Alan Turing's 1936 Paper in Perspective

Rod Downey
Victoria University
Wellington
New Zealand

Bletchley Park, September 2026

The Scope of Turing's Work

- Turing worked famously on the Entscheidungsproblem
- How this had the key idea of stored program computers via universal machines....ACE
- Ideas in cryptography both breaking cryptosystems and making them for voice.
- Word problems in cancellation semigroups.
- Cryptography and Statistics.
- "Checking a Large Routine" symbolic model checking and program verification. His thesis was in this "Systems of logic based on ordinals" and looked at transfinite methods of verification.
- "Local Programming Methods and Conventions," programming methodology.
- "Rounding-off Errors in Matrix Processes" Ill-posed problems and "the other" theory of computation.

- Intelligent Machinery and the Turing test (Avi Wigderson lecture 2)
- Computer chess (before stored program computers)
- Absolute normality (Veronica Becher's Lecture)
- Morphogenesis and the first numerical nonlinear PDE's.

Plan

- Turing's 1936 paper concerned itself with a solution to a very famous decision problem from logic.
- It did much, much more than that.
- I will attempt to explain some of this and how he did it.

Born of logic

- Thanks to Moshe Vardi for this and the next quote (my highlighting).
- Cosma R. Shalizi, Santa Fe Institute.

  If, in 1901, a talented and sympathetic outsider had been called upon (say by a granting agency) to survey the sciences an name a branch that would the the least fruitful in the century ahead, his choice might well have settled upon mathematical logic, and exceedingly recondite field whose practicioners could all have fit into a small auditorium. It had no practical applications, and not even that much mathematics to show for itself: its crown was an exceedingly obscure definition of cardinal numbers.

More recently

- Martin Davis (1988) *Influences of Mathematical Logic on Computer Science.*

  > *When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization.*

- Read a somewhat dated but wonderful collection in the Bulletin of Symbolic Logic: On the Unusual Effectiveness of Logic in Computer Science (Halpern, Harper, Immerman, Kolaitis, and Vardi).
- Echoes Wigner's 1960 article "The unreasonable effectiveness of mathematics in the natural sciences," and Galileo's "The book of nature is writ in the language of mathematics."

Formal logic

- Logic studies principles of correct reasoning.
- We represent reasoning and knowledge by symbols.
- Then manipulate the symbols using certain rules of inference (depending on the logics) to make conclusions.
- Logics include modal logics which are used to understand "possible worlds" $\diamond P$ means "P is possible", heavily used in various forms in program verification, quantum logics, fuzzy logics ("P is likely to happen"), threshold logics, which are used in neural nets, etc.
- Logic is the only part of mathematics that takes language seriously.

Propositional logics

- The simplest system, used to torture first year students, and developed by Thomas Boole in the early 19th century.
- A proposition is a statement that is either true (1) or false (0). e.g. The UK has a Tory government; represented by N, say, and false.
- We analyse compound statements made up from connectives such as "and" $\wedge$, "or" $\vee$, "not" $\neg$, "implies" $\rightarrow$.
- we can define using truth tables

P	Q	$P \wedge Q$	$P \vee Q$	$\neg P$	$P \rightarrow Q$
1	1	1	1	0	1
1	0	0	1	0	0
0	1	0	1	1	1
0	0	0	0	1	1

Truth tables

- We can test statement/arguments in this logic.
  If Jane is a scout, then Fred is a pilot. Jane is a scout. Therefore Fred is a pilot.
- This is called modus ponens and would be represented by
  $J \to F$; J
  $\therefore F$.
- Note the argument below not true:
  If Jane is a scout, then Fred is a pilot. Fred is a pilot. Therefore Jane is a scout. This is a common fallacy of formal reasoning. ("Post hoc ergo propter hoc".)
- If you think nobody would use this fallacy, I advise you to look up The L'Aquila Earthquake of 2009 Legal Case.

Algorithms for truth

- We can test by big truth tables. e.g.
  $(P \wedge (Q \vee R)) \wedge ((P \wedge Q) \wedge \neg(P \to R))$

P	Q	R	$(P \wedge (Q \vee R)) \wedge ((P \wedge Q) \wedge \neg(P \to R))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

- This is called satisfiable as at at least one line is true.
- Note that every new variable doubles the size of the table.

Want to be rich and famous?

