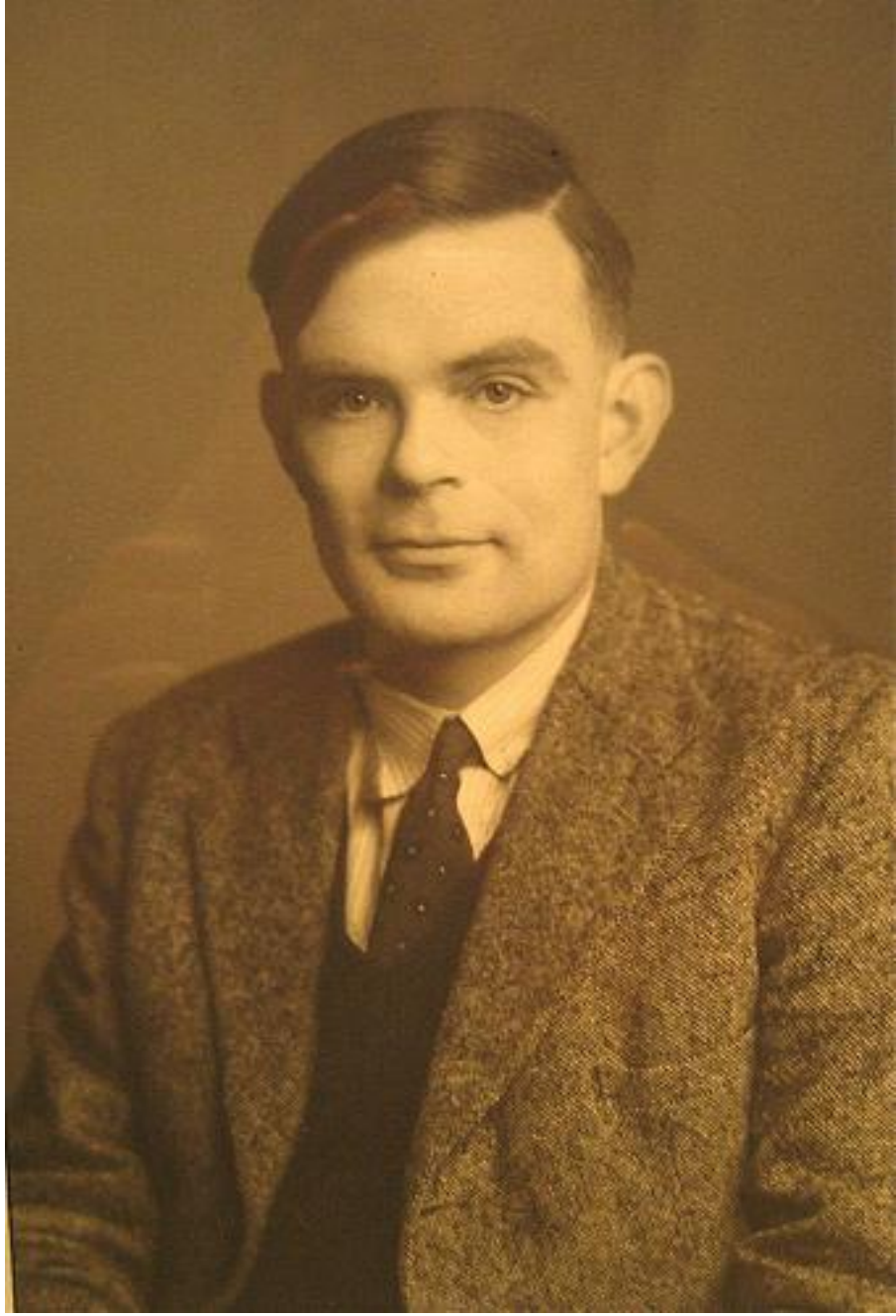

In memory of
Alan M. Turing
Christoph Stamm | Roger Wattenhofer
ETH Zurich – Distributed Computing – www.disco.ethz.ch

Overview

- Greek Philosophers
- Augustus De Morgan
- George Boole
- Gottlob Frege
- Giuseppe Peano
- Alfred North Whitehead
- Bertrand Russell
- David Hilbert
- Wilhelm Ackermann
- Emil Leon Post
- Kurt Gödel
- Alonzo Church
- Stephen Cole Kleene
- Alan Mathison Turing
- Turing Machine
- Church-Post-Turing Thesis
- Decidability
- Universal Turing Machine
- Circle-free Turing Machines
- Halting Problem
- Undecidability of the Entscheidungsproblem
- P and NP
- Turing Test

Alan Mathison Turing

- 1912 – 1954, British mathematician
- one of the fathers of computer science
- his computer model – the Turing Machine – was inspiration/premonition of the electronic computer that came two decades later
- during World War II he worked on breaking German cyphers, particularly the Enigma machine.
- Invented the “Turing Test” used in Artificial Intelligence
- Legacy: The Turing Award. “Nobel prize” in computer science



Alan Turing's Academic Career

- 1931: King's College, University of Cambridge, England
  - Profs: Newman, Russell, Wittgenstein
  - Interests
    - group theory
    - probability theory
    - formal logic
  - Thesis: proof of a main theorem in statistics
- 1935: Fellow of King's College
  - interests: Entscheidungsproblem
- 1936: PhD student in Princeton
  - student of Alonzo Church
  - equivalence of Turing machines and λ-Calculus
- 1945: National Physical Laboratory
  - works on the design of the automatic computing engine (stored-program computer)

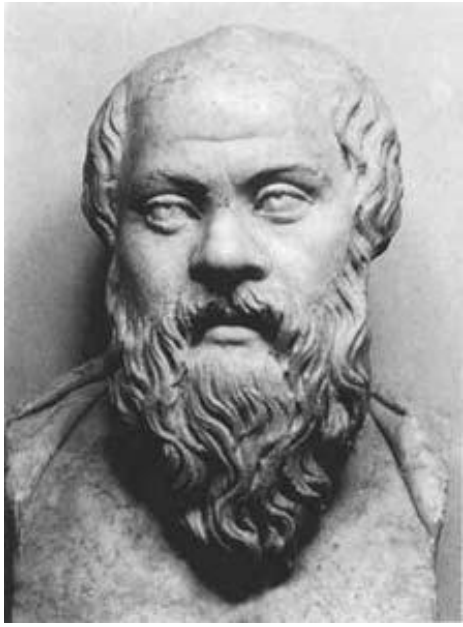
- 1948: Manchester University
  - joined Max Newman's Computing Laboratory
  - worked on software for Mark 1
  - addressed the problem of artificial intelligence (Turing Test)
  - became interested in mathematical biology

Inference and Inductive Reasoning (Greek Philosophers)

- Inference or deduction is the act or process of deriving logical conclusions from premises known or assumed to be true.
- The process by which a conclusion is inferred from multiple observations is called inductive reasoning.
- The conclusion may be correct or incorrect, or correct to within a certain degree of accuracy, or correct in certain situations. Conclusions inferred from multiple observations may be tested by additional observations.

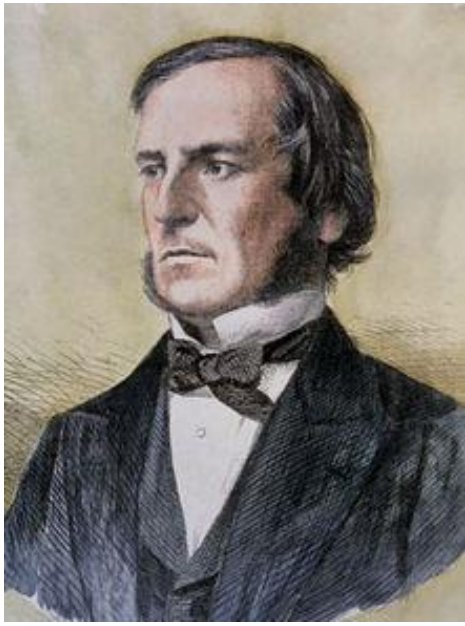
- Inference Example
  1. All men are mortal. [inductive reasoning]
  2. Socrates is a man.
  3. Therefore, Socrates is mortal.

