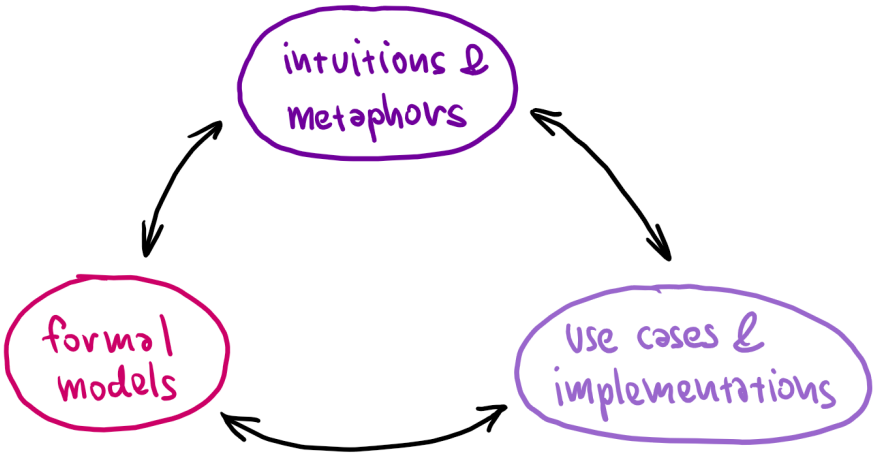
Constructivity

What does it take to show that something exists?

David Corfield

NPRG086: Functions, trees, processes

Course schema



Where we left off

- By 1900 a function could be an **arbitrary correspondence**: a table, possibly with no rule behind it.
- Counting teaser: programs can be listed; functions $\mathbb{N} \to \mathbb{N}$ cannot. So most "functions" have no program.
- Brouwer was flagged as someone who would say: every function on $[0, 1]$ is continuous.

Today: what happens if we insist that mathematical objects come *with a construction*? And what exactly is a construction?

Running question: progress, or the loss of an intuition?

A puzzle

Claim. There are irrational numbers a, b with a^b rational.

Proof. Look at $\sqrt{2}^{\sqrt{2}}$. Either it is rational or it isn't.

- If rational: take $a = b = \sqrt{2}$.
- If irrational: take $a = \sqrt{2}^{\sqrt{2}}$, $b = \sqrt{2}$. Then $a^b = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^2 = 2$. □

Which case is it?

The proof doesn't say. It shows that a pair *exists* without telling you *which* pair.

Constructivism does not object to the objects whose existence classical mathematics proves. It asks what information an existence proof actually provides.

(In fact $\sqrt{2}^{\sqrt{2}}$ is irrational: Gelfond–Schneider, 1934. A direct witness: $a = \sqrt{2}$, $b = \log_2 9$, giving $a^b = 3$.)

The same proof as a program

```
def irrational_pair():
    c = sqrt2 ** sqrt2
    if is_rational(c): # <- who implements this?
        return (sqrt2, sqrt2)
    else:
        return (c, sqrt2)
```

- The classical proof uses a test nobody can run: **"P or not P"** for an arbitrary statement P.
- This is the **law of excluded middle** (LEM).
- A programmer's instinct: an existence claim is a *promise to deliver* the thing. Today's story is about mathematicians who shared that instinct, decades before computers.

Road map

When	Who / what	Theme
1888–1905	Hilbert, Zermelo, the "five letters"	Existence without construction
1907–1928	**Brouwer**	Mathematics as mental construction
1930–1933	**Heyting**, Kolmogorov, Gödel	What logical words mean
1931–1937	**Gödel, Church, Turing**	Pinning down "mechanical"
1950s	**Markov**, Specker, Kleene	Construction = algorithm
1967	**Bishop**	Constructive analysis for everyone
1970s–	Martin-Löf, proof assistants	Proofs as programs

Computability comes before Markov and Bishop, because they build on it.