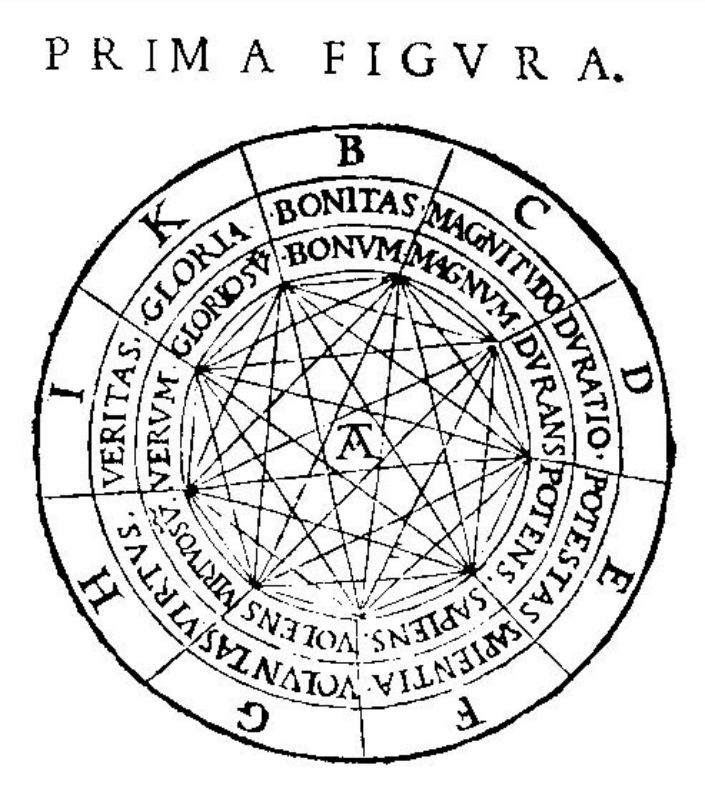
- the SAT problem asks for an efficient method for determining if a statement of propositional logic is satisfiable.
- But we have a method: truth tables!
- The problem is that if we had 100,000 variables (which happens in commercial applications) then generating the truth table would take more time than available in the universe.
- On the other hand we can guess a satisfying assignment of trues and falses and check easily. The $P \neq NP$ conjecture says that there is no efficient method taking e.g. $100,000^3$ steps. (A $1,000,000 Clay Prize.)
- Why do we care? If $P = NP$ (reasonably) then all modern cryptosystems will be insecure. Modern banking would fail. But lots of algorithms such as scheduling, voice recognition, etc would become much much faster.

Entscheidungsproblem

- The most famous mathematician of his generation, David Hilbert, famously asked for a decision procedure for number theory.
- This was born of 19th centure determinism which imagined the universe as a big machine whose path was completely determined.
- Of course this version is (perhaps) still open: is the universe mechanical; can the universe produce incomputability?
- Still others ask can the universe produce anything that is computable, or is everything random?
- This is too big for my brain and I will stick with questions in formal logic and number theory.
- Notice a weaker question is can everything that is true of some formal system be proved in such a system? (completeness)

Llull

- Raymond Llull (14th C) was one the the first to try to mechanize reasoning.
- He had a reckoner to try to calculate the aspects of god.



Llull based this notion on the idea that there were a limited number of basic, undeniable truths in all fields of knowledge, and that everything about these fields of knowledge could be understood by studying combinations of these elemental truths.

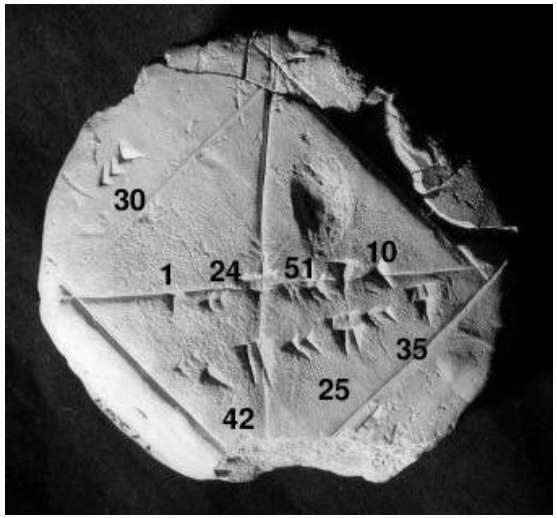
Leibnitz

- Leibnitz believed that much of human reasoning could be reduced to calculations.
- Invented calculus ratiocinator, which resembles symbolic logic,

  The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate, without further ado, to see who is right. (1685)

Algorithms; Calculation

- Intuitive algorithms have been in the centre of maths, right up to the 20th Century.
- The earliest examples of algorithms we know can be found from e.g.