- Big question: Can this process be automated?
  - The laws of valid inference are studied in the field of logic.
  - Are there other beings than men, which can draw logical conclusions?



Formalization of Propositional Logic (19th Century)

- Augustus De Morgan (1806 – 1871)
  - British mathematician and logician (Trinity College)
  - “First Notions of Logic” (1840), De Morgan's laws
  - “Formal Logic or The Calculus of Inference” (1847)
- George Boole (1815 – 1864)
  - English mathematician, philosopher and logician
  - “The Mathematical Analysis of Logic” (1847)
  - “An Investigation of The Laws of Thought” (1854)
    - algebraic system of logic → Boolean algebra



- Propositional Logic
  (Propositional Calculus, Sentential Calculus)
  - Examples
    - proposition P: “it rains for an hour”
    - proposition Q: “the ground is wet”
    - satisfiable formula $F = P \rightarrow Q \equiv \neg P \vee Q$
    - valid formula $G = \neg F \vee F$

Propositional Logic: Formal Definition and Syntax

- Definition: A propositional logic is a formal system $\Pi = (\mathbf{A}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{I})$.
  - $\mathbf{A}$ is a finite set of proposition variables (atoms), e.g. P, Q, …
  - $\mathbf{\Omega}$ is a finite set of operator symbols, e.g. {⊥, ⊤, ¬, ∨, ∧, →, ↔}.
  - $\mathbf{Z}$ is a finite set of transformation (inference) rules, e.g. De Morgan's laws.
  - $\mathbf{I}$ is a finite set of starting points, e.g. the logical formulas that are assumed to be true without controversy (= axioms). Can be empty.

- Syntax**:** The language L of Π is the set of formulas. It is inductively defined by the following rules:
  - base case: any element of $\mathbf{A}$ is a formula of L.
  - inductive case: if P_i are formulas and $\omega \in \Omega$, then $\omega(P_1, P_2, \ldots, P_k)$ is a formula

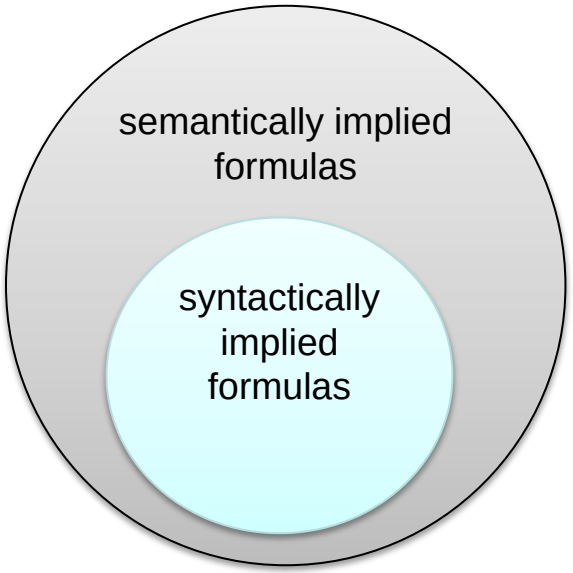
Propositional Logic: Semantic

- Semantic: A truth assignment is a function f: $\mathbf{A} \times$ {false, true}.
  - f satisfies P $\in \mathbf{A}$ if and only if f(P) = true
  - for each $\omega \in \Omega$ the semantic defines under what conditions a ω–transformed formula is satisfiable by f, e.g.
    - $\bot$ is never satisfied, $\top$ is always satisfied
    - ¬F is satisfied if and only if F is not satisfied
    - (F $\vee$ G) is satisfied if and only if at least one of either F or G are satisfied
    - (F $\wedge$ G) is satisfied if and only if both F and G are satisfied
    - (F $\rightarrow$ G) is satisfied if and only if it is not the case that F is satisfied but not G

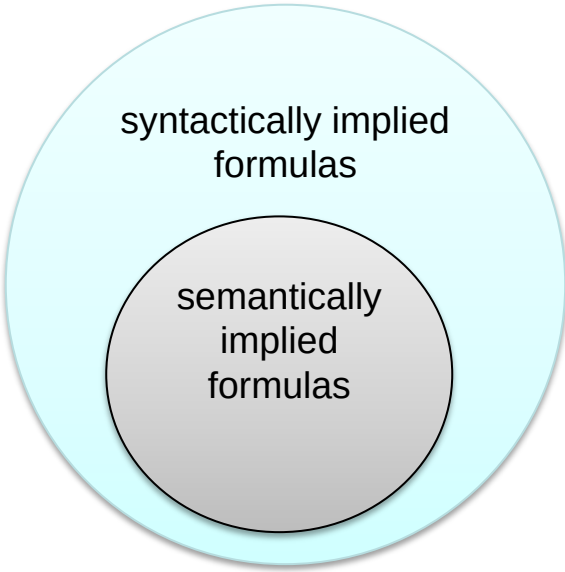
- Semantic inference
  - a set of formulas S semantically implies a formula G if all truth assignments f that satisfy all the formulas in S also satisfy G
- Syntactic inference
  - a set of formulas S syntactically implies a formula G if and only if we can derive G from S with the inference rules $\mathbf{Z}$ in a finite number of steps

Propositional Logic: Soundness and Completeness

- Soundness (Consistency)
  - if the set of formulas *S* syntactically implies the formula G, then *S* semantically implies G
  - semantic inference is a necessary condition for syntactic inference



- Completeness
  - if the set of formulas *S* semantically implies the formula G, then *S* syntactically implies G
  - semantic inference is a sufficient condition for syntactic inference



Formalization of Second-order Logic (19th Century)

- Gottlob Frege (1848 – 1925)
  - German mathematician, logician and philosopher (Jena, Göttingen)
  - wanted to show that mathematics grows out of logic
  - rigorous treatment of the ideas of functions and quantified variables and sets
  - "Begriffsschrift. Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens." (1879)
  - "Grundgesetze der Arithmetik", Band 2 (1903)

First-order logic is an extension of propositional logic

- new elements: quantified variables, functions, predicates, relations
- Example
  predicate P(a): "a is a philosopher"
  predicate S(a): "a is a scholar"
  formula F = $\forall a$: P(a) $\rightarrow$ S(a)
  formula G = $\neg\exists a$: P(a) $\wedge$ $\neg$S(a)

Second-order logic is an extension of first-order logic

- new elements: variables that range over sets of individuals
- Example
  set P $\subset$ Domain
  formula F = $\forall P\ \forall x$: x$\in$P $\vee$ x$\notin$P

Peano Axioms for Natural Numbers (1889)

- Giuseppe Peano (1858 – 1932)
  - Italian mathematician (University of Turin)
- Peano Axioms: set of axioms for the natural numbers
  - 1^{st}: asserts the existence of at least one member of $\mathbb{N}$
  - 2^{nd} to 5^{th}: general statements about equality
  - 6^{th} to 8^{th}: first-order statements about $\mathbb{N}$ expressing the fundamental properties of the successor operation
  - 9^{th}: second-order statement of the principle of mathematical induction over $\mathbb{N}$

