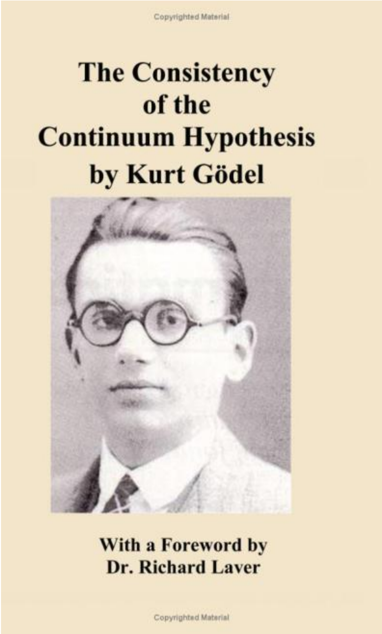
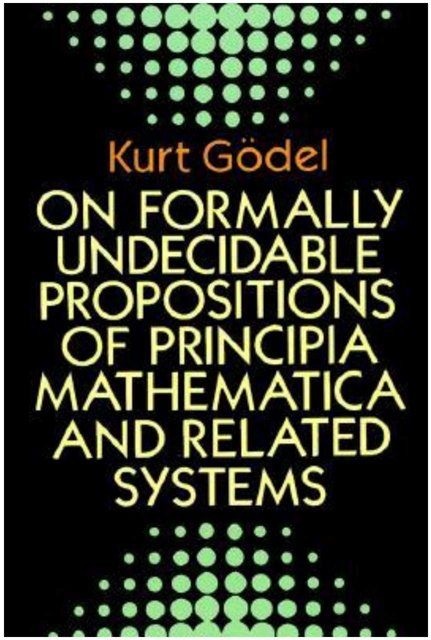
Historical Perspectives

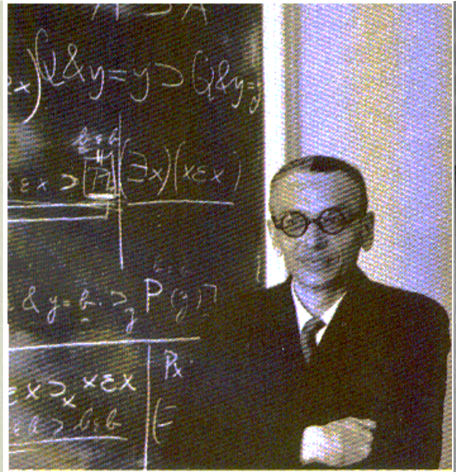
Kurt Gödel (1906-1978)

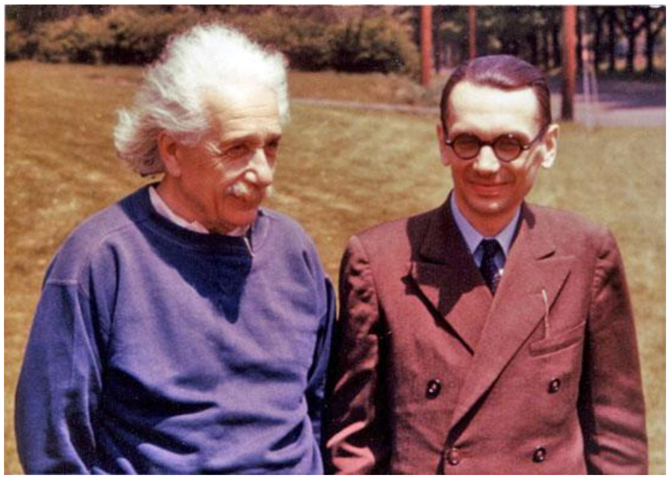
- Logician, mathematician, and philosopher
- Proved completeness of predicate logic and Gödel's incompleteness theorem
- Proved consistency of axiom of choice and the continuum hypothesis
- Invented "Gödel numbering" and "Gödel fuzzy logic"
- Developed "Gödel metric" and paradoxical relativity solutions: "Gödel spacetime / universe"
- Made enormous impact on logic, mathematics, and science