- This came from an approximation to $\sqrt{2}$ by the Babylonians nearly 4,000 years ago.

$$1 + \frac{24}{60} + \frac{51}{60^2} + \frac{10}{60^3} \approx 1.414213.$$

- Archimedes approximating the area of a circle (and hence π) using "polygons". He used a 96-gon. That is, regular, and has 96 sides.

- Amazingly his ideas still influence us today (another talk!)
- 265 AD Liu Hui, used a 3072 sided polygon to get 3.1416, and then in 480 AD Zu Chongzhi, used this algorithm applied to a 12,288 sided (!) polygon to get $\pi \approx 3.141592920 \approx \frac{355}{113}$. This is the best approximation for 800 years thereafter!
- Then others with the zenith being Christoph Grienberger (Austria) in 1630 using a polygon of 10^{40} (!!) many sides, giving a value known to be correct to 39 decimal places.
- Most of historical mathematics was concerned with calculation.
- To my knowledge all calculating devices were task specific.

Back to Logic-History

- In
  the west began with the Greeks, earliest Aristotle via syllogisms such as:
  All humans are mortal.
  All Greeks are humans.
  Therefore, All Greeks are mortal.
- Interestingly, while the Greeks had no symbolic representation, they worked within a rich logic called predicate logic. Nowadays, we'd write this as:
  $\forall x(H_x \to M_x)$.
  $\forall x(G_x \to H_x)$
  $\therefore \forall x(G_x \to M_x)$.
- Some Greeks have no children : $\exists x(G_x \wedge \neg\exists y Cxy)$.
- This logic allows us to say all, or some, of the individuals have a property. The truth of H_x depends on which x is "interpreted".

Entscheidungsproblem

- The most famous mathematician of his generation, David Hilbert, famously asked for a decision procedure for predicate logic like we had with truth tables. It is reasonable to believe that he thought there is such a method.
- This was born of 19th century determinism which imagined the universe as a big machine whose path was completely determined.
- Hilbert wanted a *formal system* where everything that is true be proved within the system? (completeness)
- (A proof is a special kind of algorithm, where we deduce statements from axioms and rules of inference.)

- Can we create an algorithm, a machine, into which one feeds a statement about mathematics or at least in a reasonable "formal system" i.e. formal logic, and from the other end a decision emerges: true or false.
- Or, for a given formal system, can we eventually produce proofs of all the "truths" of that system.
- Hilbert also proposed that we should prove the consistency of mathematics; i.e. know we can't prove a contradiction.

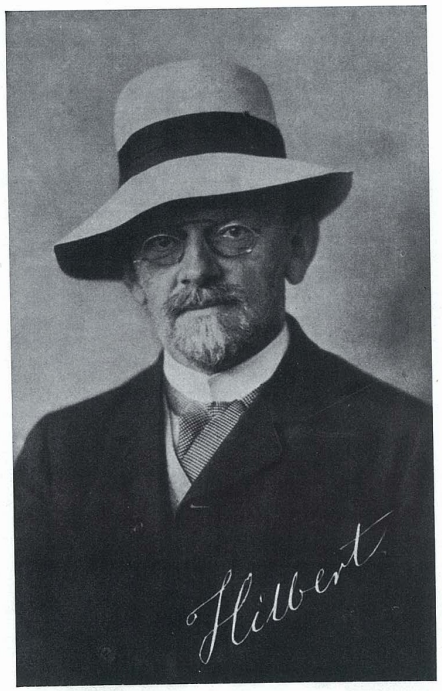
The Dark Lord

- Leibnitz' dreams and Hilbert's dreams were forever shattered by the ideas of a young mathematician, Kurt Gödel.

Gödel

- He proved the two incompleteness theorems.
- The first incompleteness theorem says that any sufficiently rich formal system has statements
- expressible in the system
- true of the system, but
- cannot be proven in the system.
- Secondly no sufficiently rich formal system can prove its own consistency.
- That is, we have no idea if our systems are consistent!
- The collective intuition of a generation of mathematicians was wrong.