- Peano Arithmetic (in combination with the first eight axioms)
  - Addition

    $\forall a: a + 0 = a$

    $\forall a \forall b: a + S(b) = S(a + b)$
  - Multiplication

    $\forall a: a \cdot 0 = 0$

    $\forall a \forall b: a \cdot S(b) = a + (a \cdot b)$

Mathematical Induction: Example

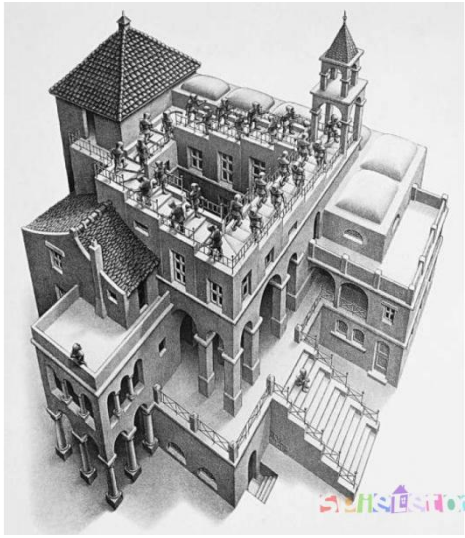
- 1st axiom: $0 \in \mathbb{N}$
- 2nd axiom: $\forall x \in \mathbb{N}: (x = x)$
- 6th axiom: $\forall x \in \mathbb{N}: S(x) \in \mathbb{N}$
  - unary representation of $\mathbb{N}$ = { 0, S(0), S(S(0)), S(S(S(0))), … }
- 7th axiom: $\forall x \in \mathbb{N}: \neg(S(x) = 0)$
- 8th axiom: $\forall x \in \mathbb{N}\ \forall y \in \mathbb{N} : (S(x) = S(y)) \rightarrow (x = y)$
- 9th axiom
  - let P be a unary predicate
  - $(P(0) \land \forall n \in \mathbb{N}: P(n) \rightarrow P(S(n))) \rightarrow \forall x \in \mathbb{N} : P(x)$
- Example
  - let $P(x)$ be $((x = 0) \lor (\exists b : S(b) = x))$
  - we want to show that P(x) is valid for all $x \in \mathbb{N}$
  - base case: $P(0) \equiv ((0 = 0) \lor (\exists b : S(b) = 0)) \equiv (\text{true} \lor (\exists b : S(b) = 0)) \equiv \text{true}$
  - inductive case: if $\forall n \in \mathbb{N}: P(n)$ is true then $\forall n \in \mathbb{N}: P(S(n))$ has to be true, too

    $P(S(n)) \equiv ((S(n) = 0) \lor (\exists b : S(b) = S(n))) \equiv (\text{false} \lor (\exists b : S(b) = S(n))) \equiv (\exists b : S(b) = S(n))$

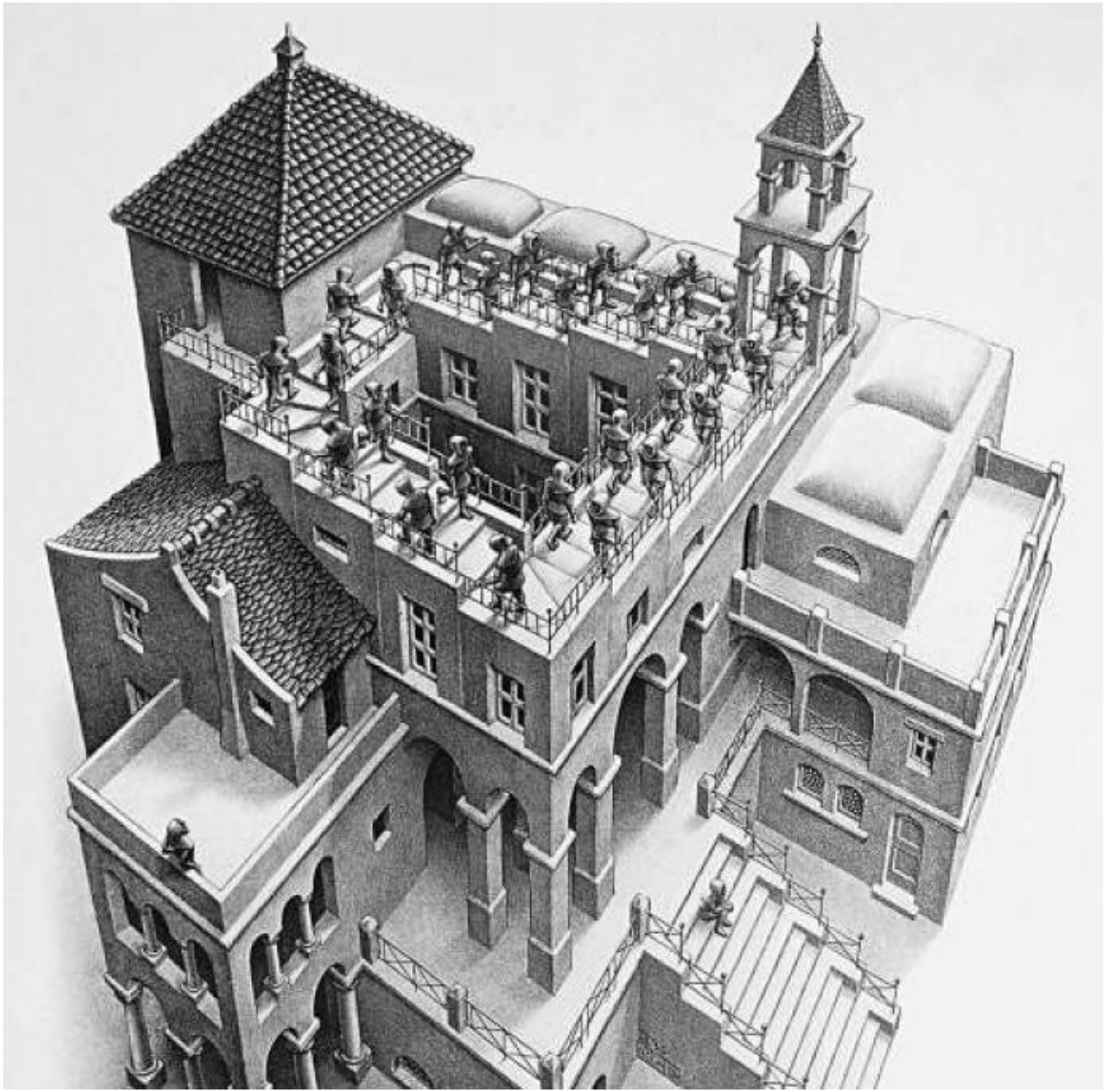
    $\equiv (S(n) = S(n)) \equiv (n = n) \equiv \text{true}$

Russell's Paradox (1901)

- Definition
  - A paradox is a statement or group of statements that leads to a contradiction
    or a situation which (if true) defies logic
    or reason, similar to circular reasoning.
- Examples
  - "This statement is false."
  - "The following statement is false.
    The previous statement is true."
- Russell's Paradox (1901)
  - the same paradox has been discovered a year before by Ernst Zermelo
  - let $R = \{x | x \notin x\}$, then $R \in R \leftrightarrow R \notin R$



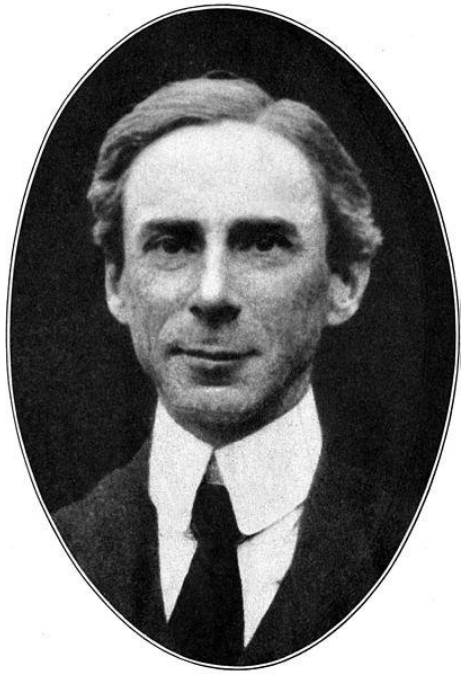
Bildergalerie
M. C. Escher
Lithographie, 1956



Treppauf, Treppab
M. C. Escher
Lithographie, 1960

Principia Mathematica PM (1913)

- Alfred North Whitehead (1861 – 1947)
  - English mathematician and philosopher (Trinity College)
  - co-author of "Principia Mathematica"
- Bertrand Russell (1872 – 1970)
  - British philosopher, logician, mathematician, historian, and social critic (Trinity College, Cambridge)
  - Pupil of Whitehead at Trinity College, Cambridge
  - shows that Frege's work on logic led to paradoxes (1901)
  - "The Principles of Mathematics" (1903)
  - "Principia Mathematica" (1910 – 1913)
    - three-volume work on the foundations of mathematics
    - an attempt to derive all mathematical truths from a well-defined set of axioms and inference rules in symbolic logic
    - inspired by Hilbert's 23 problems
    - tries to avoid paradoxes by building an elaborate system of types
- Open question: Is PM complete and consistent?