I. Existence without construction

c. 1880–1910

Early tremors

- **Kronecker** (1880s): mathematical objects should be built from the whole numbers in finitely many steps. Suspicious of completed infinities. (The line "God made the integers, all else is the work of man" is attributed to him, reported after his death.)
- **Hilbert** (1888): every system of invariants has a finite basis. Proved by showing that the alternative leads to contradiction; no basis is produced.
- Gordan, the expert in the field, is said to have replied: *"This is not mathematics, it is theology."* (C. McLarty, *Hilbert on Theology and Its Discontents*.)
- Hilbert later gave a more constructive proof (1893).

1904–05: the axiom of choice

- **Zermelo** (1904): every set can be well-ordered. Key step: *choose* one element from each of infinitely many sets at once, with no rule for the choosing.
- 1905: an exchange of **five letters** between Borel, Baire, Lebesgue and Hadamard.
  - Hadamard: the choice exists; we needn't be able to describe it.
  - Borel, Baire, Lebesgue: an object we cannot specify is not really an object. To define something is to say what singles it out.
- Notice the echo of last lecture: a choice function is a **pure table** with no rule behind it.

Add to this Russell's paradox (1901), and by 1910 intuitions about foundations looked shaky.

II. Brouwer's intuitionism

1907–1930

L. E. J. Brouwer (1881–1966)

- Dutch; also a major topologist (fixed point theorem, invariance of dimension).
- 1905 Manifesto: *Life, Art and Mysticism*
- 1907 thesis *On the Foundations of Mathematics*; 1908 *The Unreliability of the Logical Principles*.
- Core view: mathematics is a **language-free activity of the mind**, constructions carried out in time. Words and formulas only record it, imperfectly.
- Logic was abstracted from reasoning about *finite* collections. Applying it to the infinite is an unjustified extrapolation.

And yet his own fixed point theorem was proved non-constructively.

Excluded middle: finite vs. infinite

- **Finite:** "Some number below 1000 is a counterexample, or none is." We can check every case. LEM is harmless here.
- **Infinite:** "Some even number > 2 is not a sum of two primes, or none is." (Goldbach.)
  - To assert this, Brouwer wants *either* a counterexample *or* a proof that none exists.
  - We have neither.
- Brouwer does not say LEM is *false*. He says we have no right to assert it *in general*.

The disjunction "P or not P" is a claim to **know which**.

A real number no one can place

Define a number $x = 0.d_1 d_2 d_3 \ldots$ digit by digit:

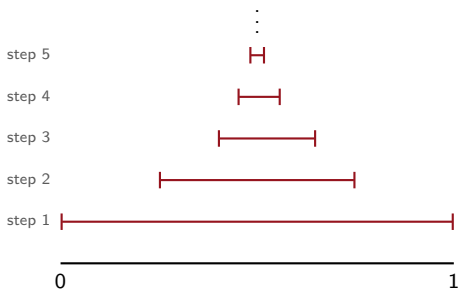
$$d_n = \begin{cases} 1 & \text{if the block 0123456789 first ends at digit } n \text{ of } \pi, \\ 0 & \text{otherwise.} \end{cases}$$

- Every digit is computable: just expand π far enough.
- Is $x = 0$ or $x \neq 0$? That depends on whether the block ever appears.
- So "every real is zero or not zero" is **not something we can assert**.
- Nor "$x = 0$ or $x \# 0$" (x is apart from 0). Nor "$x < y$ or $x = y$ or $x > y$".

The block was found in 1997, about 17 billion digits in. Brouwer's example expired! Swap in Goldbach and it works again. These are **weak counterexamples**: relative to what we currently know.

Choice sequences: the continuum as unfinished

- A real number is not a finished point. It is given by a **process** producing ever-smaller intervals.
- The process may follow a law, or be **freely chosen** step by step.
- At any moment we know only a *finite* part of it.
- Computing analogy: a stream from a sensor, or a user typing. You cannot peek ahead.