Maybe there is hope



David Hilbert, 1912 — one of a group of portraits of professors which were sold as postcards in Göttingen

- Showing the Entscheidungsproblem undecidable asks for the defeat of any mechanical method. (i.e. not just proofs in a logic system)

More Prehistory

- Hilbert himself famously used nonconstructive methods in his proof of the Hilbert Basis Theorem (finite bases for certain ideals, whilst giving no idea how to construct them). (See Metakides and Nerode, 1982)
- At Hilbert's time authors had an intuitive understanding of what they meant by an algorithm.
- For example, Borel 1912 when discussing an algorithmic way to approximate real numbers:

  > *"I intentionally leave aside the practical length of operations, which can be shorter or longer'; the essential point is that each operation can be executed in finite time with a safe method which is unambiguous"*

- Also the first edition of van der Waerden's book was all algorithmic methods (based on Emmy Noether's notes).
- But how to make precise?

The confluence of ideas in 1936

- Gödel's work involved proofs, but the Entscheidungsproblem was still open. Proofs were special algorithms.
- To answer it we'd need to define what mechanical method actually means.

- What is being asked is to prove there is no method ; or to give one.
- To Prove NO The first thing you must do is to give something that models all such methods.
- Then prove that that model won't do it.
- How to model human thought? Or at least "human decision procedures"?
- Note that is more than mathematics it is a *philosophical* question.
- First Church, then Turing and Post proposed models for computation.
- Turing : Turing machine.
- At 24, whilst on a run at Grantchester Meadows, Turing devised a brilliantly convincing model.

Turing Machine-Model Thought

- Machine is a Box with a finite number of Internal States (i.e. mental states)
- Reads/writes on a two way potentially infinite tape.
- Action : can move Left, Right, or (over-)Print a symbol,
- Depending on (state, symbol)

Why Turing?

- First and foremost Turing has a conceptual analysis giving what many regard as a proof of the thesis that TM's capture what is computable.
- This analysis is a fundamental contribution of Turing's paper.
- See "The Universal Turing Machine: A Half Century Survey" R. Herken (ed) Springer 1995 (2nd Ed).

Turing's analysis

- He considers an abstract human computor
- By limitations of sensory and mental apparatus we have
  (i) fixed bound for the symbols.
  (ii) fixed bound for number of squares
  (iii) fixed bound to the number of actions at each step
  (iv) fixed bound on the movement.
  (v) fixed bound on the number of states.
- Gandy, Soare (and others) argue that Turing proves any function calculable by an abstract human is computable by a TM.

- Gandy (1995):

  What Turing did, by his analysis of the processes and limitations of calculations of human beings, ... was to show that calculation can be broken down into the iteration (controlled by a "program") of extremely simple concrete operations; so concrete that they can easily be described in terms of (physical) mechanisms.

The Universal Machine

- Building in Gödel's notion of arithmetization (a crucial step in his proof of the incompleteness theorem), another major contribution was the notion of a universal machine, a compiler.
- The idea that there could be a single machine which interpreted programs to emulate any other machine.
- It is so easy now for us to think of everything as data, but these are the papers this idea came from!

Turing said in a lecture of 1947 with his design of ACE (automated computing engine)

> *The special machine may be called the universal machine; it works in the following quite simple manner. When we have decided what machine we wish to imitate we punch a description of it on the tape of the universal machine... . The universal machine has only to keep looking at this description in order to find out what it should do at each stage. Thus the complexity of the machine to be imitated is concentrated in the tape and does not appear in the universal machine proper in any way... . [D]igital computing machines such as the ACE ... are in fact practical versions of the universal machine.*

The proof

- Turing's proof (as we would now do it) is to code an algorithmically undecidable problem into predicate logic using what we would now call a generic reduction.
- That is, show that machine actions could be emunated faithfully by sentences in the logic.
- The problem we would use is :
  INPUT : (x, y)
  QUESTION : Does machine number x halt on input number y?
- This is a general roadmap to many undecidability proofs.
- Church-Kleene's ingenious solution did not use the "halting problem" encoded, except implicitly (this is a slightly tricky point and you need to (try to) read the original papers). They are difficult reads.

Prehistory and post-history

- Babbage said of his Analytical Engine (not a stored program machine) "it could do anything except compose country dances." (quoted in Huskey and Huskey 1980, p 300)
- Actually now computers do compose country dances.
- The idea that a computer could be universal was a long time penetrating.
- Howard Aitken (1956), a US computer expert of the time:

  If it should turn out that the basic logics of a machine designed for numerical solution of differential equations coincide with the logics of a machine intended to make bills for a department store, I would regard this as the most amazing coincidence that I have ever encountered.

- Read more on this in Martin Davis' or Herken's books.

The Birth of Computers

- The birth of computers is a topic that gets people quite agitated.
- Read, for instance, "Who Begat Computing" (Communications of the ACM) by Moshe Vardi.