Propositional Calculus of PM is Complete (1920s)

- Emil Leon Post (1897 – 1954)
  - mathematician and logician (born in a Polish family, immigrated to New York when he was a child)
  - in his doctoral thesis, he proved that the propositional calculus of PM is complete
    - all tautologies are theorems, given the Principia axioms and the rules of substitution and modus ponens
  - he came very close to discovering the incompleteness of PM
  - invented truth tables independently of Wittgenstein and C.S. Peirce and put them to good mathematical use
  - Formulation 1: mathematical model of computation that was essentially equivalent to the Turing machine model (1936)
  - the unsolvability of his Post correspondence problem turned out to be exactly what was needed to obtain unsolvability results in the theory of formal languages

b	a	ca	abc
ca	ab	a	c

Hilbert's Program (1918 – 1922)

- David Hilbert (1862 – 1943)
  - German mathematician (Göttingen)
  - recognized as one of the most influential and universal mathematicians of the 19th and early 20th centuries
  - discovered and developed a broad range of fundamental ideas, e.g. Hilbert spaces, invariant theory and the axiomatization of geometry
  - «Naturerkennung der Logik» (1930)
- Hilbert's 23 Problems (1900)
  - No. 2: Prove that the axioms of arithmetic are consistent.
- Hilbert's Program
  - the main goal of Hilbert's program was to provide secure foundations for all mathematics, in particular this includes:
    - a formalization of all mathematics
    - completeness
    - consistency
    - decidability
    - conservation

Entscheidungsproblem (1928)

- David Hilbert, Wilhelm Ackermann (1896 – 1962): «Grundzüge der theoretischen Logik» (1928), Göttingen

  «Das Entscheidungsproblem ist gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt.»

  – the Entscheidungsproblem asks for an algorithm that takes as input a statement of a first-order logic
  – and answers "Yes" or "No" according to whether the statement is universally valid, i.e. valid in every structure satisfying the axioms
  – by the completeness theorem of first-order logic, a statement is universally valid if and only if it can be deduced from the axioms
  – the Entscheidungsproblem can also be viewed as asking for an algorithm to decide whether a given first-order logic statement is provable from the axioms using the rules of the first-order logic

First-order Logic is Consistent and Complete (1930)

- Kurt Gödel (1906 – 1978)
  - Austrian (Vienna) American (Princeton) logician, mathematician
  - «Über die Vollständigkeit des Logikkalküls.» (1929), Diss.

Die Vollständigkeit der Axiome des logischen Funktionenkalküls [1].

Von **Kurt Gödel** in Wien.

Whitehead und Russell haben bekanntlich die Logik und Mathematik so aufgebaut, daß sie gewisse evidente Sätze als Axiome an die Spitze stellten und aus diesen nach einigen genau formulierten Schlußprinzipien auf rein formalem Wege (d. h. ohne weiter von der Bedeutung der Symbole Gebrauch zu machen) die Sätze der Logik und Mathematik deduzierten. Bei einem solchen Vorgehen erhebt sich natürlich sofort die Frage, ob das an die Spitze gestellte System von Axiomen und Schlußprinzipien vollständig ist, d. h. wirklich dazu ausreicht, jeden logisch-mathematischen Satz zu deduzieren, oder ob vielleicht wahre (und nach anderen Prinzipien ev. auch beweisbare) Sätze denkbar sind, welche in dem betreffenden System nicht abgeleitet werden können. Für den Bereich der logischen Aussageformeln ist diese Frage in positivem Sinn entschieden, d. h. man hat gezeigt [2]), daß tatsächlich jede richtige Aussageformel aus den in den Principia Mathematica angegebenen Axiomen folgt. Hier soll dasselbe für einen weiteren Bereich von Formeln, nämlich für die des „engeren Funktionenkalküls“ [3]), geschehen, d. h. es soll gezeigt

Incompleteness of Number Theory (1931)

- Monatshefte für Mathematik 38, 1931

Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I[1].

Von **Kurt Gödel** in Wien.

1.

Die Entwicklung der Mathematik in der Richtung zu größerer Exaktheit hat bekanntlich dazu geführt, daß weite Gebiete von ihr formalisiert wurden, in der Art, daß das Beweisen nach einigen wenigen mechanischen Regeln vollzogen werden kann. Die umfassendsten derzeit aufgestellten formalen Systeme sind das System der Principia Mathematica (PM)[2] einerseits, das Zermelo-Fraenkelsche (von J. v. Neumann weiter ausgebildete) Axiomensystem der Mengenlehre[3] andererseits. Diese beiden Systeme sind so weit, daß alle heute in der Mathematik angewendeten Beweismethoden in ihnen formalisiert, d. h. auf einige wenige Axiome und Schlußregeln zurückgeführt sind. Es liegt daher die Vermutung nahe, daß diese Axiome und Schlußregeln dazu ausreichen, alle mathematischen Fragen, die sich in den betreffenden Systemen überhaupt formal ausdrücken lassen, auch zu entscheiden. Im folgenden wird gezeigt, daß dies nicht der Fall ist, sondern daß es in den beiden angeführten Systemen sogar relativ einfache Probleme aus der Theorie der gewöhnlichen ganzen Zahlen gibt[4], die sich aus den Axiomen nicht entscheiden lassen. Dieser Umstand liegt nicht etwa an der speziellen

Incompleteness of Number Theory (1931)

Jedes ω-widerspruchsfreie System ist selbstverständlich auch widerspruchsfrei. Es gilt aber, wie später gezeigt werden wird, nicht das Umgekehrte.

Das allgemeine Resultat über die Existenz unentscheidbarer Sätze lautet:

Satz VI: Zu jeder ω-widerspruchsfreien rekursiven Klasse $\varkappa$ von *Formeln* gibt es rekursive *Klassenzeichen* r, so daß weder v Gen r noch Neg (v Gen r) zu Flg ($\varkappa$) gehört (wobei v die *freie Variable* aus r ist).

- For any consistent recursive axiomatic system powerful enough to describe the arithmetic of the natural numbers (Peano arithmetic), there are true propositions about the naturals that cannot be proved from the axioms.
- This theorem is a major step towards a solution of the Entscheidungsproblem. However, there is still an open point concerning the universal validity of a statement, i.e. the validity in every structure satisfying the axioms.

Incompleteness of Number Theory : Proof Sketch

- Gödel's proof rested on the idea that statements about numbers could be coded as numbers, and constructing a self-referential statement G to defeat Hilbert's hopes.

- Formula G
  - let G be the formula in PM with the interpretation: "G is not a statement in PM"

- Is G a statement in PM?
  - if G is a statement in PM, then ¬G is also a statement in PM : contradiction!
  - if G is not a statement in PM, then PM is consistent but incomplete, because G is also a true formula

Gödel Numbering

- Gödel numbering (GN)
  - each formula in PM can be coded symbol by symbol into a unique natural number (codon)
  - each codon can be decoded back to its original formula