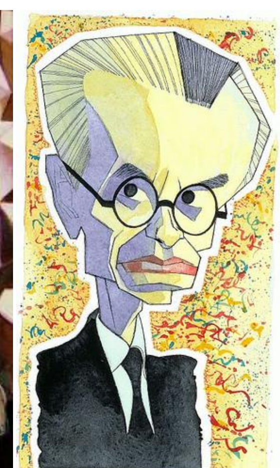


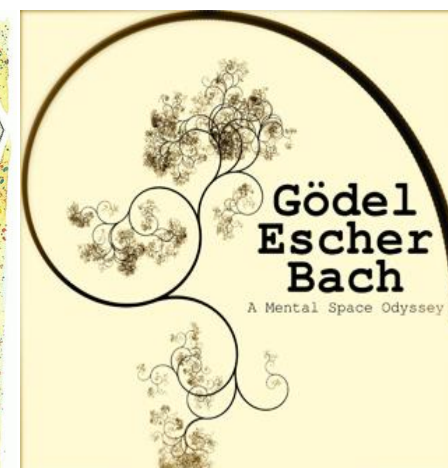
Library of Congress

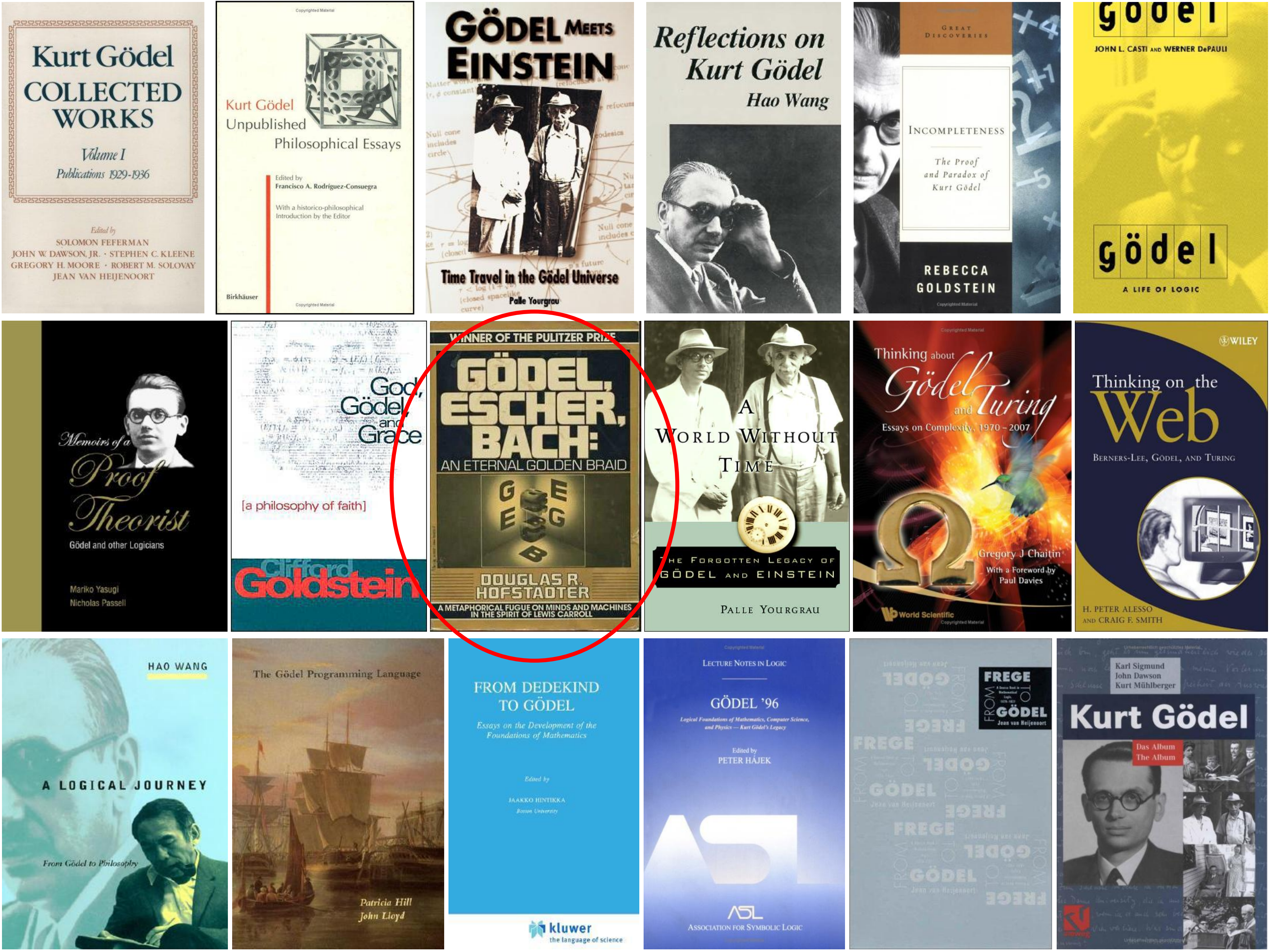




Kurt Gödel
1906 - 1978



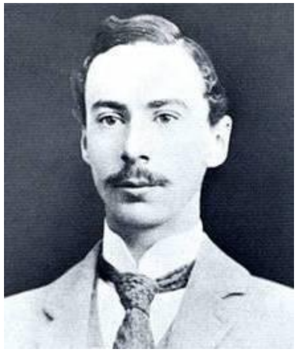



Kurt Gödel
COLLECTED WORKS
Volume I
Publications 1929-1936
Edited by
SOLOMON FEFERMAN
JOHN W. DAWSON, JR. · STEPHEN C. KLEENE
GREGORY H. MOORE · ROBERT M. SOLOVAY
JEAN VAN HEIJENOORT
Kurt Gödel
Unpublished
Philosophical Essays
Edited by
Francisco A. Rodríguez-Consuegra
With a historico-philosophical Introduction by the Editor
Birkhäuser
GÖDEL MEETS EINSTEIN
Time Travel in the Gödel Universe
Palle Yourgrau
Reflections on Kurt Gödel
Hao Wang
GREAT DISCOVERIES
INCOMPLETENESS
The Proof and Paradox of Kurt Gödel
REBECCA GOLDSTEIN
JOHN L. CASTI AND WERNER DePAULI
gödel
A LIFE OF LOGIC
Memoirs of a Proof Theorist
Gödel and other Logicians
Mariko Yasugi
Nicholas Passell
God, Gödel and Grace
[a philosophy of faith]
Clifford Goldstein
WINNER OF THE PULITZER PRIZE
GÖDEL, ESCHER, BACH:
AN ETERNAL GOLDEN BRAID
DOUGLAS R. HOFSTADTER
A METAPHORICAL FUGUE ON MINDS AND MACHINES IN THE SPIRIT OF LEWIS CARROLL
A WORLD WITHOUT TIME
THE FORGOTTEN LEGACY OF GÖDEL AND EINSTEIN
PALLE YOURGRAU
Thinking about Gödel and Turing
Essays on Complexity, 1970 – 2007
Gregory J Chaitin
With a Foreword by Paul Davies
World Scientific
WILEY
Thinking on the Web
BERNERS-LEE, GÖDEL, AND TURING
H. PETER ALESSO AND CRAIG F. SMITH
HAO WANG
A LOGICAL JOURNEY
From Gödel to Philosophy
The Gödel Programming Language
Patricia Hill
John Lloyd
FROM DEDEKIND TO GÖDEL
Essays on the Development of the Foundations of Mathematics
Edited by
JAAKKO HINTIKKA
Boston University
kluwer
the language of science
LECTURE NOTES IN LOGIC
GÖDEL '96
Logical Foundations of Mathematics, Computer Science, and Physics — Kurt Gödel's Legacy
Edited by
PETER HÁJEK
ASL
ASSOCIATION FOR SYMBOLIC LOGIC
FROM FREGE TO GÖDEL
Jean van Heijenoort
Karl Sigmund
John Dawson
Kurt Mühlberger
Kurt Gödel
Das Album
The Album


GÖDEL'S PROOF
ERNEST NAGEL AND JAMES R. NEWMAN
Palle Yourgrau
Gödel, Einstein und die Folgen
Vermächtnis einer ungewöhnlichen Freundschaft
C.H. Beck
GÖDEL'S INCOMPLETENESS THEOREMS
RAYMOND M. SMULLYAN
Rebecca Goldstein
KURT GÖDEL
Jahrhundertmathematiker und großer Entdecker
PIPER
GODEL'S THEOREM SIMPLIFIED
Harry J. Gensler
UNIVERSITY PRESS OF AMERICA
Gödel, Putnam, and Functionalism
JEFF BUECHNER
forever UNDECIDED
A PUZZLE GUIDE TO GÖDEL
Raymond Smullyan
OXFORD
GÖDEL'S THEOREM
AN INCOMPLETE GUIDE TO ITS USE AND ABUSE
TORKEL FRANZÉN
Logical Dilemmas
THE LIFE AND WORK OF KURT GÖDEL
JOHN W. DAWSON, JR.
WADSWORTH PHILOSOPHERS SERIES
ON GÖDEL
JAAKKO HINTIKKA
CAMBRIDGE INTRODUCTIONS TO PHILOSOPHY
An Introduction to Gödel's Theorems
PETER SMITH
CAMBRIDGE
112 MERCER STREET
EINSTEIN, RUSSELL, GÖDEL, PAULI, and the End of Innocence in Science
BURTON FELDMAN
AUTHOR OF THE NOBEL PRIZE
Kurt Gödel: Leben und Werk
John W. Dawson Jr.
SpringerWienNewYork
KURT GÖDEL UND DIE MATHEMATISCHE LOGIK
EUROPOLIS 5
Werner DePauli-Schimanovich (Herausgeber: Franz Pichler)
Schriftenreihe GESCHICHTE DER NATURWISSENSCHAFTEN UND DER TECHNIK. Johannes Kepler Universität Linz. Universitätsverlag Rudolf Trauner
Band 122
Verena Huber-Dyson
Gödel's Theorems; a Workbook on Formalization
TEUBNER-TEXTE zur Mathematik
Frege and Godel
2 Fundamental Texts in Mathematical Logic
Jean van Heijenoort, Ed.
Gödel's Theorem in focus
Edited by S.G.SHANKER
Gödel: une révolution en mathématiques
Essai sur les conséquences scientifiques et philosophiques des théorèmes gödeliens
André Delessert
PRESSES POLYTECHNIQUES ET UNIVERSITAIRES ROMANDES

Gödel's Incompleteness Theorem

Frege & Russell:
- Mechanically verifying proofs
- Automatic theorem proving


A set of axioms is:
- Sound: iff only true statements can be proved
- Complete: iff any statement or its negation can be proved
- Consistent: iff no statement and its negation can be proved

Hilbert's program: find an axiom set for all of mathematics
i.e., find a axiom set that is consistent and complete

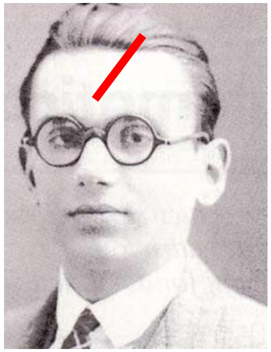
Gödel: any consistent axiomatic system is incomplete!
(as long as it subsume elementary arithmetic)

i.e., any consistent axiomatic system must contain true but unprovable statements

Mathematical surprise: truth and provability are not the same!

Gödel's Incompleteness Theorem

That some axiomatic systems are incomplete is not surprising, since an important axiom may be missing (e.g., Euclidean geometry without the parallel postulate)



However, that every consistent axiomatic system must be incomplete was an unexpected shock to mathematics! This undermined not only a particular system (e.g., logic), but axiomatic reasoning and human thinking itself!

Truth = Provability

Justice ≠ Legality

Gödel's Incompleteness Theorem

Gödel: consistency or completeness - pick one!

Which is more important?

Incomplete: not all true statements can be proved.
But if useful theorems arise, the system is still useful.

Inconsistent: some false statement can be proved.
This can be catastrophic to the theory:

E.g., supposed in an axiomatic system we proved that "1=2".
Then we can use this to prove that, e.g., all things are equal!
Consider the set: {Bush, Pope}

| {Bush, Pope} | = 2

$\Rightarrow$ | {Bush, Pope} | = 1 (since 1=2)

$\Rightarrow$ Bush = Pope QED

$\Rightarrow$ All things become true: system is "complete" but useless!

Gödel's Incompleteness Theorem

Moral: it is better to be consistent than complete,
If you can not be both.

"It is better to be feared than loved, if you cannot be both."
- Niccolo Machiavelli (1469-1527), "The Prince"

"You can have it good, cheap, or fast – pick any two."
- Popular business adage

Gödel's Incompleteness Theorem

Thm: any consistent axiomatic system is incomplete!

Proof idea:

- Every formula is encoded uniquely as an integer
- Extend "Gödel numbering" to formula sequences (proofs)

Dovetailing!

- Construct a "proof checking" formula P(n,m) such that P(n,m) iff n encodes a proof of the formula encoded by m

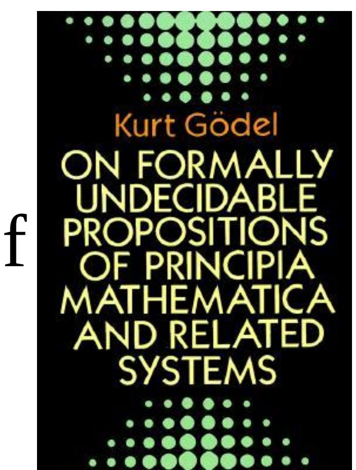
Algorithm!

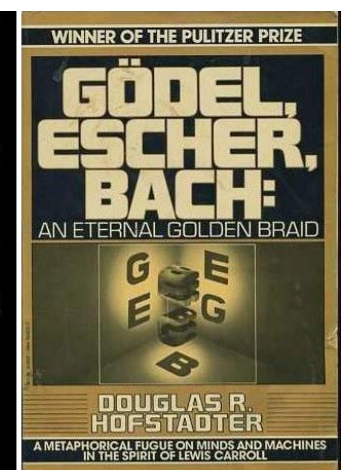
- Construct a self-referential formula that asserts its own non-provability: "I am not provable"

Diagonalization!