Why every function must be continuous



- If F gives an answer for *every* input stream, it must commit after reading *finitely many* digits.
- Then every input sharing those digits gets the same answer. That is **continuity**.
- Step function: $f(x) = 0$ if $x < \frac{1}{2}$, $f(x) = 1$ if $x \geq \frac{1}{2}$. Feed in $0.4999\ldots$: every finite prefix is compatible with both sides. F can never commit.

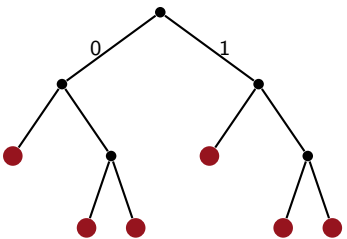
Brouwer's continuity theorem (1920s)

Every function $[0,1] \to \mathbb{R}$ is (uniformly) continuous.

- Classically **false**. Step functions, Dirichlet's function and Thomae's function are counterexamples.
- Intuitionistically those monsters are not refuted. They are **never defined**: there is no construction that would give them a value at every point.
- Compare last lecture: Euler's functions graphed as connected curves; Dirichlet opened the door to arbitrary tables.

Discussion: is this a loss of generality, or the recovery of an older intuition that the 19th century talked itself out of?

The fan theorem



Every path passes a red node, all by depth 3.

Classically this follows from König's lemma. Brouwer justified it by analysing what a proof that every path is barred could look like.

- An infinite path of left/right choices is a point (think binary expansions in $[0, 1]$).
- A **bar**: a set of nodes that every infinite path eventually hits. Each node is a finite string of 0s and 1s; the set itself may be *infinite*.
- **Fan theorem**: then there is a single depth N by which every path has hit the bar. (Trivial if the bar is finite.)
- Programming reading: a program that halts on *every* infinite bit stream has a **uniform bound** on how many bits it ever reads.

The foundations quarrel (*Grundlagenstreit*)

- Weyl (1921): "Brouwer, that is the revolution!" (He later drifted back.)
- Hilbert (1927), taking excluded middle from the mathematician is like taking the telescope from the astronomer, or fists from the boxer.
- **Hilbert's programme**: keep all of classical mathematics, but prove it consistent using only finite, unquestionably safe methods.
- 1928: Hilbert has Brouwer removed from the editorial board of *Mathematische Annalen*.
- Also his treatment of Finsler and non-wellfounded sets, cf. my **article**.

Three strategies: Formalism, Intuitionism, Logicism

Secure classical mathematics by formalizing it (Hilbert),
or **rebuild** it from constructions (Brouwer),
or **reconstruct** it as logic (Frege, Russell).

III. What logical words mean

Heyting, Kolmogorov, Gödel: 1930–1933

Heyting and Kolmogorov

- **Arend Heyting** (1930), Brouwer's student, wrote down formal rules for intuitionistic logic and arithmetic. Brouwer distrusted formalization.
- The rules for logic are classical logic *minus* excluded middle/double negation elimination.
- **Kolmogorov** (1932): read a proposition as a *problem*; a proof is a *solution*.
- Together: the **BHK interpretation** (Brouwer–Heyting–Kolmogorov). The meaning of a statement is given by what counts as a proof of it.

The BHK interpretation

To prove...	**you must give...**	**in programming terms**
$A \wedge B$	a proof of A and a proof of B	a pair
$A \vee B$	a proof of one, **and say which**	a tagged union
$A \to B$	a method turning proofs of A into proofs of B	a function
$\neg A$	a method turning a proof of A into absurdity	a function into the empty type
$\exists x\, P(x)$	a **witness** a and a proof of $P(a)$	a value plus a certificate
$\forall x\, P(x)$	a method giving, for each a, a proof of $P(a)$	a function

Note the "and say which" and the "witness": these are exactly what the $\sqrt{2}^{\sqrt{2}}$ proof failed to supply.