> *"While these ideas undoubtedly influenced the efforts of John von Neumann and his collaborators at the University of Pennsylvania in the 1940s, we should not confuse a mathematical idea with an engineering design. It was the EDVAC Report of 1945 that offered the first explicit exposition of the stored-program computer. Turing's ACE Report, which elaborated on this idea and cited the EDVAC Report, was submitted in early 1946. "*

- It is clear that is some kind of combination of theory (such as Turing, Gödel, Church, etc), and engineering.
- Scholars have made careers studying this, and I am no such scholar, and offer some brief comments with trepidation:

The Birth of Computers

- McCulloch and Pitt used Turing ideas to show the control mechanism for a TM could be simulated by a finite collection of gates (1943).

John von Neumann, Princeton, 1932

- Von Neumann knew of Turing's ideas and with two other co-authors proposed a practical architecture for stored program machines. He uses the McCulloch and Pitt ideas. (1945) EDIAC.
- Williams, Kilburn and Tootill, Manchester Baby first electronic storage machine. (1948)
- Turing's pilot ACE was around 1949-50.

- Stanley Frankel (friend of von Neumann)

  von Neumann was well aware of the fundamental importance of Turing's paper of 1936 'On computable numbers ...', which describes in principle the 'Universal Computer'.... Many people have acclaimed von Neumann as the 'father of the computer' (in a modern sense of the term) but I am sure that he would never have made that mistake himself. He might well be called the midwife, perhaps, but he firmly emphasized to me, and to others I am sure, that the fundamental conception is owing to Turing

- Von Neumann, could be the most brilliant person ever, but
- Advocated *A*-bombing Kyoto (and not Nagasaki/Hiroshima) as it would have more effect on the Japanese soul.
- Post WW2

  "If you say why not bomb them [the Russians] tomorrow, I say why not bomb them today? If you say today at five o'clock, I say why not one o'clock?" (1950)

Rest of the talk

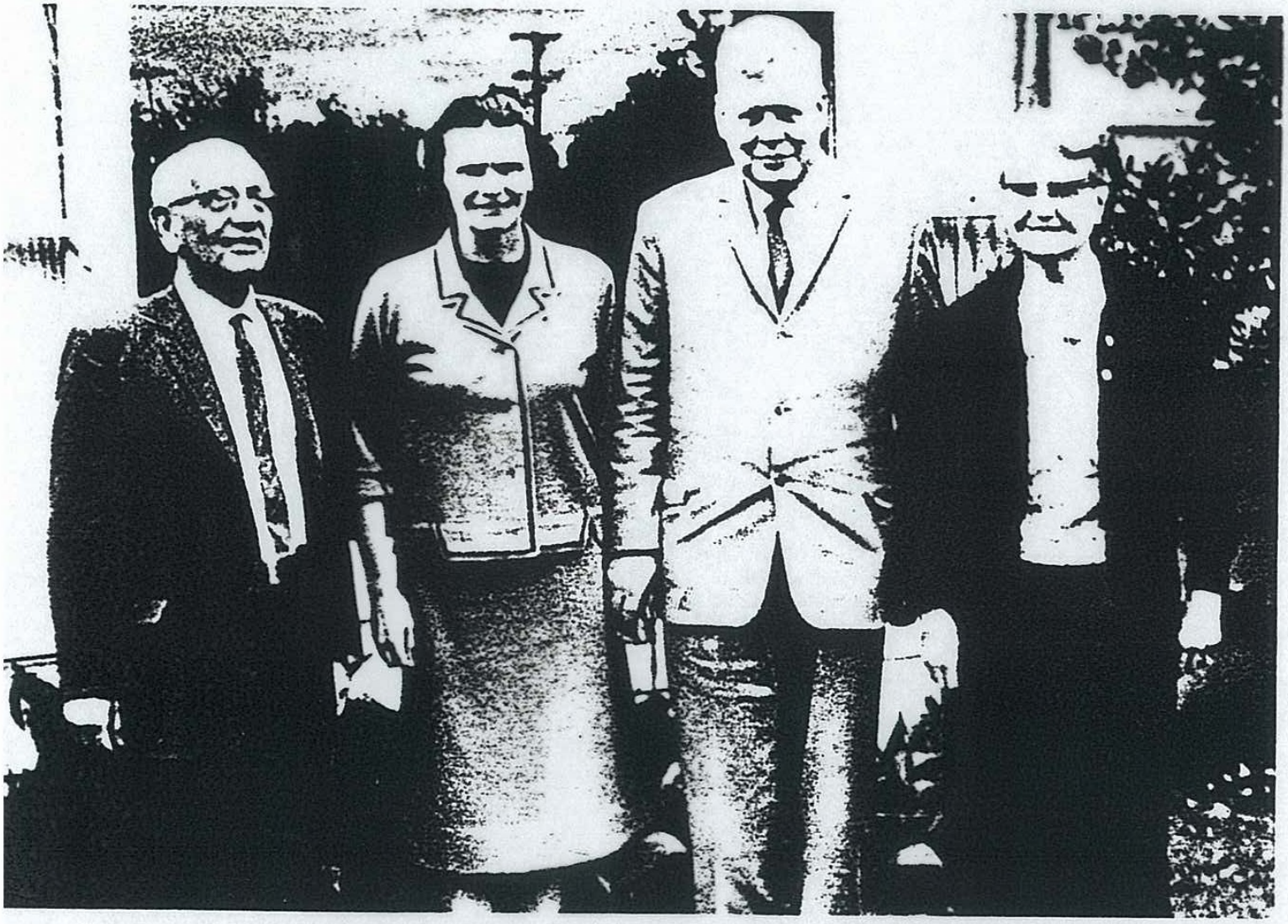
- I will discuss some of the areas of research in maths and CS which directly emanated from Turing's 1936 paper, especially in combination with his 1939 paper.