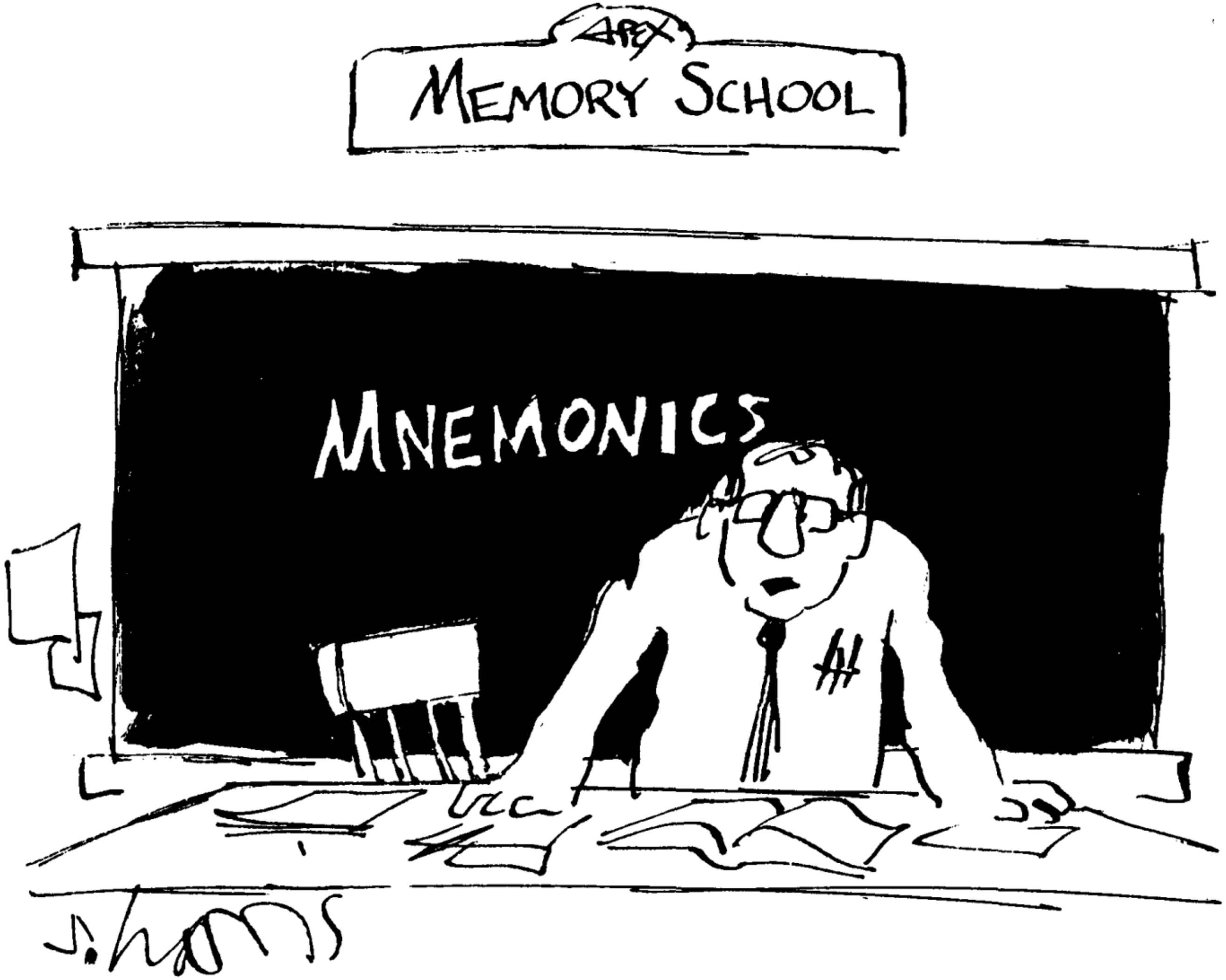
- Show this formula is neither provable nor disprovable

George Boolos (1989) gave shorter proof based on formalizing Berry's paradox





The set of true statements is not R.E.!



"YOU SIMPLY ASSOCIATE EACH NUMBER WITH A WORD, SUCH AS 'TABLE' AND 3,476,029."

Gödel's Incompleteness Theorem

Systems known to be complete and consistent:

- Propositional logic (Boolean algebra)
- Predicate calculus (first-order logic) [Gödel, 1930]
- Sentential calculus [Bernays,1918; Post, 1921]
- Presburger arithmetic (also decidable)

Systems known to be either inconsistent or incomplete:

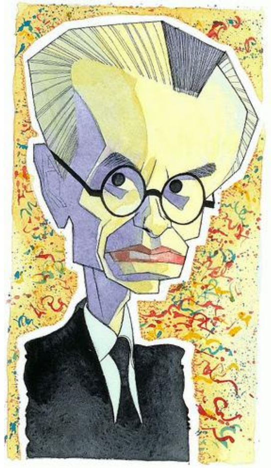
- Peano arithmetic
- Primitive recursive arithmetic
- Zermelo–Frankel set theory
- Second-order logic

Q: Is our mathematics both consistent and complete?
A: No [Gödel, 1931]

Q: Is our mathematics at least consistent?
A: We don't know! But we sure hope so.

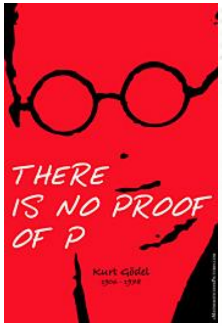
Gödel's "Ontological Proof" that God exists!

Formalized Saint Anselm's ontological argument using modal logic:



$$
\begin{array}{ll}
\text{Ax. 1.} & P(\varphi) \wedge \Box\, \forall x[\varphi(x) \to \psi(x)] \to P(\psi) \\
\text{Ax. 2.} & P(\neg\varphi) \leftrightarrow \neg P(\varphi) \\
\text{Th. 1.} & P(\varphi) \to \Diamond\, \exists x\, [\varphi(x)] \\
\text{Df. 1.} & G(x) \iff \forall\varphi[P(\varphi) \to \varphi(x)] \\
\text{Ax. 3.} & P(G) \\
\text{Th. 2.} & \Diamond\, \exists x\, G(x) \\
\text{Df. 2.} & \varphi \text{ ess } x \iff \varphi(x) \wedge \forall\psi\{\psi(x) \to \Box\, \forall x[\varphi(x) \to \psi(x)]\} \\
\text{Ax. 4.} & P(\varphi) \to \Box\, P(\varphi) \\
\text{Th. 3.} & G(x) \to G \text{ ess } x \\
\text{Df. 3.} & E(x) \iff \forall\varphi[\varphi \text{ ess } x \to \Box\, \exists x\, \varphi(x)] \\
\text{Ax. 5.} & P(E) \\
\text{Th. 4.} & \Box\, \exists x\, G(x)
\end{array}
$$

For more details, see:
http://en.wikipedia.org/wiki/Godel_ontological_proof



KGS: Welcome - Mozilla Firefox

http://kgs.logic.at/

The Kurt Gödel Society

- **Welcome**
- News and Activities
  - Lecture Series
  - Conferences
  - Publications
  - Other activities
- Organization
  - Our presidents
- Useful links
- Membership
  - Application Form
- Grants
  - Gödel Fellowship
- Kurt Gödel
- Contact

Welcome

The Kurt Gödel Society was founded in 1987 and is chartered in Vienna. It is an international organization for the promotion of research in the areas of Logic, Philosophy, History of Mathematics, above all in connection with the biography of Kurt Gödel, and in other areas to which Gödel made contributions, especially mathematics, physics, theology, philosophy and Leibniz studies.

Top News

09-06-08 12:00

Fourth Vienna Tbilisi Summer School in Logic and Language

For the third time students and teachers meet in Tbilisi, Georgia, for a summer school. Please see the conference page http://www.logic.at/tbilisi08/ fo... [more...]

05-12-07 23:22

Collegium Logicum Lecture Series

6 December 2007, 16:00 Peter Schuster (LMU München) - Finite methods in commutative algebra [more...]

15-11-07 12:27

Workshop Two and beyond

The KGS is organizing a workshop on truth-functional logics. [more...]

Horizons of Truth
Gödel *Centenary* 2006

Horizons of Truth
Logics, Foundations of Mathematics, and the Quest for Understanding the Nature of Knowledge

Gödel Centenary 2006
An International Symposium Celebrating the 100th Birthday of Kurt Gödel

27.-29. April 2006
Festsaal of the University of Vienna

Organized by:
The Kurt Gödel Society
Co-organized by:
University of Vienna, Institute for Experimental Physics, Kurt Gödel Research Center, Institute Vienna Circle, Vienna University of Technology,
Sponsored by:
The John Templeton Foundation
The Federation of Austrian Industry
The Federal Ministery of Infrastructure
The Federal Ministery of Education, Science and Culture
The Government of the Ciry of Vienna
The Austrian Mathematical Society
Microsoft Corporation

Print this page

Horizons of Truth

Logics, Foundations of Mathematics, and the Quest for Understanding the Nature of Knowledge

Gödel Centenary 2006

An International Symposium Celebrating the 100th Birthday of Kurt Gödel

27.-29. April 2006
***Festsaal* of the University of Vienna**

Organized by the Kurt Gödel Society with the support of the John Templeton Foundation. Co-organized by the University of Vienna, the Institute for Experimental Physics, the Kurt Gödel Research Center, the Institute Vienna Circle, and the Vienna University of Technology.

The purpose of the Symposium is to commemorate the life, work, and foundational views of Kurt Gödel, perhaps the greatest logician of the twentieth century. In the spirit of Gödel's work, the Symposium will also explore current research advances and ideas for future possibilities in the fields of the foundations of mathematics and logic. The symposium intends to put Gödel's ideas and works into a more general context in the light of current understanding and perception. The symposium will also present various implications of his work for other areas of intellectual endeavor such as artificial intelligence, cosmology, philosophy, and theology.

The Symposium will take place 27-29 April in the Celebration Hall of the University of Vienna, famous for its architectural beauty and the murals of Klimt. More than 20 lectures by eminent scientists in the fields of logics, mathematics, philosophy, physics, and theology will provide new insights into the life and work of Kurt Gödel and their implications for future generations.