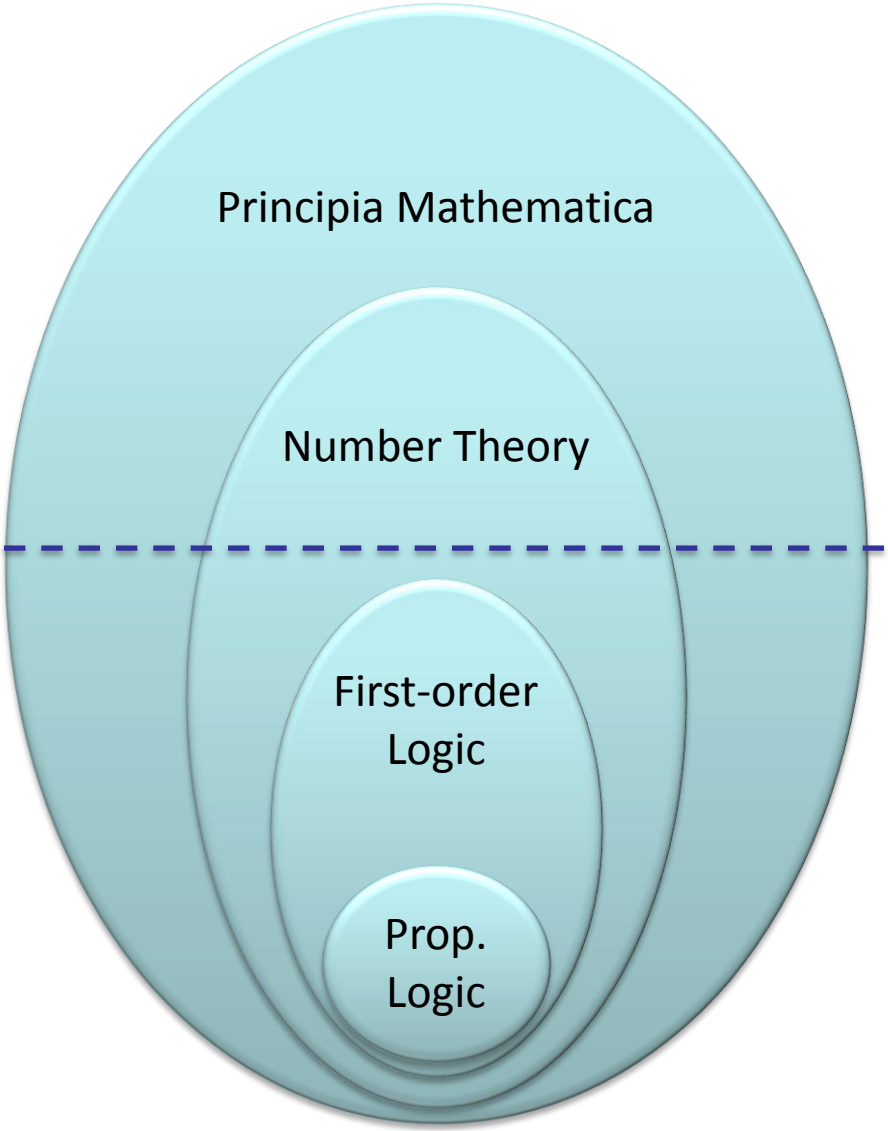
- Example
  - formula F: $\forall a : \neg S(a) = 0$
  - GN f: 626’262’636’223’123’362’262’323’111’666

Symbol	Codon
0	666
S	123
=	111
(	362
)	323
a	262
¬	223
∀	626
:	636
	…

Formula G

- ProofPair(a,a’)
  - a is the GN of the entire deduction of statement A’ with GN a’
- Subst(a’’,a,a’)
  - a’’ and a’ are the GNs of A’’ and A’, respectively
  - formula A’ is the result of a substitution of a free variable in formula A’’ by *a*
- SelfSubst(a’’,a’) = Subst(a’’,a’’,a’)
- formula U with GN *u*
  - ¬∃a,a’: ProofPair(a,a’) ⋀ SelfSubst(a’’,a’)
- formula G
  - ¬∃a,a’: ProofPair(a,a’) ⋀ SelfSubst(“all *a’’* substituted by number *u*” ,a’)
- Interpretations of G
  - There are no numbers *a* and *a’*, so that both form a PM proof pair, and that *a’* is the self-substitution of *u*.
  - There is no number *a*, which forms a PM proof pair together with the self-substitution of *u*.
  - The formula whose GN is the self-substitution of *u*, is not a statement in PM.
  - G is not a statement in PM.

PM: Completeness and Consistency



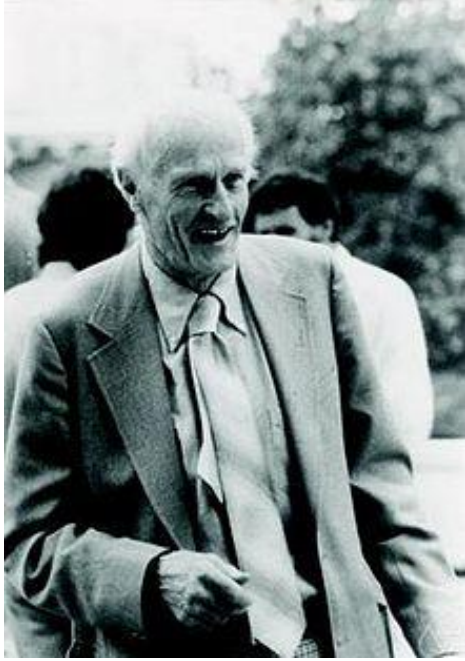
incomplete and consistent
or
complete and inconsistent

complete and consistent

Entscheidungsproblem is Undecidable (1936)

- Alonzo S. Church (1903 – 1995)
  - American mathematician and logician (Princeton)
  - made major contributions to mathematical logic and the foundations of theoretical computer science
  - best known for the lambda calculus
  - Church-Turing theorem: proving the undecidability of the Entscheidungsproblem
  - Church-[Post]-Turing thesis
- Lambda Calculus
  - equivalent in capabilities to Turing machines
  - influenced functional programming, e.g. LISP
  - C# and C++ provide lambda expressions

Recursion Theory (Computability Theory)



- Stephen Cole Kleene (1909 – 1994)
  - American mathematician who, distinguished students of Alonzo Church
  - best known as one of the founders of the recursion theory (a branch of mathematical logic), together with Turing, Post, and others
  - concepts named after him: Kleene hierarchy, Kleene algebra, the Kleene star (Kleene closure), Kleene's recursion theorem and the Kleene fixpoint theorem
  - invented regular expressions

- Recursion Theory
  - is a branch of mathematical logic, of computer science, and of the theory of computation
  - originated in the 1930s with the study of computable functions and Turing degrees

On computable numbers … (1936)

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means.

putable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

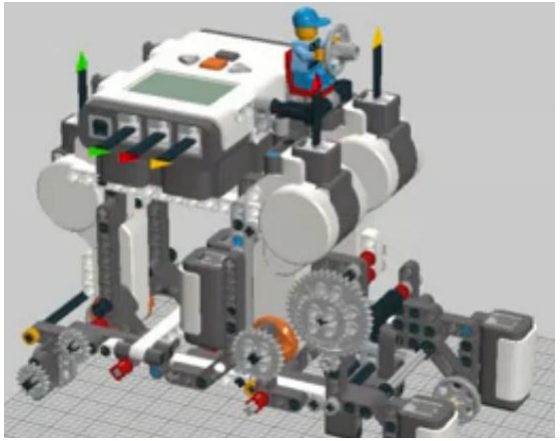
Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

Turing Machine (1936)

- A Turing Machine (TM) is a device with a finite amount of *read-only* "*hard*" memory (states), and an unbounded amount of read/write tape-memory. There is no separate input. Rather, the input is assumed to reside on the tape at the time when the TM starts running.

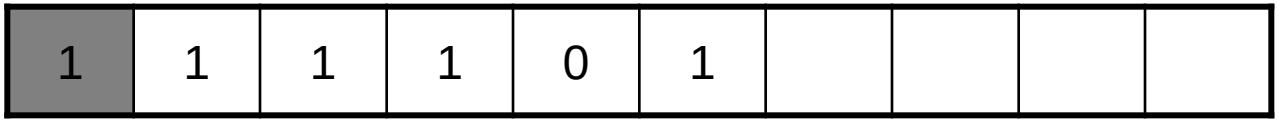
- Just as with Automata, TM's can either be input/output machines (compare with Finite State Transducers), or yes/no decision machines.