Why excluded middle fails under BHK

- A proof of $P \vee \neg P$ for *every* P would be a universal method for deciding every question. No reason to expect one. (Turing will soon show there isn't one even for simple questions about programs.)
- Double negation: $\neg\neg P \to P$ fails too. Showing that "P is refutable" is absurd does not hand you a proof of P.
- But $\neg\neg(P \vee \neg P)$ *is* provable:
  - suppose $\neg(P \vee \neg P)$;
  - then P would give $P \vee \neg P$, so $\neg P$;
  - but $\neg P$ also gives $P \vee \neg P$. Contradiction.
- The $\sqrt{2}^{\sqrt{2}}$ argument proves $\neg\neg\exists$, not $\exists$, rules out the impossibility of finding a witness.

A preview: proofs as programs

```
const = lambda a: lambda b: a # proves A -> (B -> A)
swap = lambda p: (p[1], p[0]) # proves A and B -> B and A
fst = lambda p: p[0] # proves A and B -> A
dneg = lambda a: lambda k: k(a) # proves A -> not not A, application
# not not A -> A : no program does this in general
```

- Each intuitionistic proof *is* a program, and the statement is its type.
- Observed by Curry (1930s) and Howard (1969): the **Curry–Howard correspondence**.

Intuitionism doesn't lose classical mathematics

- **Gödel and Gentzen** (1933): showed how to translate classical arithmetic into intuitionistic arithmetic. A classical proof of A yields an intuitionistic proof of its negative translation A^N, which is classically equivalent to A.
- So intuitionistic arithmetic is consistent exactly if classical arithmetic is.
- Better picture: intuitionistic logic makes **extra distinctions**.

constructive: with information	**classical shadow**
$\exists x\, P(x)$ (here is one)	$\neg\forall x\, \neg P(x)$ (not none)
$A \vee B$ (this one)	$\neg(\neg A \wedge \neg B)$ (not neither)

Classical logic is the logic that forgets *which*.

IV. Pinning down “mechanical”

Gödel, Church, Turing: 1931–1937

Everyone appealed to "a construction"

- Kronecker, Lebesgue, Brouwer, Hilbert's "finitary" methods: all relied on an *informal* idea of a definite, step-by-step procedure.
- **Entscheidungsproblem** (Hilbert and Ackermann, 1928): is there a mechanical procedure that decides whether any given logical statement is valid?
- To show *yes*, exhibit a procedure. To show *no*, you need to know what counts as a procedure at all.
- 1931–36: several precise definitions arrive at once.

Gödel (1931)

- **Incompleteness**: any consistent formal system that can do basic arithmetic, and whose rules can be mechanically checked, has true statements it cannot prove.
- It cannot prove its own consistency either: a severe blow to Hilbert's programme.
- Two tools that matter for computing:
  - **Gödel numbering**: formulas and proofs coded as numbers. Syntax becomes data.
  - **Primitive recursive functions**: a precise class of "obviously computable" functions.

Primitive recursion = `for` loops

Build up from 0 and +1 by recursion:

$$\begin{aligned} \mathrm{add}(x, 0) &= x \\ \mathrm{add}(x, y+1) &= \mathrm{add}(x, y) + 1 \\ \mathrm{mult}(x, 0) &= 0 \\ \mathrm{mult}(x, y+1) &= \mathrm{add}(\mathrm{mult}(x, y), x) \end{aligned}$$

```
def add(x, y):
    r = x
    for _ in range(y):
        r = r + 1
    return r

def mult(x, y):
    r = 0
    for _ in range(y):
        r = add(r, x)
    return r
```

- Loops whose length is fixed *before* they start. Always terminate.
- But not everything: **Ackermann's function** (1928) is clearly computable yet grows faster than any such program.

Unbounded search = `while` loops

```
def mu(p): # least n with p(n) true
    n = 0
    while not p(n):
        n = n + 1
    return n # ...if we ever get here
```

- Add this search operator and you get the **general recursive** functions (Herbrand, Gödel 1934; Kleene 1936).
- Price: a function may **never return**. Partial functions enter mathematics.
- A new twist in our story of "function": not a total table, not even always defined, but a *rule that might run forever*.