Contributions

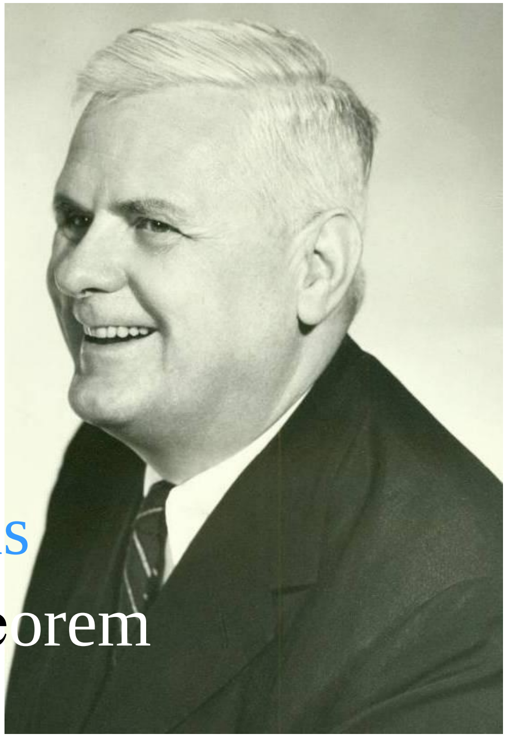
The program will contain

Talks by the invited speakers

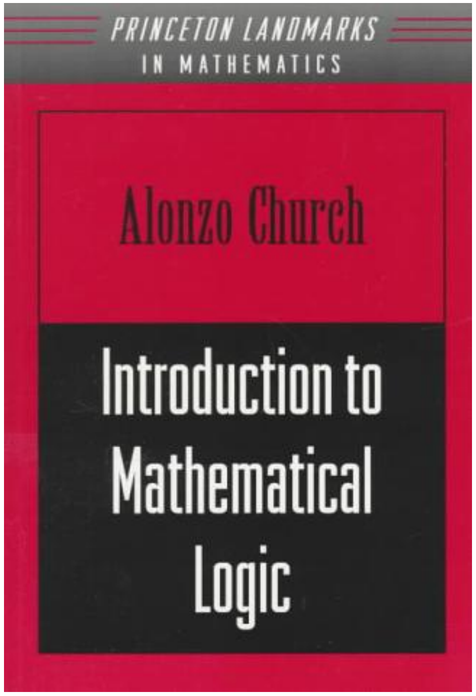
John D. Barrow, Cambridge University, UK

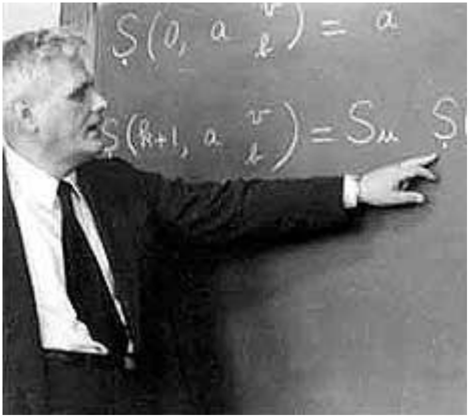
Historical Perspectives

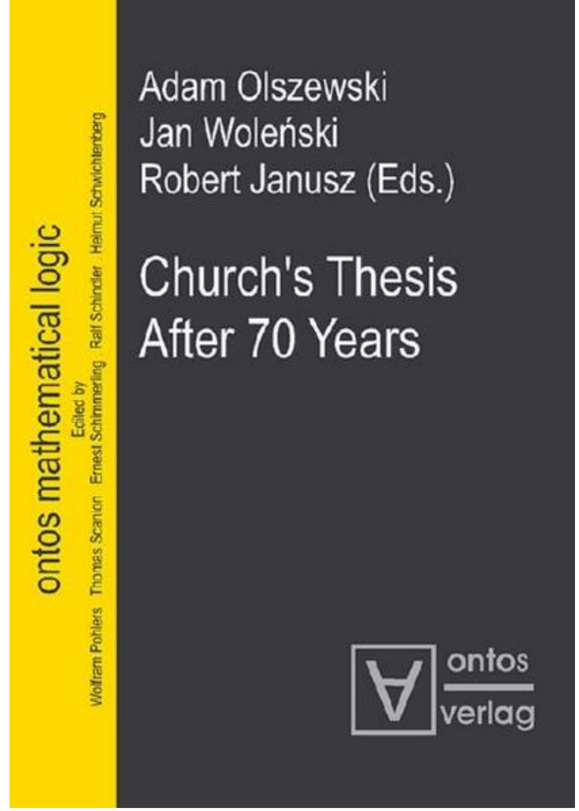
Alonzo Church (1903-1995)

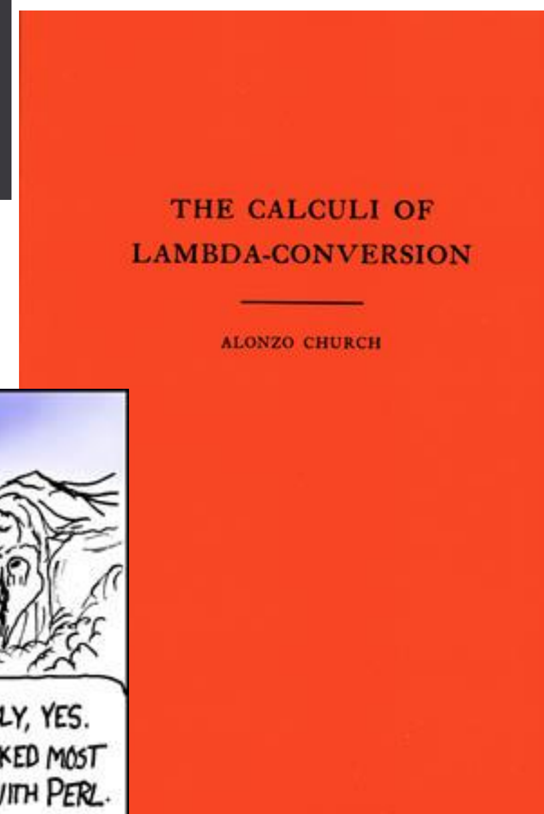


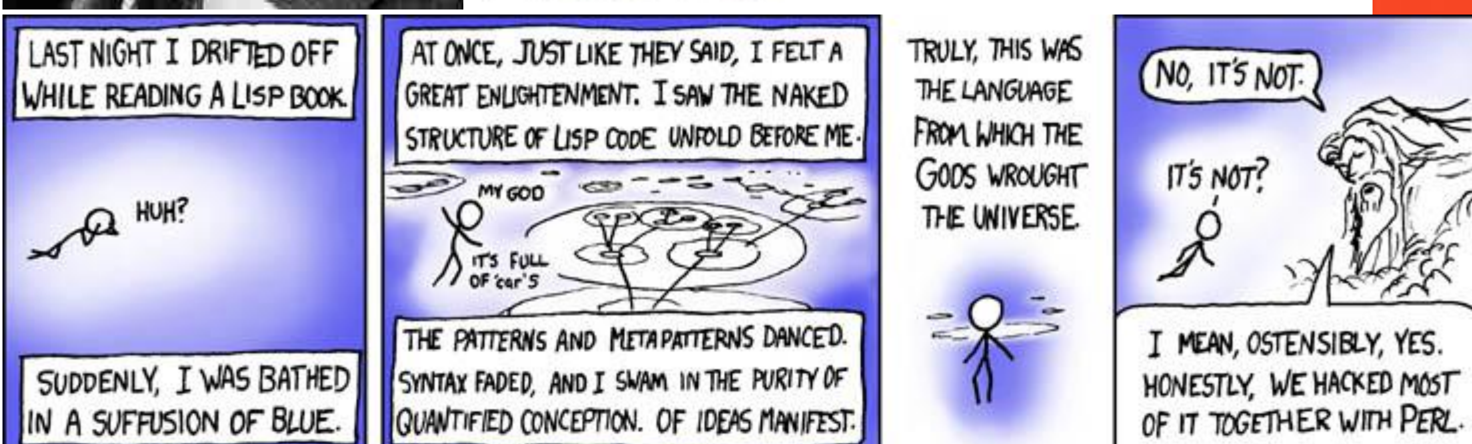
- Founder of theoretical computer science
- Made major contributions to logic
- Invented Lambda-calculus, Church-Turing Thesis
- Originated Church-Frege Ontology, Church's theorem Church encoding, Church-Kleene ordinal,
- Inspired LISP and functional programming
- Was Turing's Ph.D. advisor! Other students: Davis, Kleene, Rabin, Rogers, Scott, Smullyan
- Founded / edited Journal of Symbolic Logic
- Taught at UCLA until 1990; published "A Theory of the Meaning of Names" in 1995, at age 92!

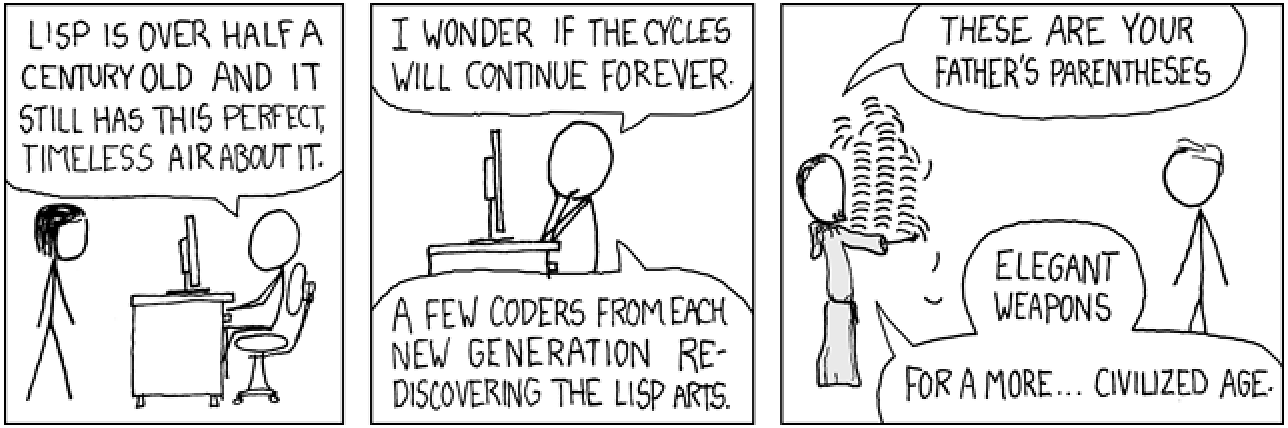





ontos mathematical logic
Edited by
Wolfram Pohlers · Thomas Scanlon · Ernest Schimmerling · Ralf Schindler · Helmut Schwichtenberg
Adam Olszewski
Jan Woleński
Robert Janusz (Eds.)
Church's Thesis
After 70 Years
ontos
verlag


