0)$.

In this case the coefficient vector for $f \circ f$ can be computed explicitly by substitution. This is useful in particular when computation takes place in a quotient ring such as $R[x]/(x^n - 1)$ so that the expressions cannot blow up.

We will encounter this again later when discussing cellular automata.

But Beware of Hasty Conclusions

But we cannot conclude that $f^t(x)$ can always be computed in $O(\log t)$ operations.

The reason fast exponentiation and the examples above work is that we can explicitly compute the powers of the number and the powers of the function we are dealing with.

But in general, there are no shortcuts to evaluating $f \circ f$: we have to evaluate f twice.

Just think of f as being given by a piece of C code. We can produce another piece of C code that computes f^2 , and more generally for f^t , but the code just evaluates f t -times, in the obvious brute-force way.

This is very different from performing a squaring operation on, say, an integer: we obtain an integer, given an integer as input.

Weird Mod Sequence

Speaking about hasty conclusions, here is a simple inductively defined sequence of integers.

$$a_1 = 1$$

$$a_n = a_{n-1} + (a_{n-1} \bmod 2n)$$

Thus, the sequence starts like so:

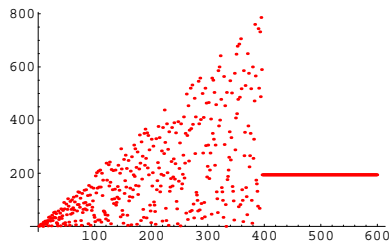
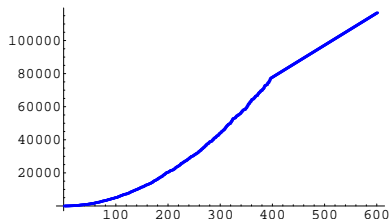
1, 2, 4, 8, 16, 20, 26, 36, 36, 52, 60, 72, 92, 100, 110, 124, 146, 148, 182, 204

This seems rather complicated. The function appears to be increasing but in a complicate manner.

Alas, there is a rude surprise.

Ultimately Linear

The sequence is ultimately linear: $a_{396+k} = a_{396} + k \cdot 194$ for $k \geq 0$.



The plot on the left is the sequence, on the right (in red) are the forward differences.

Exercise

Figure out why the sequence is ultimately linear.

- Finding Cycles
- Fast Iteration
- ③ Iteration and Decidability
 - Theory of Fixed Points

Reachability and Confluence

How about decidability issues related to iteration?

We have already seen two decision problems

Problem: **Reachability Problem**
Instance: A function $f : A \rightarrow A$, two points $a, b \in A$.
Question: Is b in the orbit of a under f ?

Problem: **Confluence Problem**
Instance: A function $f : A \rightarrow A$, two points $a, b \in A$.
Question: Do the orbits of a and b under f overlap?

Actually, there are two versions: f may be fixed and only a and b change, or f may be part of the input.

Undecidability

When the carrier set A is finite the questions are trivially decidable and we even have clever algorithms to solve these problems based on Floyd's trick.

- In the infinite case the most interesting situation is when f is computable and when $A = \mathbb{N}$ or $A = \Sigma^*$.
- Note that both problems are always semi-decidable in this case.
- However, in general both Reachability and Confluence are undecidable, even for fixed f .

Note: With some effort one can rig things up so that Reachability is undecidable but confluence is decidable, and the other way round. So the two problems are clearly related (e.g. Reachability implies Confluence) but there is no simple computational link.

Word Processing

If we use the standard coding of configurations of a Turing machine as words in

$$\Sigma^* Q \Sigma^*$$

then the one-step operation is very simple, it's really just word processing: we scan the word, copying until we hit $p \in Q$, then perform a small edit operation near p , and afterwards copy the rest of the word.

It is easy to see that this operation can be implemented in linear time and constant space.

Exercise

Give a careful description of the one-step operation as a string editing operation.

Why Belabor This?

In essence, we are just dealing with the Halting Problem, for the umpteenth time.

But note that undecidability often is reflected in various easier versions of the problem still being difficult.

For example, consider a directed graph G on n points. There is a natural Reachability Problem for G : is there a path from a to b ?

Of course, the problem is easily solvable: we can use, say, depth-first-search to check path existence in linear time.

But all standard algorithms require linear space, not just linear time. Logarithmic space suffices when we allow nondeterminism (more later) but in the deterministic case it seems to be too little.

Back To Collatz

The Collatz Conjecture is very easy to state:

Conjecture (Collatz)

All orbits of the Collatz function end in 1.

Unfortunately, though this problem is not included in the Clay challenge, some believe it to be enormously difficult.