Church: the λ-calculus

- Alonzo Church (early 1930s): a calculus in which **everything is a function**. One rule: $(\lambda x.\, M)\, N \rightsquigarrow M$ with N for x.
- Numbers as functions: "2" means *do it twice*.

```
two = lambda f: lambda x: f(f(x))
succ = lambda n: lambda f: lambda x: f(n(f)(x))
succ(two)(lambda k: k + 1)(0) # -> 3
```

- The "function as rule" picture returns, in pure form.
- **Church's thesis** (1936): effectively calculable = λ-definable & Church (1936): no λ-definable procedure decides validity.
- Therefore, the Entscheidungsproblem has no solution.

Self-application: blessing and curse

As computation

- Let $\Theta_f = (\lambda x.\, f(x\,x))\,(\lambda x.\, f(x\,x))$. One step of reduction gives $\Theta_f \rightsquigarrow f(\Theta_f)$.
- So every f has a **fixed point**. That gives recursion without names: general recursion, `while` loops.

```
Z = lambda f: (lambda x: f(lambda v: x(x)(v)))\
            (lambda x: f(lambda v: x(x)(v)))
fact = Z(lambda rec: lambda n:
        1 if n == 0 else n * rec(n - 1))
fact(5) # -> 120
```

(The textbook Y loops forever in Python, which evaluates arguments first; Z delays them.)

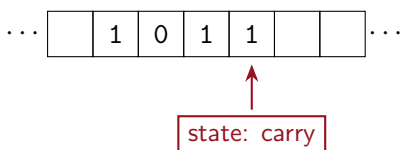
As logic

- Church's first system (early 1930s) also read λ-terms as *formulas*.
- Take $f = \neg$: then $\Theta_\neg = \neg\,\Theta_\neg$, a statement equal to its own negation. Russell's paradox again.
- Kleene and Rosser (1935): the system is **inconsistent**.
- The pure calculus survives as a model of computation (consistent: Church–Rosser, 1936).
- 1940: Church's **simply typed** λ-calculus. No self-application, every program terminates.

Computational power and logical consistency pull in opposite directions. Unrestricted self-application makes general recursion possible; controlling it is essential when programs are also to serve as proofs.

Turing (1936): analysing a human clerk

- "On computable numbers, with an application to the Entscheidungsproblem."
- Turing asks what a person *doing a calculation* can do:
  - finitely many symbols, finitely many states of mind;
  - looks at a few squares of paper at a time;
  - changes one symbol, moves a little, changes state.
- Strip that down and you get a machine:

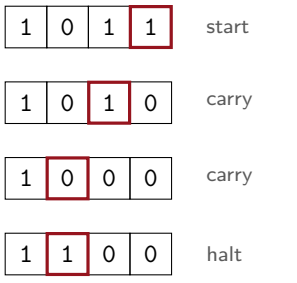


tape (unbounded)
head reads one square

A Turing machine: add 1 in binary

Start at the rightmost digit, in state **carry**.

read	write	move	next
1	0	left	carry
0	1	—	halt
blank	1	—	halt



$1011\ (= 11) \mapsto 1100\ (= 12)$

- A program is just a finite table. Tables can be written down as symbols...

The universal machine

- . . . so put the table of machine M on the tape, followed by the input.
- Turing builds a single machine U that reads the description and **simulates** M.
- One machine, any program: the **program is data**. This is the idea behind the stored-program computer.
- Compare Gödel numbering: formulas as numbers. Here, machines as strings.

The halting problem

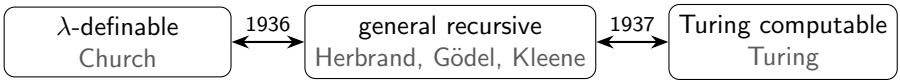
Suppose someone hands you `halts(prog, inp)`, always correct, always returning.

```
def trouble(prog):
    if halts(prog, prog):
        while True: pass # loop forever
    else:
        return "done"

trouble(trouble) # does this halt?
```

- If it halts, then `halts` said yes, so it loops. If it loops, `halts` said no, so it returns. Contradiction.
- This is **Cantor's diagonal argument** again, run on programs instead of sequences.
- So there is no decision method even for this simple question: LEM has *no algorithm behind it*.