THE CALCULI OF
LAMBDA-CONVERSION
ALONZO CHURCH


LAST NIGHT I DRIFTED OFF WHILE READING A LISP BOOK.
HUH?
SUDDENLY, I WAS BATHED IN A SUFFUSION OF BLUE.
AT ONCE, JUST LIKE THEY SAID, I FELT A GREAT ENLIGHTENMENT. I SAW THE NAKED STRUCTURE OF LISP CODE UNFOLD BEFORE ME.
MY GOD
IT'S FULL OF car'S
THE PATTERNS AND METAPATTERNS DANCED. SYNTAX FADED, AND I SWAM IN THE PURITY OF QUANTIFIED CONCEPTION. OF IDEAS MANIFEST.
TRULY, THIS WAS THE LANGUAGE FROM WHICH THE GODS WROUGHT THE UNIVERSE.
NO, IT'S NOT.
IT'S NOT?
I MEAN, OSTENSIBLY, YES. HONESTLY, WE HACKED MOST OF IT TOGETHER WITH PERL.


LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

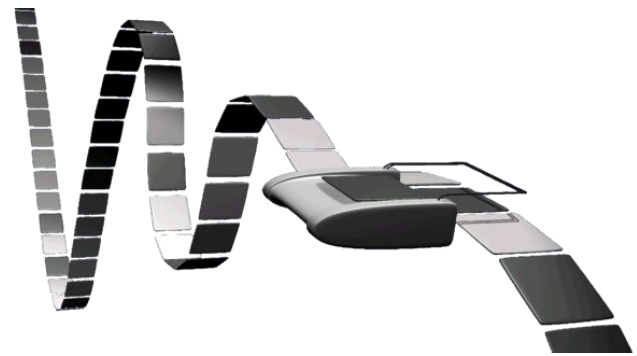

A GOD'S LAMENT
SOME SAID THE WORLD SHOULD BE IN PERL;
SOME SAID IN LISP.
NOW, HAVING GIVEN BOTH A WHIRL,
I HELD WITH THOSE WHO FAVORED PERL.
BUT I FEAR WE PASSED TO MEN
A DISAPPOINTING FOUNDING MYTH,
AND SHOULD WE WRITE IT ALL AGAIN,
I'D END IT WITH
A CLOSE-PAREN.


AS YOU KNOW, WE'RE IN THE EIGHTH YEAR OF OUR NORTHERN WARS AGAINST THE HASKELLERS. THERE ARE RUMORS THAT MORE OF OUR TROOPS ARE DEFECTING TO THE OTHER SIDE EVERY DAY...
REPUBLIC OF HASKELL
LAND OF LISP
DON'T BE TEMPTED TO BREAK RANKS!
I ASSURE YOU

Historical Perspectives

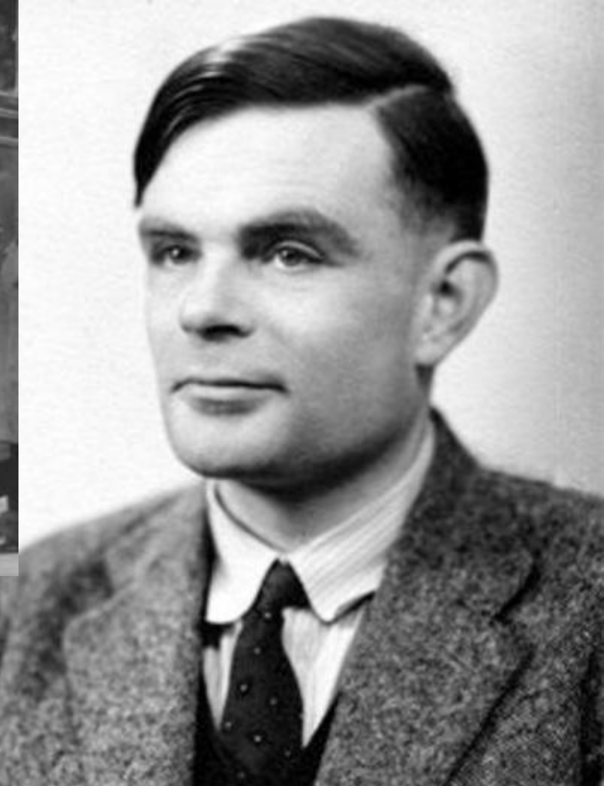
Alan Turing (1912-1954)

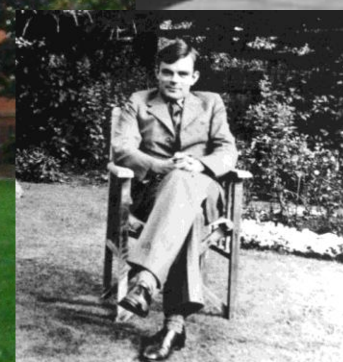
- Mathematician, logician, cryptanalyst, and founder of computer science
- First to formally define computation / algorithm
- Invented the Turing machine model
  - theoretical basis of all modern computers

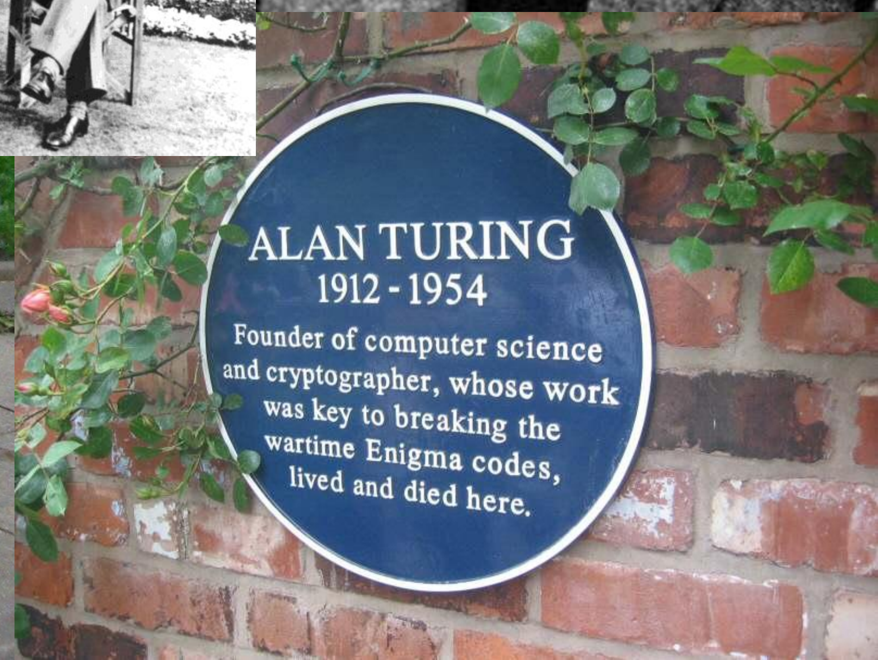


- Investigated computational "universality"
- Introduced "definable" real numbers
- Proved undecidability of halting problem
- Originated oracles and the "Turing test"
- Pioneered artificial intelligence
- Anticipated neural networks
- Designed the Manchester Mark 1 (1948)
- Helped break the German Enigma cypher
- Turing Award was created in his honor