Turing Machine: Example Program

- Sample Rules:
  - If read 1, write 0, go right, repeat.
  - If read 0, write 1, HALT!
  - If read □, write 1, HALT! (the symbol □ stands for the blank cell)

- Let's see how these rules are carried out on an input with the reverse binary representation of 47:



Turing Machine: Formal Definition

- Definition: A Turing machine (TM) consists of a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$.
  - Q, Σ, and q_0, are the same as for an FA.
  - q_{acc} and q_{rej} are accept and reject states, respectively.
  - Γ is the tape alphabet which necessarily contains the blank symbol □, as well as the input alphabet Σ.
  - δ is as follows:

$$\delta : (Q - \{q_{acc}, q_{rej}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

  - Therefore given a non-halt state p, and a tape symbol x, $\delta(p,x) = (q,y,D)$ means that TM goes into state q, replaces x by y, and the tape head moves in direction D (left or right).

- A string x is accepted by M if after being put on the tape with the Turing machine head set to the left-most position, and letting M run, M eventually enters the accept state. In this case w is an element of $L(M)$ – the language accepted by M.

Church-Post-Turing Thesis (1936)

- First Goal of Turing's Machine: A "computer" which is as powerful as any real computer / programming language
  - As powerful as C, or "Java++"
  - Can execute all the same algorithms / code
  - Not as fast though (move the head left and right instead of RAM)
  - Historically: A model that can compute anything that a human can compute. Before invention of electronic computers the term "computer" actually referred to a *person* who's line of work is to calculate numerical quantities!
  - This is known as the [Church-[Post-]] Turing thesis, 1936.

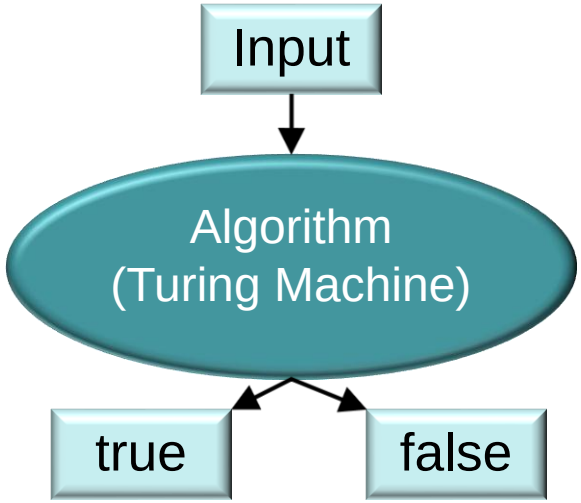
- Second Goal of Turing's Machine: And at the same time a model that is simple enough to actually prove interesting epistemological results.

Decidability

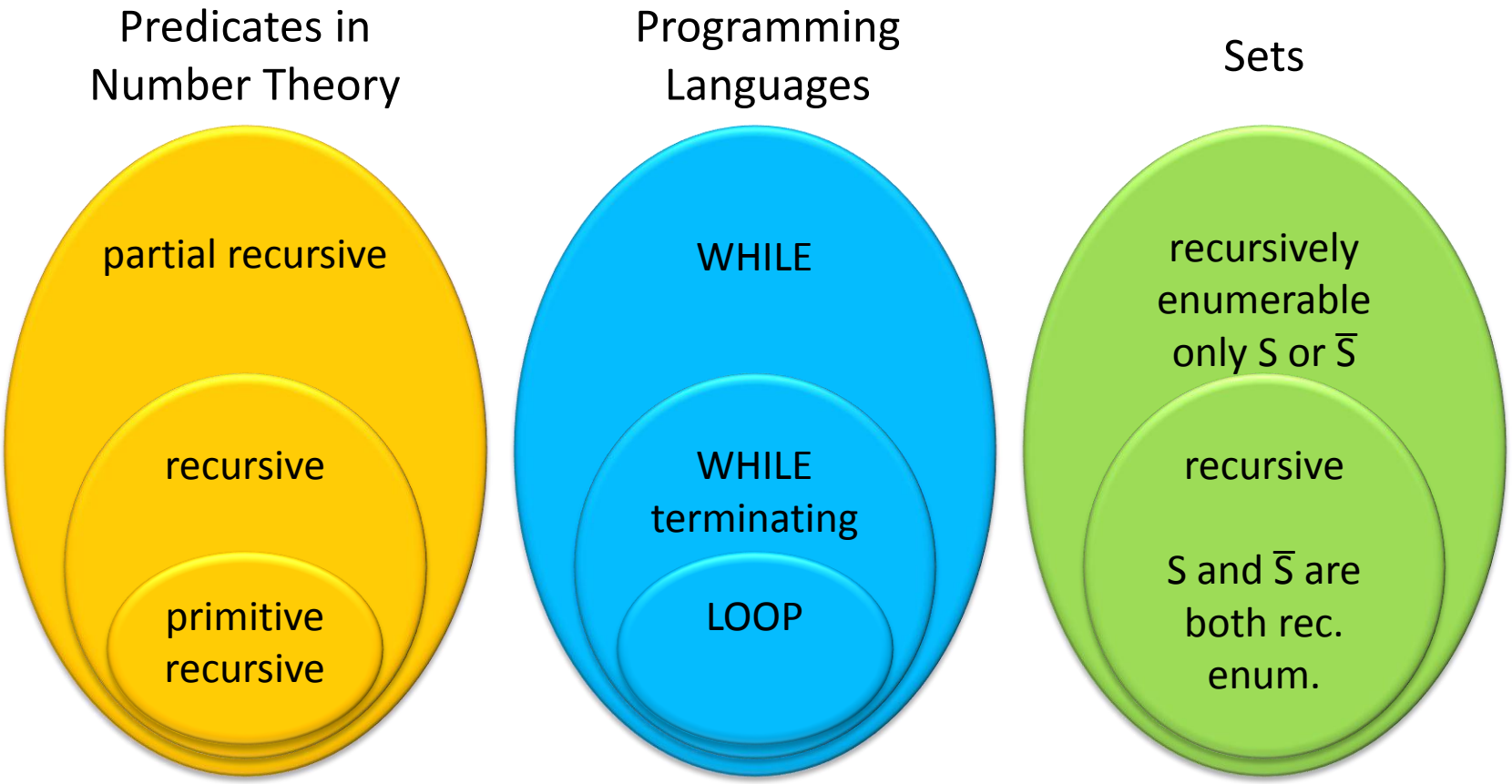
- A function is computable if there is an algorithm (according to the Church-Turing-Thesis a Turing machine is sufficient) that computes the function (in finite time).

- A subset T of a set M is called decidable (or recursive), if the function f: M → {true, false} with f(m) = true if m ∈ T, is computable.

- A more general class are the semi-decidable problems, for which the algorithm must only terminate in finite time in either the true or the false branch, but not the other.



Computability Theory (Recursion Theory)



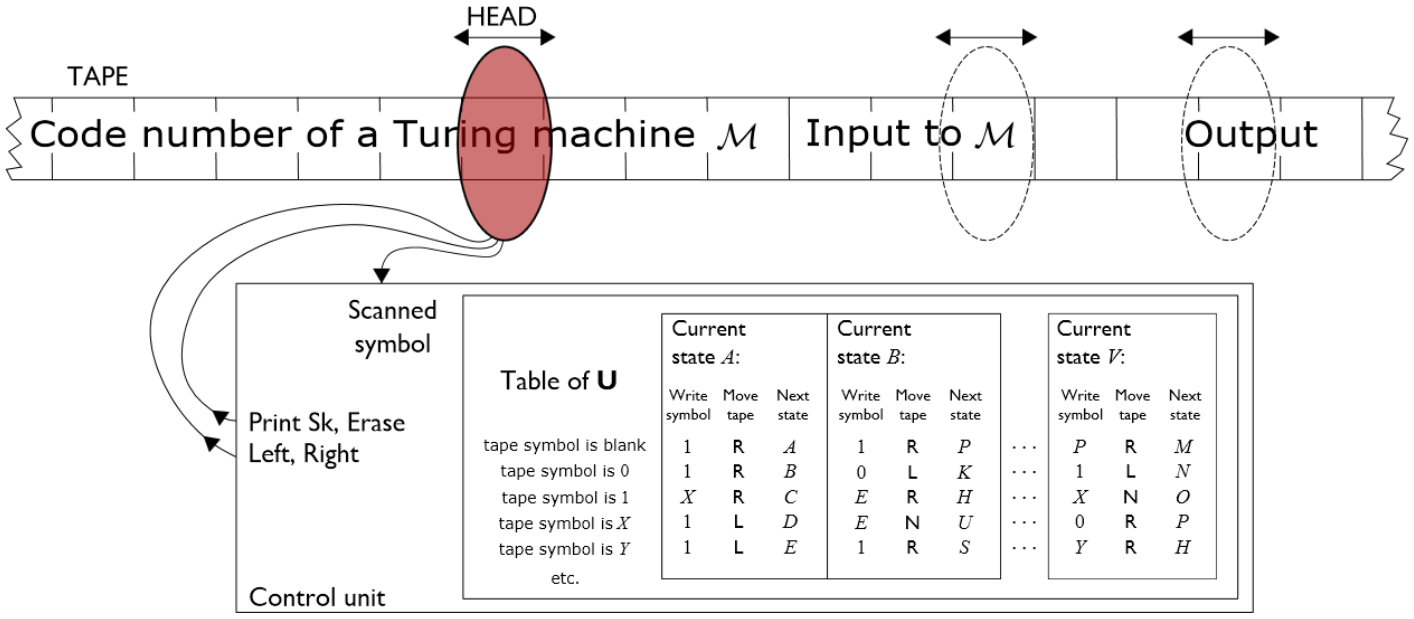
- primitive recursive: predictably finite running time
- LOOP: modern PL without if-then-else, while-loop, recursion
- WHILE: modern programs without endless loops