Lots of Undecidable Problems

- Many algorithmic problems are undecidable.
- I'll include some below which used generic reductions and had a big effect on maths and CS.
- For example:
  INPUT a set of square coloured tiles of the same size. Only same colour borders next to one another.
  QUESTION Can an initial configuration be extended to colour the plane?
- Wang in the 60's showed that there is no algorithm to decide this.

Hilbert's 10th problem

- INPUT A integer polynomial P in variables $x_1, ..., x_n$
  QUESTION Is there a positive integer solution to the equation $P = 0$?
- Matijasevich, using work of Martin Davis, Hilary Purtnam, and Julia Robinson in the 70's showed there is no algorithm to decide such questions.
- But there is now a polynomial whose only positive integer zeroes are the primes! (This is bacuse the reduction was so generic it showed many problems were diophantine.)
- Alexandra Shlapentokh will be speaking on related matters.



This shows myself, Julia Robinson, Raphael Robinson, and my wife.

$$
\begin{aligned}
Q(a,\ldots,z) = (k+2)\{&1-[wz+h+j-q]^2\\
&-[(gk+2g+k+1)(h+j)+h-z]^2\\
&-[2n+p+q+z-e]^2-[16(k+1)^3(k+2)(n+1)^2+1-f^2]^2\\
&-[e^3(e+2)(a+1)^2+1-o^2]^2-[(a^2-1)y^2+1-x^2]^2\\
&-[16r^2y^4(a^2-1)+1-u^2]^2\\
&-[((a+u^2(u^2-a))^2-1)(n+4dy)^2+1-(x+cu)^2]^2\\
&-[n+1+v-y]^2\\
&-[(a^2-1)l^2+1-m^2]^2-[ai+k+1-l-i]^2\\
&-[p+l(a-n-1)+b(2an+2a-n^2-2n-2)-m]^2\\
&-[q+y(a-p-1)+s(2ap+2a-p^2-2p-2)-x]^2\\
&-[z+pl(a-p)+t(2ap-p^2-1)-pm]^2\}.
\end{aligned}
$$

Word Problems

- Novikov-Boone: No algorithm to decide whether a word is trivial in a finitely presented group. Similarly whether two finitely presented groups are isomorphic. (Questions of Dehn, 1911.)
- Dehn gave some examples where we can decide, but Novikov and Boone showed how to faithfully simulate Turing Machines in finitely presented groups.
- Led to combinatorial group theory; showing that understanding the algorithmic content of mathematics needs deep understanding of the fine structure.
- Similar comments for Hilbert's Basis Theorem and Gröbner Basis Theory. To actually find the finite basis requires much deeper understanding of the theory of invariants.

Topology

- Markov showed that for dimension $n \geq 4$ there is no algorithm to decide if two n-manifolds are homeomorphic. (They are presented as simplicial manifolds.)
- There are algorithms for dimensions 2 and 3; 3 depends on very famous work of Perelman (2003).