Convergence, and a thesis



- Three very different definitions pick out **exactly the same** functions. (Post's 1936 model too.)
- **Church–Turing thesis**: these capture what can be computed by any effective method. A claim linking an informal idea to a formal one, so a thesis, not a theorem.
- Gödel was persuaded by Turing's *analysis of the clerk*, not by the equivalences.
- Back to counting: there are only countably many programs, so the computable functions are a vanishing fraction of Cantor's functions.

Turing meets Brouwer's problem

Turing's 1936 paper is about **computable real numbers**: those whose decimal digits a machine can print.

- Suppose machines produce $x = 0.3333\ldots$ and $y = 0.6666\ldots$, digit by digit. We don't know in advance that the 3s and 6s continue.
- What is the first digit of $x + y$?
  - a later 4 in x: the sum exceeds 1, first digit 1;
  - a later 2 in x: the sum is below 1, first digit 0;
  - neither, ever: $0.999\ldots = 1.000\ldots$
- No finite amount of reading settles it. **Addition is not computable on decimal digits!**

Same problem as the step function: a decimal digit is a step function of the number. Fix: represent reals by approximations with error bounds (Turing's 1937 correction moves in this direction).

V. Markov: construction = algorithm

The Russian school, 1950s

A. A. Markov Jr. (1903–1979)

- Son of the Markov of Markov chains.
- 1950s: his own model of computation, **normal algorithms**: ordered lists of string-rewriting rules. Apply the first rule that fits, at its leftmost match; repeat.

`ba` $\to$ `ab`

`baba` $\longrightarrow$ `abba` $\longrightarrow$ `abab` $\longrightarrow$ `aabb`

no rule applies: stop

- One rule, and you have a sorting program. Equivalent in power to Turing machines.
- Close to how modern text-processing and term-rewriting tools work.

Russian constructivism

- Markov, Shanin and students, from around 1950.
- The only objects are **words** over a finite alphabet, and algorithms (themselves words, such as in $\{\mathrm{a}, \mathrm{b}, \rightarrow\}$).
- A real number: an algorithm producing rationals, plus an algorithm saying how fast they converge.
- A function on reals: an algorithm that acts on such algorithms.
- No free choice sequences. **Construction means algorithm**, in the precise Church–Turing sense.

Contrast

Brouwer: construction is an act of the *mind*, not fully capturable in language.
Markov: construction is a *mechanism*, fully written down.

Three answers to the same question

	A construction is…	**A function is…**
Brouwer	An act of the mind, unfolding in time; choices may be free.	A way of responding to finite information.
Markov	An algorithm, finitely specified and mechanically executable.	An algorithm acting on algorithms.
Bishop	Explicit mathematical evidence, with enough information to compute.	A rule accompanied by appropriate bounds or moduli.

Bishop (1967) will seek a constructive mathematics that need not choose between Brouwer and Markov.

Question: Must a construction be an algorithm, or can it be an open-ended process?

Markov's principle

If $P(n)$ can be checked mechanically for each n, and it is *impossible* that no n satisfies P, then some n satisfies P.

$$\neg\neg\,\exists n\, P(n) \;\rightarrow\; \exists n\, P(n)$$

- Reading: if a search *cannot run forever*, it **will stop**. So just run it.
- Markov: an unbounded but certain search is a perfectly good construction.
- Brouwer and Heyting: no. "Cannot fail" gives no idea of *when*; there is no bound.

A question for programmers: is "this loop can't run forever" as good as "this loop stops"?