Mathematics is not ready for this kind of problem.

Paul Erdős

This is not an instance of sour grapes, Erdős was one of the shining lights of 20th century discrete mathematics (Erdős number).

Can We Use Computability?

Here is a desperate idea: Perhaps we could use our knowledge of computation to shed some light on this problem?

After all, the Collatz function is easily computable and we could build a small register machine that computes $C(x)$, or the stopping time function $\sigma(x)$.

Maybe we can use decidability or undecidability to tackle the problem?

Stopping Time is Computable

Recall the stopping time function $\sigma : \mathbb{N} \rightarrow_p \mathbb{N}$:

$$\sigma(x) = \begin{cases} \min(t \mid C^t(x) = 1) & \text{if } t \text{ exists,} \\ \uparrow & \text{otherwise.} \end{cases}$$

σ is computable: just wrap a while-loop around the register machine computing C and add a counter.

Then the Collatz Conjecture is equivalent to σ being a total function.

Alas, it is undecidable whether a computable function is total; in fact this problem is worse than the Halting Problem, we would need access to \emptyset'' .

Collatz Decision Problem

So how do we turn Collatz into a decision problem?

Here is one fairly natural approach, really a version of Reachability.

Problem:	Collatz Orbit Problem
Instance:	A positive natural number x .
Question:	Does the orbit of x under C contain 1?

Observations:

- This problem is clearly semi-decidable.
- If the Collatz Conjecture is true, this problem is trivially decidable.
- Unfortunately, if this problem is decidable, the Collatz Conjecture may still be wrong (though the counterexamples are not terribly complicated: the set of all counterexamples is decidable).

Not too promising.

Prof. Dr. Alois Wurzelbrunft's Brilliant Idea

One might think that the right question to ask is this:

Problem:	Wurzelbrunft's Collatz Problem
Instance:	The Collatz function (or, if you prefer: a banana).
Question:	Is the Collatz Conjecture true?

Wurzelbrunft's Collatz problem is trivially decidable, albeit for entirely the wrong reasons.

The issue here is that there is only one instance.

Say What?

For decidability, all we need is an algorithm that solves Fred's Collatz Problem.

No problem, here are two candidates:

- Algorithm 1: Ignore the input and output Yes.
- Algorithm 2: Ignore the input and output No.

One of those two algorithms solves this decision problem, we just don't know which.

Note that no one said that we need to be able to write down the algorithm explicitly, we just have to make sure an algorithm exists. But one of the two candidates is guaranteed to work.

Theology

This type of argument caused a huge uproar in the mathematics community when first used by D. Hilbert in 1890 (finite basis theorem).

"This is not mathematics, this is theology."

Paul Gordon

Gordon was upset since he had found a constructive, computational proof for special case $n = 2$ in 1886, but had failed in all attempts to generalize the argument to arbitrary n .

Hilbert handled the general case not by explicitly computing a solution, but by showing that the assumption that there is no solution leads to a contradiction.

Later On

Felix Klein (of the eponymous bottle) strongly supported Hilbert's work. A while later, following suggestions of Klein and Gordon, Hilbert wrote a second paper showing how his approach can actually yield some bounds on the degree of the polynomials in question.

Gordon grudgingly concluded:

Theology may have its uses.

Finite Problems

The same trick works for any decision problem with only finitely many instances x_1, x_2, \dots, x_n :

This time there are 2^n algorithms that are potential solutions:

$$A_0, A_1, \dots, A_{2^n-2}, A_{2^n-1}$$

We just may not know which of these algorithms is the right one.

But: The algorithm does exist, so the problem is decidable.

In other words, classical computability theory is ill-equipped to deal with finite decision problems: the definition just does not bite.

So all this Computability Stuff is Useless?

In a certain sense, yes.

Problems with a single instance like the Collatz Conjecture, Riemann Hypothesis, $\mathbb{P} = \mathbb{NP}$ problem, and so, don't naturally fit into this framework.

Also, in practical computations one only deals with instances of limited size and there are only finitely many of those. For bounded size input an algorithm always exists.

But, we have no clue which one it is. Moreover, undecidability and unsolvability cast a noticeable shadow in the realm of "small instances". For example, Diophantine equations are difficult even when restricted to apparently simple special cases. Testing a C program for termination is hard work.

And Collatz?

For the Collatz problem John Horton Conway, of Game-of-Life fame, found a beautiful way to show how undecidability lurks nearby.

Conway's idea: How about constructing infinitely many Collatz Conjectures?

More precisely, come up with a family of functions that generalize the Collatz function slightly. Then ask if for one of these functions all orbits are ultimately periodic.