THE
ALAN TURING MEMO
ALAN MATHISON TURING





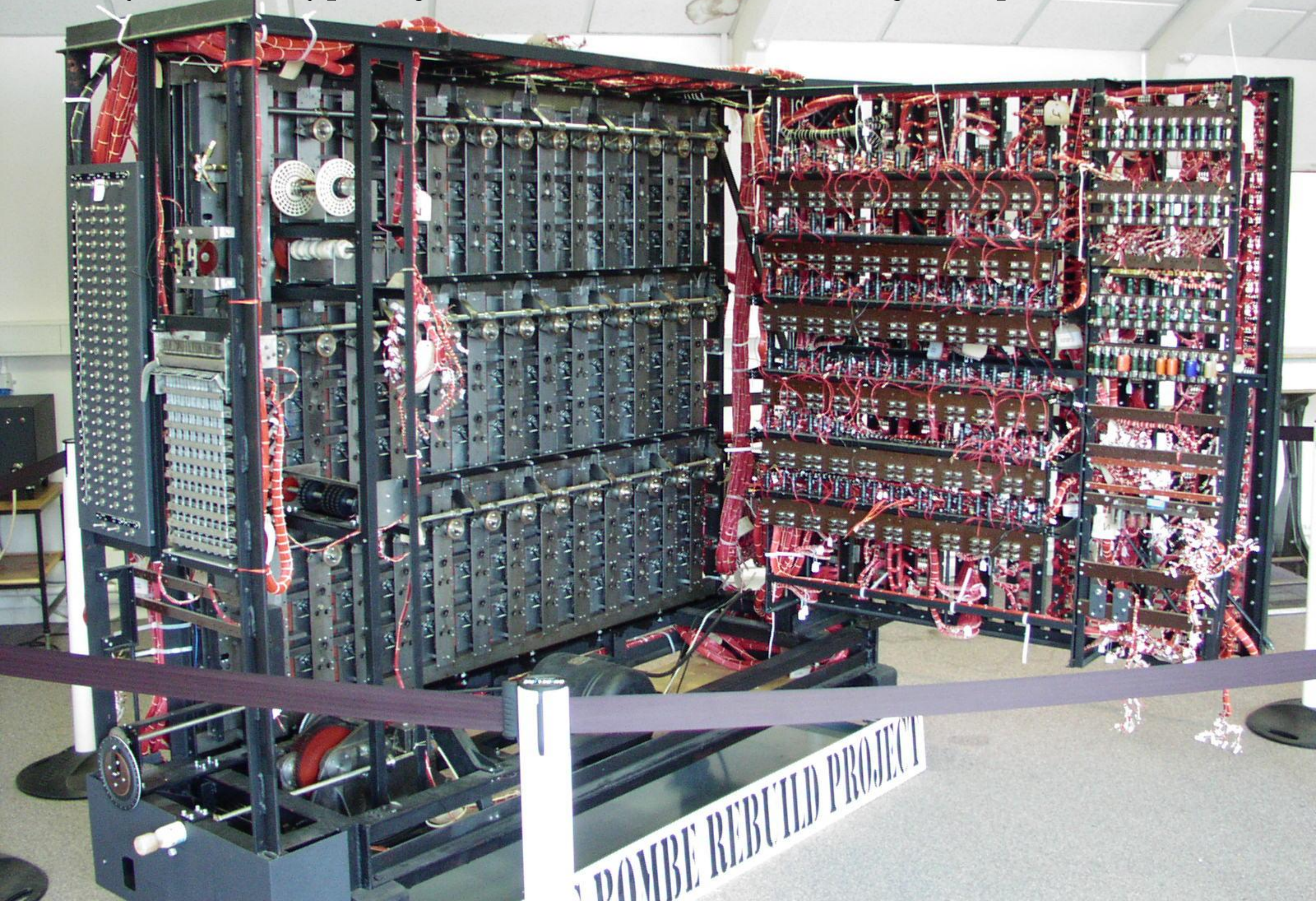

ALAN TURING
1912 - 1954
Founder of computer science
and cryptographer, whose work
was key to breaking the
wartime Enigma codes,
lived and died here.


Alan Turing
1912 - 1954
Mathematician - codebreaker
This statue was donated by
The Sidney E. Frank Foundation
and unveiled by
Abel Hadden
on 19th June 2007


© Guy Erwood 2007. All rights reserved

Bletchley Park ("Station X"), Bletchley, Buckinghamshire, England
England's code-breaking and cryptanalysis center during WWII

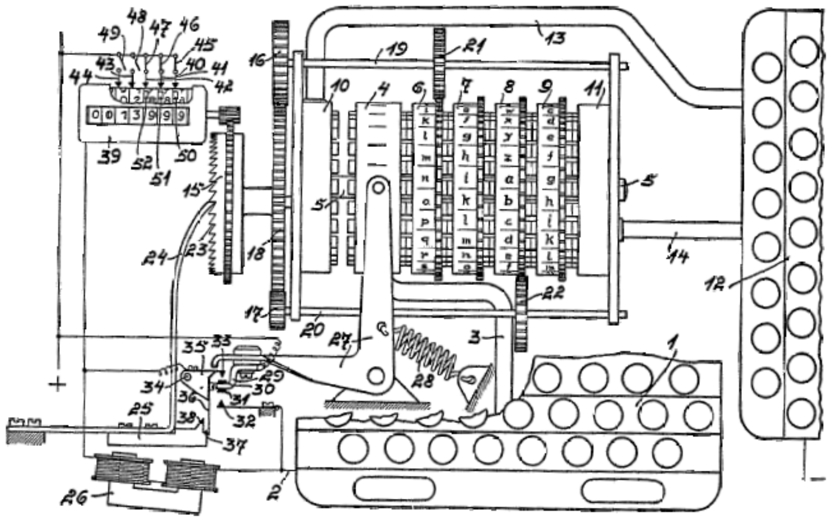
“Bombe” - electromechanical computer designed by Alan Turing.
Used by British cryptologists to break the German Enigma cipher



1918 First Enigma Patent

The official history of the Enigma starts in 1918, when the German **Arthur Scherbius** filed his first patent for the Enigma coding machine. It is listed as patent number 416219 in the archives of the German *Reichspatentamt* (patent office). Please note the time at which the Enigma was invented: **1918**, just after the First World War, more then 20 years before WWII! The image below clearly shows the coding wheels (rotors) in the centre part of the drawing. Below it is the keyboard and to the right is the lamp panel. At the top left is a counter, used to count the number of letters entered on the keyboard. This counter can still be found on certain Enigma models.

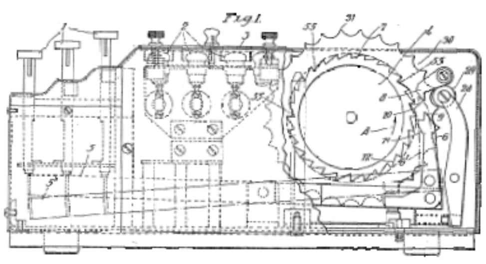
Arthur Scherbius' company **Securitas** was based in Berlin (Germany) and had an office in Amsterdam (The Netherlands). As he wanted to protect his invention outside Germany, he also registered his patent in the USA (1922), Great Britain (1923) and France (1923).

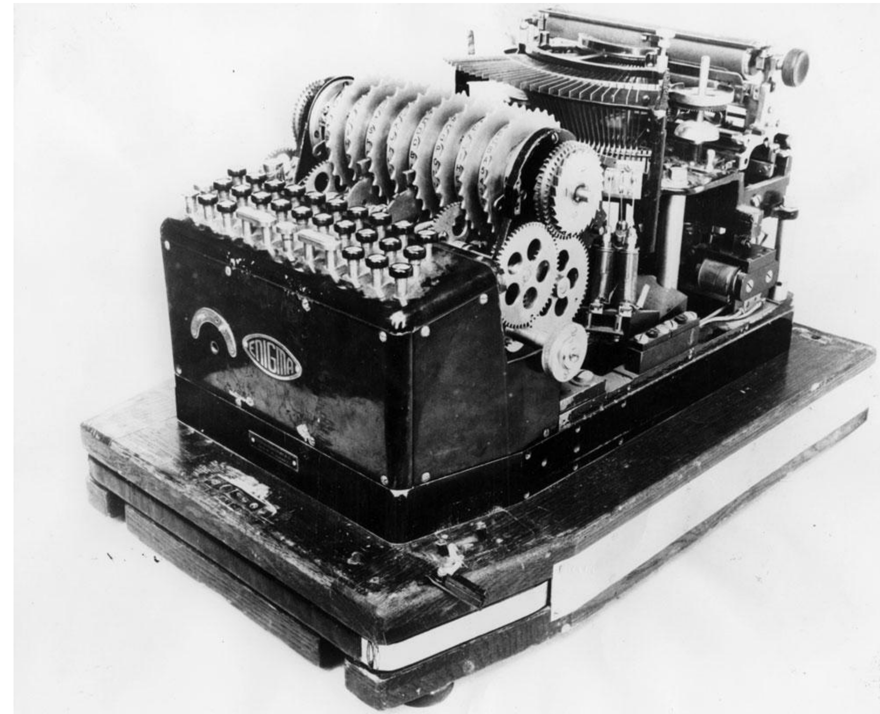


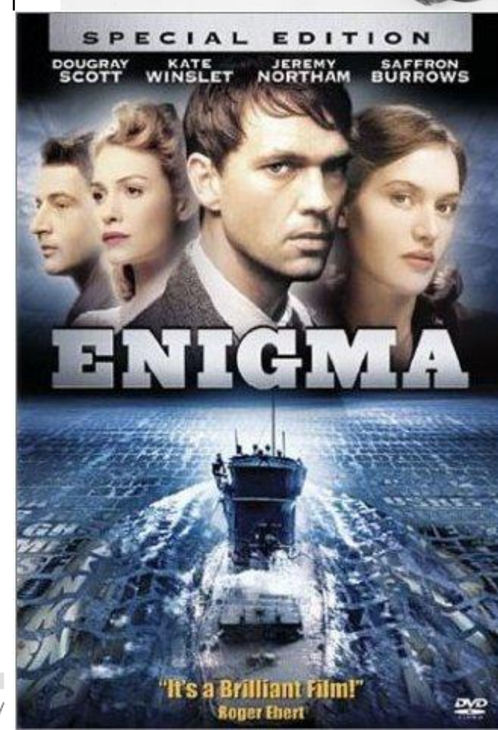
This image is taken from patent number 193,035 that was registered in Great Britain in 1923, long before WWII. It was also registered in a number of other countries, such as France and the USA.

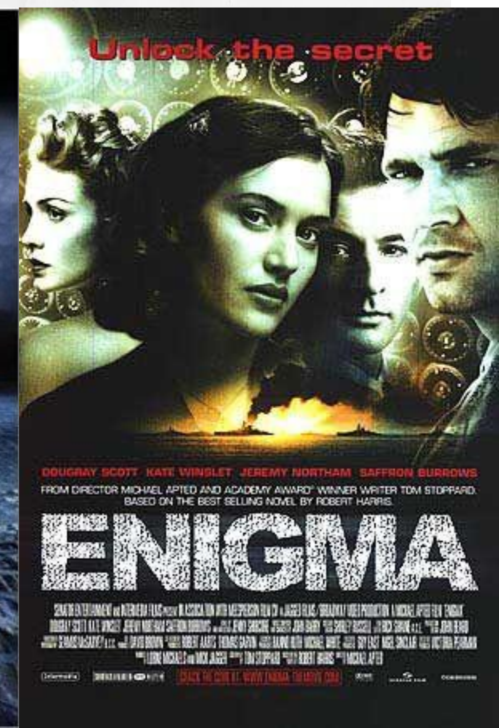
During the 1920s the Enigma was available as a commercial device, available for use by companies and embassies for their confidential messages. Remember that in those days, most companies had to use morse code and radio links for long distance communication. The devices were advertised having over 800.000 possibilities.