Specker's sequence (1949)

- List all programs $p_0, p_1, p_2, \ldots$ and set

$$a_n = \sum \left\{ 2 \cdot 4^{-(k+1)} : k \leq n \text{ and } p_k \text{ halts within } n \text{ steps} \right\}.$$

- Each a_n is a rational we can compute. The sequence increases and stays below 1.
- Classically it converges to a limit L. Its base-4 digits are 0 or 2, according to whether each program halts.
- If we could compute L to arbitrary accuracy, we could recover those digits and **solve the halting problem**. So L is not a computable real.

In Russian constructivism: "every bounded increasing sequence converges" is **false**. A pillar of first-year analysis falls.

Brouwer and Markov disagree

	Brouwer	**Markov**	**Classical**
Every function $[0,1] \to \mathbb{R}$ is continuous	✓	✓	×
Fan theorem (uniform bounds)	✓	×	✓
Markov's principle	$\times^*$	✓	✓
Bounded increasing sequences converge	–	×	✓
Every function $\mathbb{N} \to \mathbb{N}$ is computable	$\times^*$	✓	×

✓ = accepted; – = not derivable; × = refutable.

- * Brouwer's stronger choice-sequence principles (including Kripke's Schema) refute Markov's principle and imply noncomputable functions.
- Continuity in Markov's setting: Tseitin; Kreisel, Lacombe and Shoenfield (late 1950s).
- **Kleene's tree**: an infinite computable binary tree with no computable infinite path; its complement refutes the decidable fan theorem in Markov's setting.
- Same starting intuition, **incompatible theorems**.

VI. Bishop: constructive mathematics for working mathematicians

1967

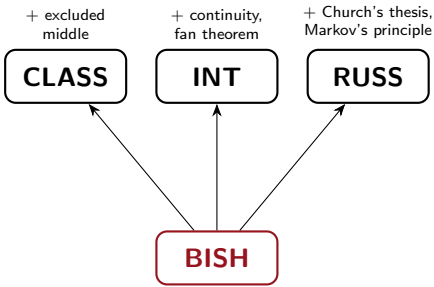
Errett Bishop (1928–1983)

- A successful classical analyst who changed sides.
- *Foundations of Constructive Analysis* (1967): a large body of real analysis, redone constructively, in ordinary mathematical prose.
- His complaint about classical mathematics: too much of it has no **numerical meaning**. Its existence claims don't tell you how to compute anything.
- On Brouwer: right critique, but choice sequences and metaphysics put people off.

See his lecture "Schizophrenia in contemporary mathematics" (1973).

Bishop's minimalism

- Intuitionistic logic, an *informal* notion of algorithm.
- No choice sequences, no Church's thesis, no Markov's principle, no fan theorem.
- So every Bishop theorem is **also true** for Brouwer, for Markov, *and* classically.
- Avoid counterexamples by building extra data into definitions, e.g. a continuous function on $[a, b]$ comes *with* a modulus telling you how close is close enough.



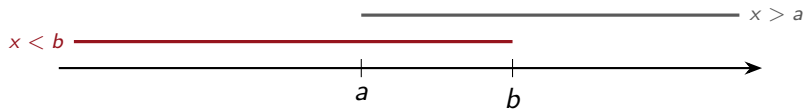
After Bridges & Richman, *Varieties of Constructive Mathematics* (1987).

Bishop's real numbers

- A real is a sequence of rationals $x_1, x_2, \ldots$ with a built-in error bound: $|x_m - x_n| \leq \frac{1}{m} + \frac{1}{n}$.
- $x < y$ needs **positive evidence**: some n where y_n exceeds x_n by more than the error allows.
- Equality and trichotomy are not decidable. The replacement:

Approximate splitting

If $a < b$, then for every x: $x > a$ or $x < b$.
(Compute x well enough and look; the overlap leaves room for error.)