Problem:	Conway-Collatz Problem
Instance:	A Conway-Collatz function f .
Question:	Are all orbits of f ultimately periodic?

Conway's theorem

Define a family of Collatz-like functions

$$C_n(\mathbf{a}, \mathbf{b})(n) = a_i \cdot m + b_i$$

where $k = |\mathbf{a}| = |\mathbf{b}|$, $i = n \bmod k$, $m = n \operatorname{div} k$.

Classical Collatz is essentially the special case

$k = 2$, $a_0 = 1$, $a_1 = 6$, $b_0 = 0$, $b_1 = 4$,

so nothing is lost by considering Conway's functions.

But now we have infinitely many functions to deal with (though one of them is perhaps more interesting than all the others).

Conway-Collatz Problem

Problem: **Conway-Collatz Problem**
Instance: The parameters \mathbf{a} , \mathbf{b} .
Question: Is every orbit of $C_n(\mathbf{a}, \mathbf{b})$ ultimately periodic?

Theorem

The Conway-Collatz Problem is undecidable.

It remains undecidable even if all the b_i 's are 0.

The theorem indicates that there is no good general way to answer questions about Collatz-like functions. So it is not surprising that the classical Collatz function is also very difficult to analyze.

Conway's T Function

A famous example of a Conway function other than the classical Collatz function is the following:

$$\begin{aligned}T(2n) &= 3n \\T(4n + 1) &= 3n + 1 \\T(4n - 1) &= 3n - 1\end{aligned}$$

This is just $C_n(6, 3, 6, 3; 0, 3, 1, 2)$.

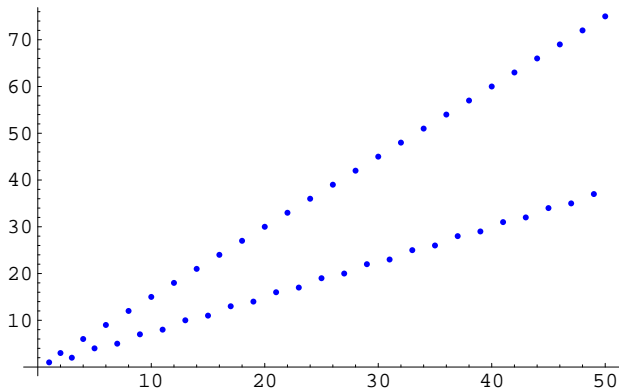
Proposition

$T : \mathbb{N} \rightarrow \mathbb{N}$ is a bijection.

Exercise

Prove that the T function is a bijection. Then look for cycles under T .

Plot



Looks very similar to the Collatz function.

But note that the lower line wobbles; there are really 3 linear functions here.

Conway's T Function

Known finite cycles are:

(1),

(2, 3),

(4, 6, 9, 7, 5),

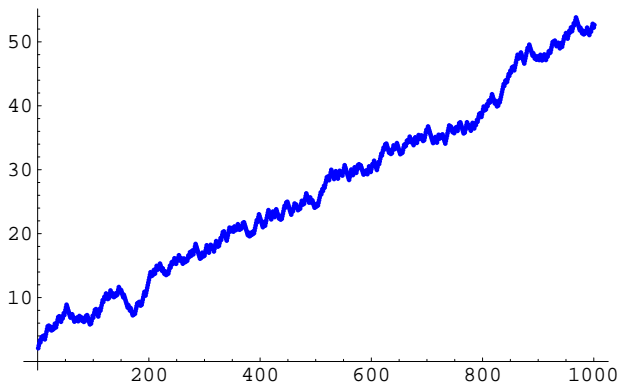
(44, 66, 99, 74, 111, 83, 62, 93, 70, 105, 79, 59).

Open Problem:

Are there any other finite orbits?

In particular, is the orbit of 8 finite?

Orbit of 8



This is a log-plot. It seems to suggest the orbit of 8 grows without bound, but of course this is neither here nor there: the first 1000 values are really meaningless.

- Finding Cycles
- Fast Iteration
- Iteration and Decidability
- ④ Theory of Fixed Points

The Continuous Case

In the world of continuous functions there are a few well-known results that guarantee the existence of a fixed point.

Theorem (Brouwer)

Any continuous map from the closed unit sphere in dimension n to itself has at least one fixed point.

Theorem (Banach)

Every contraction map on a complete metric space has exactly one fixed point.

Recall: Lattices

To obtain comparable results in the discrete domain requires a bit of preparatory work. We need to find the right algebraic structure (just as complete metric spaces are the right structure for Banach's theorem).