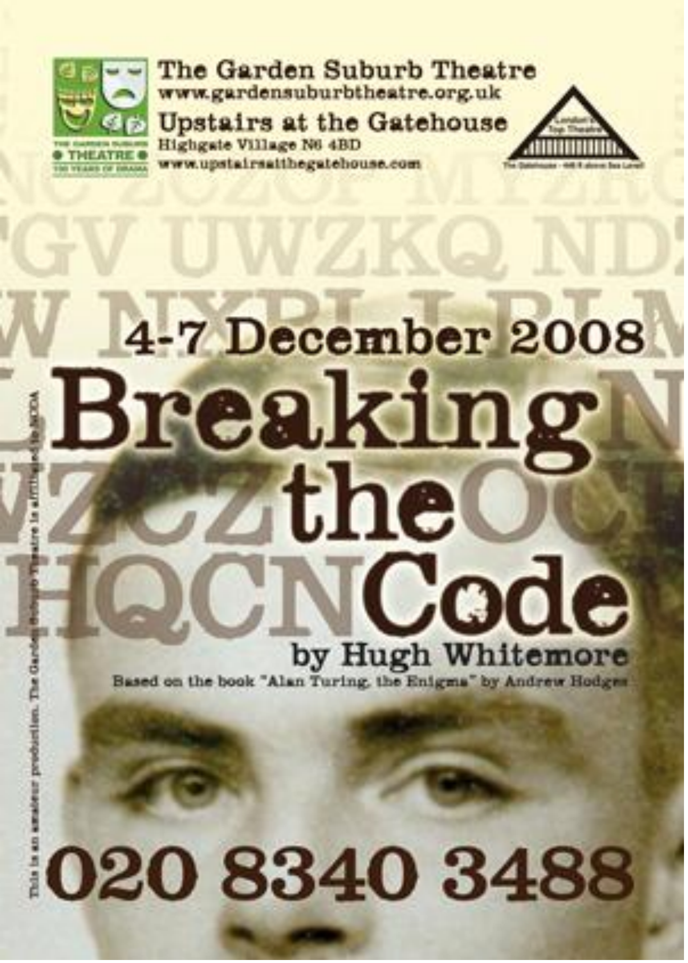
In the following years, additional patents with improvements of the coding machine were applied. E.g. in GB Patent 267,482, dated 17 Jan 1927, the Umkehrwalze was added and a later patent of 14 Nov 1929 (GB 343,146) claims the addition of the Ringstellung, multiple notches, etc. One of the drawings of that patent shows a coding device, that we now know as The Enigma, in great detail.

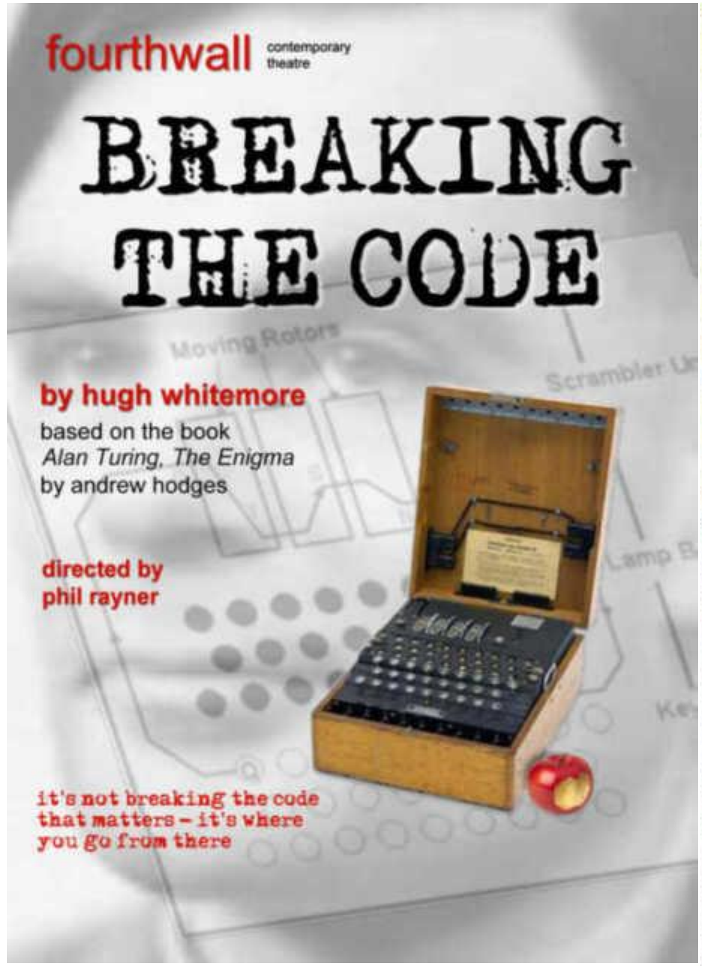


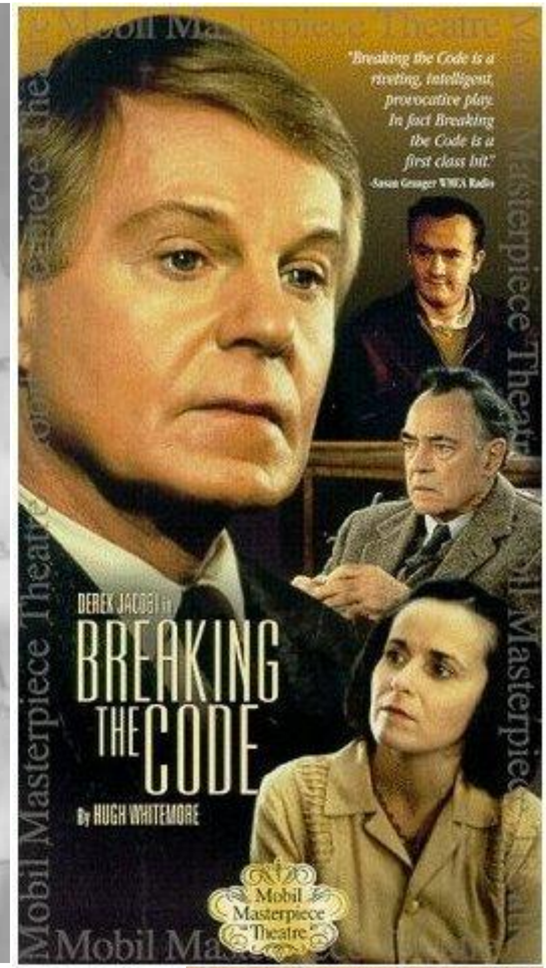




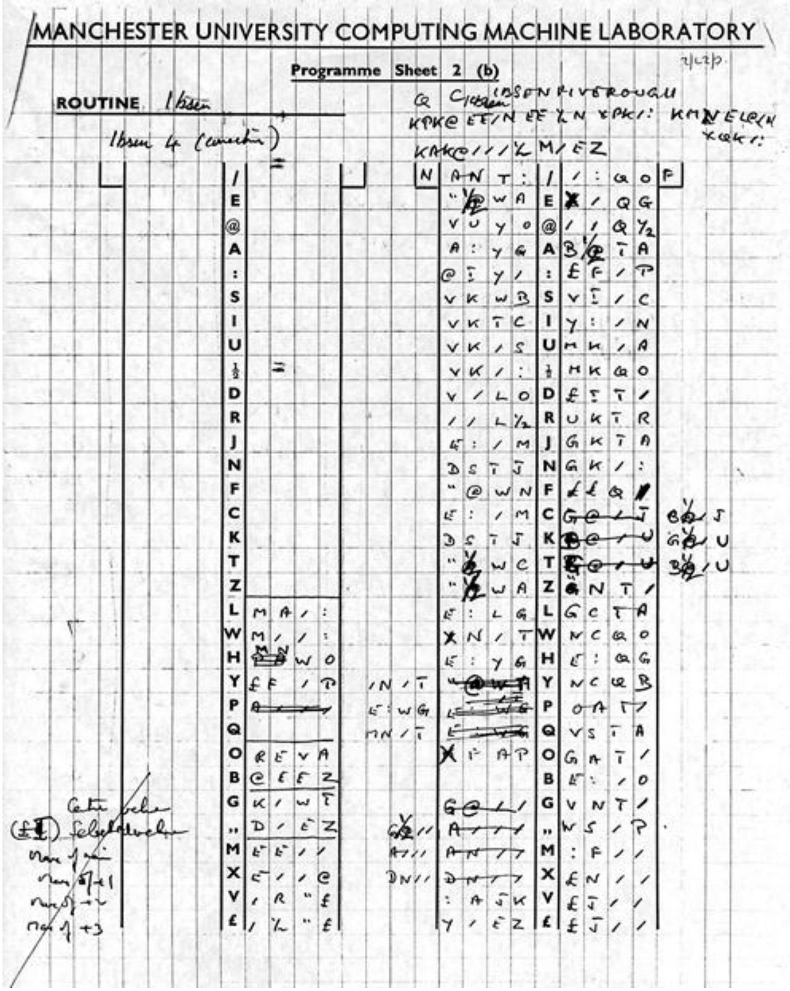



The Garden Suburb Theatre
www.gardensuburbtheatre.org.uk
Upstairs at the Gatehouse
Highgate Village N6 4BD
www.upstairsatthegatehouse.com
4-7 December 2008
Breaking the Code
by Hugh Whitemore
Based on the book "Alan Turing, the Enigma" by Andrew Hodges
020 8340 3488