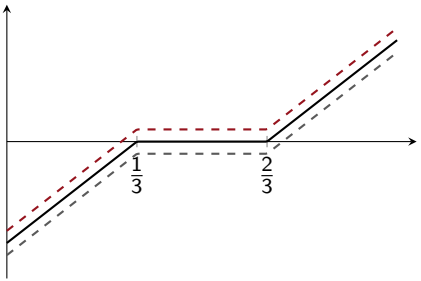


The intermediate value theorem

- Classical proof: bisect, check the sign of f at the midpoint. But "$f(m) > 0$, $= 0$ or < 0" isn't decidable.
- Trouble case: f is flat near zero. Nudge it up or down by an amount too small to detect, and the root jumps from near $\frac{1}{3}$ to near $\frac{2}{3}$.
- **Bishop's version**: For continuous $f : [a, b] \to \mathbb{R}$ with $f(a) < 0 < f(b)$,

$$\forall \varepsilon > 0 \; \exists x \in [a, b] \quad |f(x)| < \varepsilon.$$

- Exact root: fine if f is, say, strictly increasing.



Legacy

- **Martin-Löf type theory** (1970s): BHK turned into a programming language. Propositions are types; proofs are programs.
- Proof assistants: Agda, Rocq (formerly Coq), Lean.
- **Program extraction**: a constructive proof of "for every x there is a y with $P(x, y)$" yields a program computing y from x.
- Exact real arithmetic libraries use Bishop-style reals: approximations with error bounds.
- Bishop and Bridges, *Constructive Analysis* (1985) extended the 1967 book.

What counts as a construction today?

A terminating program Computes an answer after finitely many steps. We may also want a proof that it terminates.

An interactive process Responds as inputs arrive: a server, a sensor, a controller. Its future need not be known in advance, nor need it ever terminate.

A certified computation Comes with evidence of a property: an error bound, an invariant, or a proof of correctness.

Constructivity is not just about computing an answer. It is also about interaction and the evidence that a computation does what we claim.

Question: Which of these would Brouwer, Markov, and Bishop recognize as a construction?

VII. Taking stock

What happened to “function”?

Who	A function is...	Consequence
Dirichlet, Cantor	any table of pairs	monsters allowed
Brouwer	a way of answering from finite information	all continuous
Church, Turing, Kleene	a rule, possibly partial	may never return
Markov	an algorithm acting on algorithms	continuous, but no uniform bounds
Bishop	a rule plus data (moduli, bounds)	true in every setting
Martin-Löf	a program with a type	proofs are functions

The “rule vs. table” tension from last lecture, now with several rival notions of “rule”.

Progress, or lost intuitions?

Did we discover what a construction is, or did we develop several different notions of construction for different purposes?

When we formalize an intuitive activity, how can we tell whether we have captured its essence or lost something essential?

- Brouwer preserves the openness of mathematical activity but makes it difficult to identify mathematics with formal symbolic systems.
- Church and Turing give us a remarkably robust account of mechanical computation but do not thereby capture every sense of mathematical construction.
- Markov obtains a mathematically powerful account of effective objects, but loses some principles Brouwer considered essential.
- Bishop shows how far one can get without making either commitment.
- Martin-Löf connects proofs and programs, making the informational content of mathematical reasoning explicit in a formal language.

Further reading

- M. van Atten, "Luitzen Egbertus Jan Brouwer", *Stanford Encyclopedia of Philosophy*.
- D. Bridges and E. Palmgren, "Constructive Mathematics", *Stanford Encyclopedia of Philosophy*.
- P. Mancosu (ed.), *From Brouwer to Hilbert* (1998). Source texts from the 1920s debate.
- A. S. Troelstra and D. van Dalen, *Constructivism in Mathematics*, 2 vols (1988).
- D. Bridges and F. Richman, *Varieties of Constructive Mathematics* (1987).
- E. Bishop, *Foundations of Constructive Analysis* (1967).
- C. Petzold, *The Annotated Turing* (2008). Turing's 1936 paper, line by line; very accessible.
- G. H. Moore, *Zermelo's Axiom of Choice* (1982). Includes the five letters.