Recent Examples

- Nabutovsky and Weinberger (Geometrica Dedicata, 2003) showed that basins of attraction in differential geometry faithfully emulated certain computations. Refer to Soare Bull. Symbolic Logic.
- Remark: Earlier work by Lee Rubel on the existence of universal PDE's.
- (Hochman and Meyerovitch) The values of entropies of subshifts of finite type over $\mathbb{Z}^d$ for $d \geq 2$ are exactly the complements of "halting probabilities". Again the proof is a generic emulalation.
- In Physics, recent work by Cubitt, Perez-Garcia, and Wolf showing the undecidability of the "spectral gap" for Hamiltonians.

Biology

- This is very closely related to DNA.
- Early work on cellular automata.
- DNA can be thought of as a programming language.
- Recent "self assembly" modelled by tiles where certain tiles can "bond" (in a nondeterministic way) with others. (Winfree, only verified by experiment 15 years later)
- Code universal Turing machines.
- Currently going back to construct nano-scale molecular shapes (e.g. "smiley faces") at will.

Computability Theory

- The concept of relative computation from Turing's 1936 and 1939 papers using "oracles" has seen a lot of development.
- How does one calibrate the relative computational complexity of problems?
- And how does this relate to algebraic and analytic structure?
- For example, we know that Turing Reducibility (from Turing 1939) correlates to quantifier alternations and leads to various hierarchies used extensively in logic.
- In proof theory, model theory, set theory, computational complexity theory.
- Julia Knight will have something to say here.

e.g. Using computation to show No Invariants

- There has been massive development in computability theory and I will mention only one.
- Much of mathematics is concerned with classification of structures (groups rings, DE's etc) by invariants.
- Bases for vector spaces, Ulm invariants for abelian groups.
- How can we show that no invariants are possible?
- A computability theorists's view.

Analytic=Σ^1_1

- The halting problem is called Σ^0_1 in that $\varphi_x(y)$ halts iff

$$\exists t \in \mathbb{N} \varphi_x(y)$$

  halts in t steps. (And $\varphi_x(y)$ halts in t steps is computable.) This is arithmetic, where the quantifier searches over $\mathbb{N}$.
- Almost all problems in normal mathematics are analytic.
- A is analytic or Σ^1_1 iff deciding $x \in A$ entails asking if there is a function f from $\mathbb{N}$ to $\mathbb{N}$ such that some computable relation holds for all $f(n)$.
- E.g. isomorphism is typically in Σ^1_1.

Σ^1_1 completeness

- Many problems in Σ^1_1 are much easier. E.g. isomorphism for finitely presented groups is Σ^0_3. (Is there a matching of generators for which every equation in the first holds in the second?)
- If some problem is shown to be Σ^1_1 complete, then no simpler set of invariants is possible.
- E.g. (Downey and Montalbán) the problem of deciding if two finitely presented groups have $H_i(G) \cong H_i(\hat{G})$ for $i \leq 3$ is Σ^1_1 complete.
- Uses the result that the isomorphism problem for computable torsion free Abelian groups is Σ^1_1 complete. (DM)

Computable Analysis

- It is noteworthy that Turing's paper was actually concerned not with finite objects but real functions.
- He considered computably continuous functions acting on the field of computable reals, which are those with computable Cauchy sequences (something he got wrong in the paper but corrected later).
- Now we think of algorithmic analysis via computable operators acting on fast Cauchy sequencs.
- e.g. (Pour-El and Richards) An linear operator on a computable Banach space is effective on computable inputs iff it is bounded.
- e.g. (Braverman and Yampolsky) that Julia sets can be noncomputable, any halting problem being codable. (Also Blum-Smale-Shub, but that's another story.)
- Julia set: $z \mapsto z^2 + \alpha z$, where $\alpha = e^{2\pi i \theta}$.
- These two traditions of discrete and analtyic computable mathematics diverged, but have recently re-converged via algorithmic duality theory. (See, or even better buy, my book with Melnikov-I need retirement income).