Definition

A **lattice** is an algebraic structure $\mathcal{A} = \langle A, \sqcap, \sqcup \rangle$ with two binary operations, referred to as **meet** and **join** that are both associative, commutative and idempotent. Moreover, the absorption law holds:

$$x \sqcap (x \sqcup y) = x \sqcup (x \sqcap y) = x$$

Examples

Example

Boolean values true and false with logical connectives “and” and “or” form a lattice.

Example

The powerset of any set with operations intersection and union form a lattice.

Example

Binary relations on a set form a lattice with intersection and union. Likewise, equivalence relations form a lattice, albeit with a different meet operation.

Example

The positive integers with gcd and lcm form a lattice.

The Poset Interpretation

Another way to look at lattices is to consider a poset $\langle A, \leq \rangle$.

If the poset is properly behaved, we can define binary operations

$$\begin{aligned}\inf(x, y) &= \max\{z \in A \mid z \leq x, y\} \\ \sup(x, y) &= \min\{z \in A \mid x, y \leq z\}\end{aligned}$$

Then $\langle A, \inf, \sup \rangle$ is a lattice.

One the other hand, given a lattice we can define a partial order by

$$x \leq y \iff x \sqcap y = x.$$

This partial order has sups and infs which turn out to be exactly the join and meet operations of the lattice.

Complete Lattices

In a lattice, the join and meet operations are binary by definition. Of course, they can be generalized to a finite number of arguments, but there is requirement that sups and infs should exist for arbitrary subsets of A .

Definition

A lattice is **complete** if every subset of the carrier set has a join and a meet.

Note that every complete lattice must have a least and a largest element.

Example

Every finite lattice is complete.

The infinite examples from above are all complete.

The reals with standard order form an incomplete lattice.

The natural numbers with divisibility form a complete lattice (this is tricky).

Lattices and Fixed Points

In analysis, the existence of fixed points is a difficult topic. In our setting, there is a very powerful theorem that often can be used to demonstrate the existence of fixed points.

Theorem (Knaster-Tarski)

Let L be a complete lattice and $f : L \rightarrow L$ a monotonic map on L . Then the set of fixed points of f is a complete sublattice of L .

It follows from the theorem that fixed points exist; there may even be many of them.

Moreover, there is a least fixed point μf as well as a largest fixed point νf .

We will skip over the proof and quickly look at some applications.

Equivalential Closure

Suppose we have a binary relation ρ on A .

Define the following operation f for any binary relation X on A :

$$f(X) = I_A \cup \rho \cup (X \circ X) \cup X^{-1}$$

The lattice of binary relations on A is clearly complete, and f is monotonic: $X \leq Y$ implies $f(X) \leq f(Y)$.

Then $\mu f = \bigcup_{n < \omega} f^n(\emptyset)$ is the equivalential closure of ρ . The closure ordinal is ω since the chains required by transitivity are all of finite length.

Banach's lemma

Lemma

Let $f : A \rightarrow B$ and $g : B \rightarrow A$ be two arbitrary functions. Then there are partitions $A = A_1 \cup A_2$ and $B = B_1 \cup B_2$ such that $f(A_1) = B_1$ and $g(B_2) = A_2$.

Proof. We exploit the Knaster-Tarski theorem. Consider the powerset of A , clearly a complete lattice.

For any $X \subseteq A$ define

$$F(X) = A - g(B - f(X))$$

A moment's thought reveals that F is indeed monotonic and thus has a least fixed point.

But then $A_1 = \mu F$ works as required (make sure to verify this fact). □

Cantor-Bernstein

Note that the Cantor-Bernstein theorem is a simple corollary to Banach's lemma: when f and g are both injective we have $|A_1| = |B_1|$ and $|A_2| = |B_2|$.

This argument is considerably less complicated than the standard proof of Cantor-Bernstein.

Exercise

Try to prove Banach's lemma from scratch, without reference to the Knaster-Tarski theorem. Try some special cases first, say, f maps to one point, is surjective, and so on.

Exercise

Look up some standard proofs for Cantor-Bernstein. How do they compare to the proof above.

Summary

- Iteration produces complicated behavior even in simple functions.
- It matters little whether the domain is discrete or continuous.
- Many algorithms can be construed as fixed point constructions.
- Some computational environments such as Mathematica offer a fixed point operation as a primitive.
- There is a memoryless linear time algorithm to compute transient and period of a map on a finite carrier set.
- Questions about the orbits of functions may easily be undecidable, even if the functions in question are very simple.
- There are results in the discrete realm that are similar to classical fixed point theorems in analysis, but they require a bit of machinery.