Confluence of Ideas in 1936

- **Alonzo Church:** "A note on the Entscheidungsproblem"
  - there is no solution for Hilbert's Entscheidungsproblem
- Alonzo Church: "An unsolvable problem of elementary number theory"
  - a function *f* is effectively calculable iff *f* can be defined in the λ-calculus
- **Stephen Kleene:** "λ-definability and recursivness"
  - a function *f* is recursive iff *f* can be defined in the λ-calculus
- **Emil Post:** "Finite combinatory processes"
  - introduction of a computational model, almost similar to Turing's machine
- **Alan Turing:** "On computational numbers, with an application to the Entscheidungsproblem"
  - there is no solution for Hilbert's Entscheidungsproblem
  - a function *f* is computable iff *f* can be computed by a Turing machine
  - contains in three carefully distinct categories clear justifications for his model
  - introduces the concept of an interpreter (universal Turing machine)
  - computation time and memory requirements can be easily defined (complexity classes)

Universal Turing Machine

- A universal Turing machine (UTM) is a Turing machine that can simulate an arbitrary Turing machine M on arbitrary input
- The UTM essentially achieves this by reading both the description of the machine to be simulated as well as the input thereof from its own tape.
- Is the origin of the stored program computer – used by John von Neumann (1946) for the "Electronic Computing Instrument" that now bears von Neumann's name: the von Neumann architecture.



Circle-free vs. Circular Turing Machines

- Definition
  - a TM is called circle-free if it only takes finite time to write down the next symbol of a computable number; otherwise it is said to be circular
- Theorem
  - Circle-Freeness is not decidable. (There is no TM which can decide if a given number is the description of a circle-free TM.)
- Proof by contradiction
  - assumption
    - there exists a TM D(n), which can decide if a given number n is the description of a circle-free TM
  - construction of a circle-free machine M (combination of D and UTM)
    - let b be the output of M
    - M enumerates all naturals and checks each number i with D(i) for being the description of a circle-free TM
    - before testing number n, a certain number R(n – 1) of machines have been found to be the description of circle-free TMs

Circle-Freeness is not decidable: Proof (Cont.)

- construction
  - in step *n* do:

    if D(*n*) sais circle-free, then

    R(*n*) = R(*n* – 1) + 1;

    compute the first R(*n*) output symbols of machine *n* using a UTM

    append the R(*n*)-th output symbol to output *b*

    else

    R(*n*) = R(*n* – 1)
- application of the diagonal process
  - let *k* be the description of our machine M
  - what happens in step *k*?

    if D(*k*) sais circle-free, then

    R(*k*) = R(*k* – 1) + 1

    compute the first R(*k*) output symbols of machine *k* (our machine M) using a UTM, but output *b* only contains R(*k*) – 1 valid output symbols, so the R(*k*)-th output symbol would never be found, and therefore M is circular →contradiction

    else D(*k*) sais circular

    but M is by construction circle-free → contradiction

Halting Problem

- The halting problem is a famous example of an undecidable (semi-decidable) problem. Essentially, you cannot write a computer program that decides whether another computer program ever terminates (or has an infinite loop) on some given input.

- In pseudo code, we would like to have:

```
procedure halting(program, input) {
     if program(input) terminates
     then return true
     else return false
}
```

Halting Problem Proof

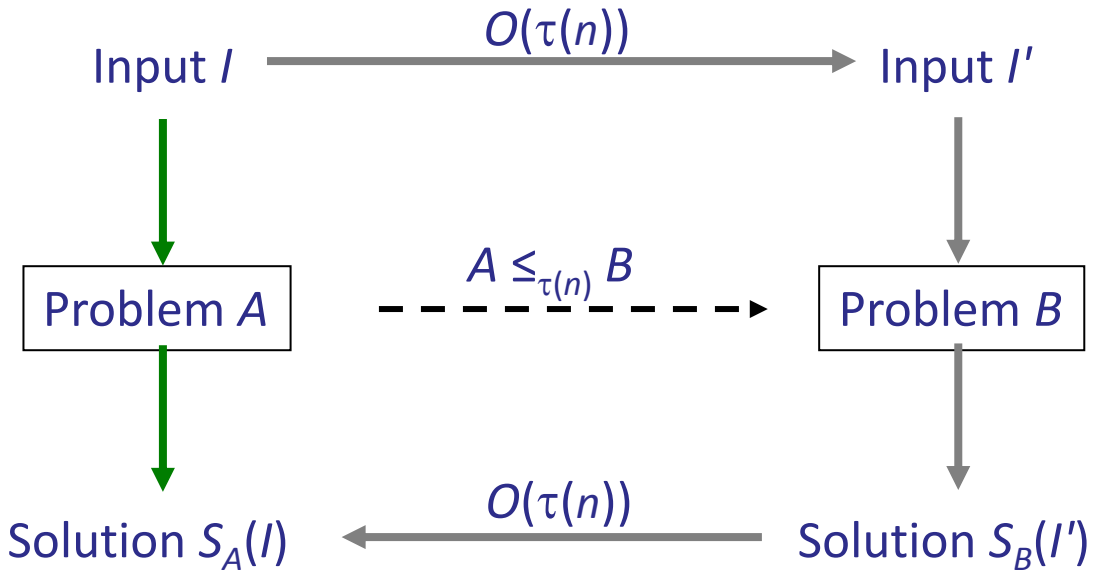
- Now we write a little wrapper around our halting procedure

```
procedure test(program) {
     if halting(program,program)
     then loop forever
     else return
}
```

- Now we simply run: `test(test)` ! Does it halt?!?
  - If halting(test,test) = true, test(test) should terminate, but it does not!
  - If halting(test,test) = false, test(test) should not terminate, but it does!