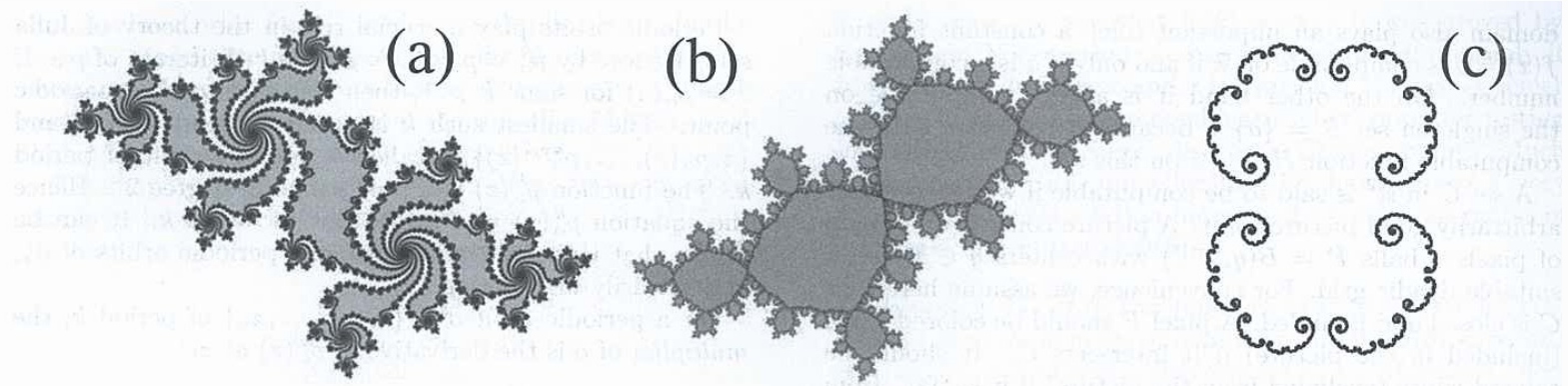


Figure 1: Examples of quadratic Julia sets J_p (black), and filled Julia sets K_p (gray); orbits that originate at white points escape to ∞; note that on picture (c) $K_p = J_p$, since K_p has empty interior

Computational Complexity Theory

- The ideas stemming from Turing's 1936 paper led to computability theory and thence to computational complexity theory.
- That is measure the resources needed to compute things: How many steps? How much memory(space)?
- Hartmanis and Sterns (1960's) put clocks on Turing machines and along with Edmonds proposed polynomial time and space as outer measures of feasibility.
- Miniaturization of the Turing undecidability proofs led to Levin and Cook proposing the model of NP-completeness, via a generic reduction from a nondeterministic Turing Machine.
- So we again see Turing's ideas flowering beyond what might be expected.

Algorithmic Randomness

- How to give meaning to randomness for individual sequences?
- Observe that it it passes enough computational randomness "tests" as far as normal processes are concernes it will act in the expected way.
- For example, α is computably random if no computable fair betting strategy betting sequentially on the bits of α can win infinite capital.
- Computable randomness is enough to provably accelerate the computation of some computable language by more than a polynomial amount.
- Polynomial randomness is enough to guarantee "normality" in number theory.
- Huge area; Turing used ideas from his 1936 paper to anticipate this, and Veronica Becher will tell us about this in another lecture. (Peter Millican (2021) argues that "definability" considerations like these were the original motivation for Turing's 1936 paper, and it was only during the writing that he realized that he could also some the Entschiendungsproblem.)

Some paradoxes

- I'll finish with some deep open questions and apparent paradoxes; showing we've not finished with these stories.
- Many many problems are undecidable, and Turing proved that, for example, we can't decide if two reals are equal, yet:
- Algorithms based on the real RAM model and complexity based on counting arithmetical operations often work very well in practice.
- Some solutions can be based on generic case complexity, as per Kapovich et. al. in combinatarial group theory.
- Question Explain.
- Look at smoothed analysis etc.

Another Paradox

- We can easily show that many many problems we really care about can be converted into instances of SAT. 50 years of research has generated modern SAT SOLVERS which work very well on instances which come from real data. E.g. NASA uses this for its robot navigation.
- We have no idea why they work. I love this question.
- If we understood this we'd be able to revolutionize algorithm design.

> *"The need for deeply understanding when algorithms work (or not) has never been greater"*

(T. Roughgarden-Beyond Worst Case Analysis-Communications Association for Computing Machinery-2019)

Yet another

- Modern AI engines in some sense give up on intelligent design of algorithms and use statistical learning and calculus.
- You build a big annotated graph of vectors (representing various aspects of "tokens") and then use training of some kind to iteratively refine the model using some kind of Bayesian refinement method.
- Why do they work?
- And in what sense do they work?
- Is this intellegence? What is intelligence anyway?
- In some sense we are back to the question of intelligent design vs evolution.

Summary

- We live in the most mathematical age of all time.
- Turing's work in his 1936 (and other) papers still resonates today.
- I have given a brief account of some the history of how we got here.
- Logic was the mother of all of this.
- And pointed out that all of this came from Blue Skies Research.

Forbidden Fruit

- Thanks for Listening