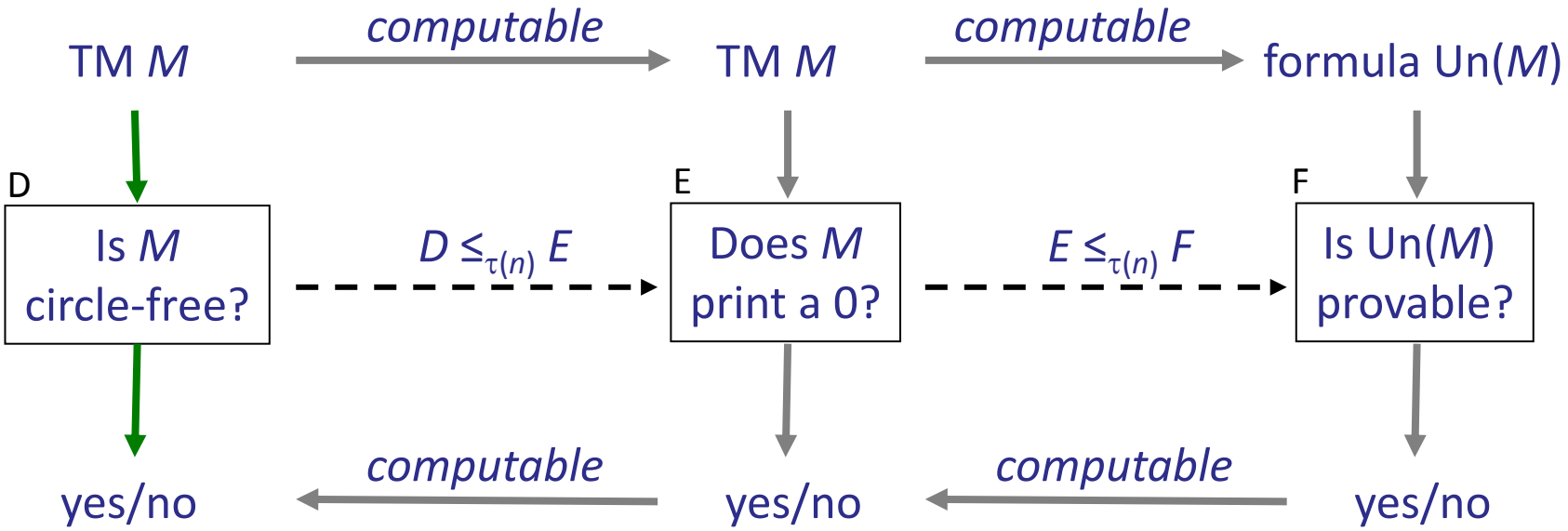
- Wicked! Our (generic) halting procedure is wrong, no matter what!
- We have a contradiction. No halting procedure can exist.

Problem Reduction



- a quick way of solving problem A is
  - to transform each instance of A into instances of the already solved problem B
  - solve these using our existing solution
  - use these solutions to obtain our final solution
- suppose we have a problem A that we've proven is hard to solve, and we have a similar problem B
  - we might suspect that B, too, is hard to solve
  - we argue by contradiction: suppose B is easy to solve
  - then, if we can show that *every* instance of A can be solved easily by transforming it into instances of B and solving those, we have a contradiction
  - this establishes that B is also hard to solve

Undecidability of the Entscheidungsproblem



- *"[…] if there is a machine E, then there is a general process for determining whether a given machine M prints 0 infinitely often."*
- *"Similarly there is a general process for determining whether M prints 1 infinitely often. By a combination of these processes we have a process for determining whether M prints an infinity of figures, i.e. we have a process for determining whether M is circle-free. There can therefore be no machine E."*
- *"Corresponding to each computing machine M we construct a formula Un(M) and we show that, if there is a general method for determining whether Un(M) is provable, then there is a general method for determining whether it ever prints 0, and this is impossible."*
- *"*Hence the Entscheidungsproblem cannot be solved."

Complexity Classes: P and NP

- P is the complexity class containing decision problems which can be solved by a Turing machine in time polynomial of the input size.

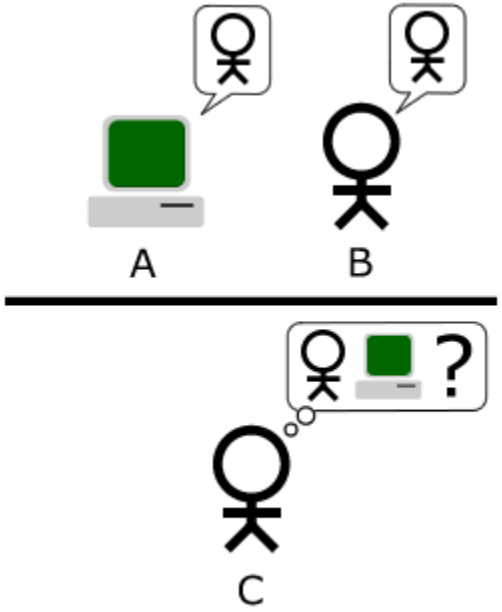
- NP is the class of decision problems solvable by a non-deterministic polynomial time Turing machine such that the machine answers "yes," if at least one computation path accepts, and answers “no,” if all computation paths reject.
  - Quite similarly to the nondeterministic finite automaton from Chapter 1.
  - Informally, there is a Turing machine which can check the correctness of an answer in polynomial time.
  - E.g. one can check in polynomial time whether a traveling salesperson path connects *n* cities with less than a total distance *d*.
  - Or one can check in polynomial time whether two big numbers are the factors of an even bigger number (with *n* digits).

P vs. NP

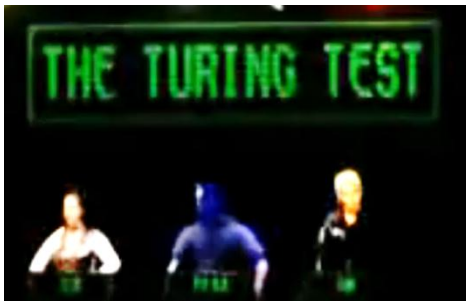
- An important notion in this context is the large set of NP-complete decision problems, which is a subset of NP and might be informally described as the "hardest" problems in NP. If there is a polynomial-time algorithm for even one of them, then there is a polynomial-time algorithm for all the problems in NP.
  - E.g. Given a set of *n* integers, is there a non-empty subset which sums up to 0? This problem was shown to be NP-complete.
  - Also the traveling salesperson problem is NP-complete, or Tetris, or Minesweeper.

- One of the big questions in Math and CS: Is P = NP?
  - Or are there problems which cannot be solved in polynomial time.
  - Big practical impact (e.g. in Cryptography).
  - One of the seven $1M problems by the Clay Mathematics Institute of Cambridge, Massachusetts.

Turing Test (1950)

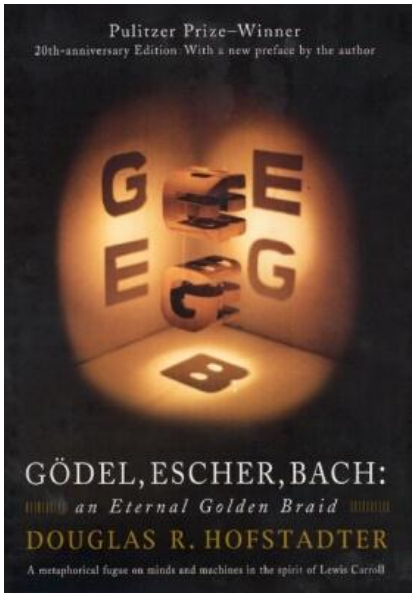


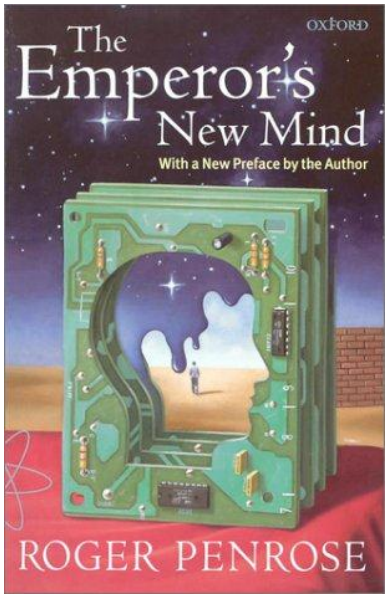
- Definition: The Turing test is a test of a machine's ability to exhibit intelligent behavior, equivalent to or indistinguishable from, that of an actual human.
- original illustrative example
  - a human judge engages in a natural language conversation with a human and a machine designed to generate performance indistinguishable from that of a human being
  - all participants are separated from one another
  - if the judge cannot reliably tell the machine from the human, the machine is said to have passed the test
- the test does not check the ability to give correct answers
- it checks how closely the answers resemble typical human answers
- the conversation is limited to a text-only channel such as a computer keyboard and screen



Bedtime Reading

If you're leaning towards "human = machine"





If you're leaning towards "human ⊃ machine"