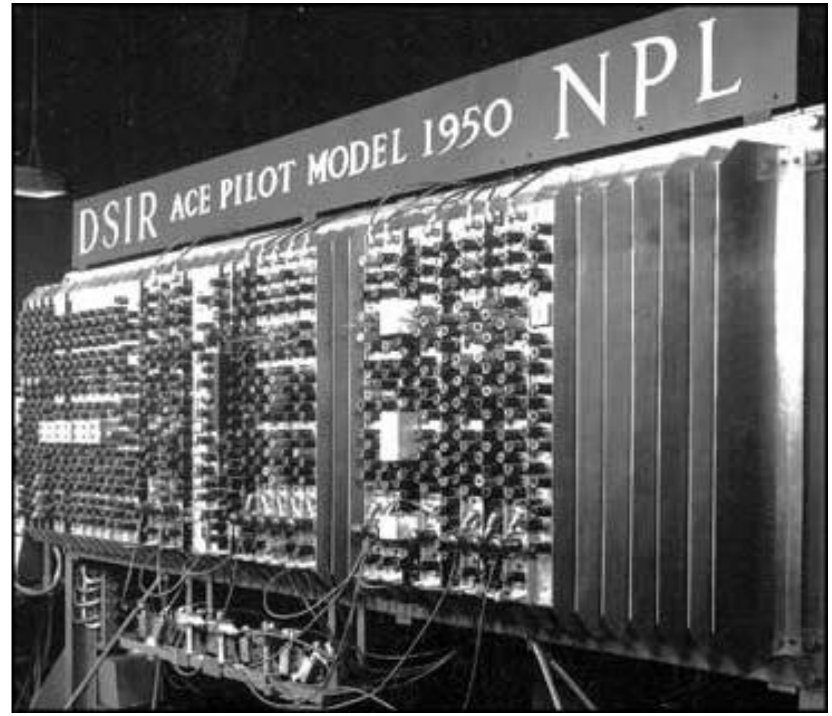

fourthwall
contemporary theatre
BREAKING THE CODE
by hugh whitemore
based on the book
Alan Turing, The Enigma
by andrew hodges
directed by
phil rayner
it's not breaking the code
that matters – it's where
you go from there

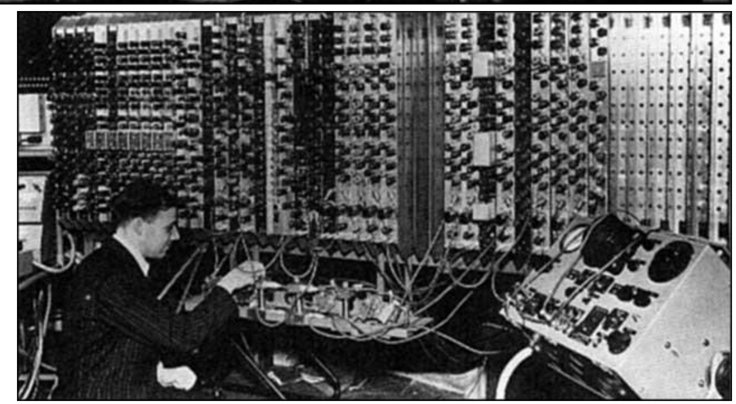

"Breaking the Code is a riveting, intelligent, provocative play. In fact Breaking the Code is a first class hit."
DEREK JACOBI in
BREAKING THE CODE
By HUGH WHITEMORE
Mobil Masterpiece Theatre

ALAN TURING, 1912 - 1954



Program for ACE computer
hand-written by Alan Turing

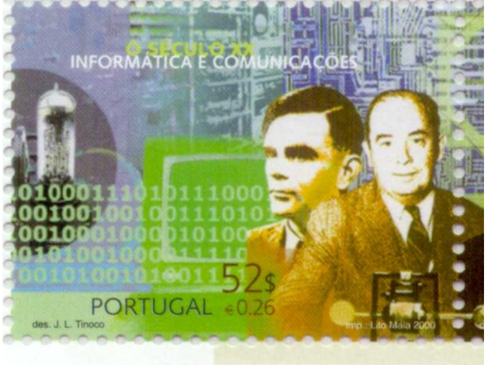




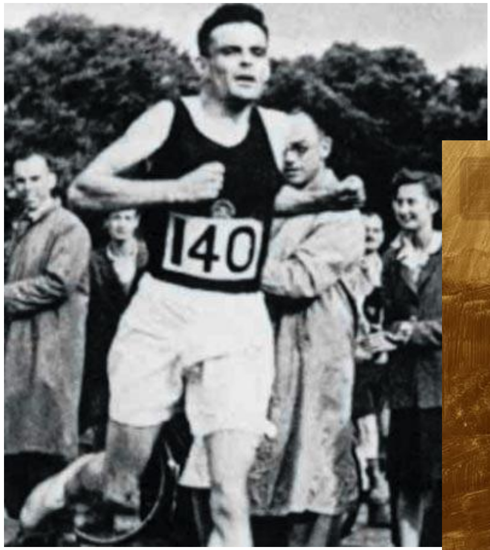

CHARLES PETZOLD
THE ANNOTATED TURING
A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine
EDITED BY B. JACK COPELAND
THE ESSENTIAL TURING
The ideas that gave birth to the computer age
ALAN TURING: THE ENIGMA
ANDREW HODGES
Foreword by DOUGLAS HOFSTADTER
WITH A NEW PREFACE
THE UNIVERSAL COMPUTER
THE ROAD FROM LEIBNIZ TO TURING
MARTIN DAVIS
ALAN TURING'S AUTOMATIC COMPUTING ENGINE
EDITED BY B. JACK COPELAND
OXFORD
PROFILES IN MATHEMATICS
Alan Turing
Jim Corrigan
Christof Teuscher (Ed.)
Alan Turing: Life and Legacy of a Great Thinker
Springer
A MADMAN DREAMS OF Turing Machines
a novel
by Janna Levin
Turing
(A Novel about Computation)
Christos H. Papadimitriou
The Turing Test and the Frame Problem
AI's Mistaken Understanding of Intelligence
Larry J. Crockett
MAKERS OF MODERN SCIENCE
ALAN TURING
Computing Genius and Wartime Codebreaker
RAY SPANGENBURG AND DIANE KIT MOSER
THE GREAT PHILOSOPHERS
TURING
Andrew Hodges
THE NEW TURING OMNIBUS
66 EXCURSIONS IN COMPUTER SCIENCE
A.K.DEWDNEY
DAVID LEAVITT
Alan Turing and the Invention of the Computer
THE MAN WHO KNEW TOO MUCH
GREAT DISCOVERIES
The Universal Turing Machine
A Half-Century Survey
Springer-Verlag Wien New York
ALAN TURING
The ARCHITECT of the COMPUTER AGE
by TED GOTTFRIED
THE BIG iDEA
TURING AND THE COMPUTER
PAUL STRATHERN
Turing and the Universal Machine
Jon Agar
The Making of the Modern Computer

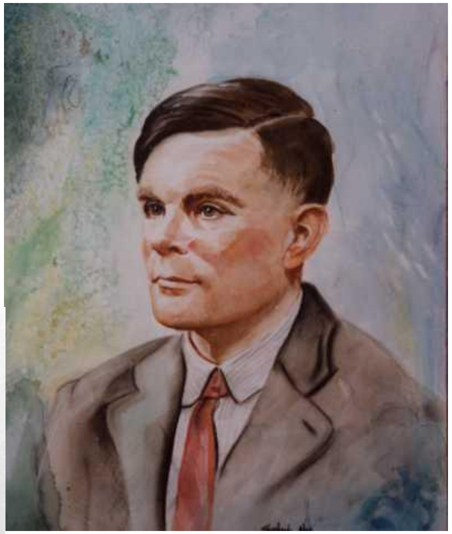

50 rocznica
złamania szyfru
Enigmy
POLSKA
5zł


St. Vincent
& the Grenadines
20¢
MILLENNIUM
1937: Alan Turing's theory of digital
computing


O SÉCULO XX
INFORMÁTICA E COMUNICAÇÕES
52$
PORTUGAL €0.26
des. J. L. Tinoco


Alan M. Turing, codebreaker
and an Enigma Code Machine
30p
ST HELENA


140




ALAN MATHISON TURING 1912-1954
2006
35
TATARSTAN

MIXX SHARE EMAIL SAVE PRINT

British PM apologizes for treatment of gay code-breaker

updated 6:17 a.m. EDT, Fri September 11, 2009

STORY HIGHLIGHTS
- Online petition calling for apology for Alan Turing elicits response from UK PM
- Gordon Brown issues statement apologizing for his "appalling" treatment
- Turing committed suicide two years after undergoing chemical castration
- Best known for decoding messages from German Enigma machines in WWII

Next Article in World »

By Hilary Whiteman
CNN

TEXT SIZE

LONDON, England (CNN) -- British Prime Minister Gordon Brown has issued a posthumous apology for the "appalling" treatment of Alan Turing, the British code-breaker who was chemically castrated for being gay.

A portrait of Alan Turing is currently on display at the National Portrait Gallery's "Gay Icons" exhibition.

1 of 3

The apology came after more than 30,000 people signed an online petition on the UK Government Web site calling for the government to recognize the "tragic consequences of prejudice that ended this man's life and career."

Turing was just 41 years old when he committed suicide, two years after undergoing a court-ordered chemical castration. He had been found guilty of gross indecency for having a homosexual relationship. The punishment in 1952 was either a prison sentence or chemical castration. Turing chose the latter.

In a statement on the British Government Web site, Prime Minister Gordon Brown acknowledged Turing's "outstanding" contribution during World War II.

"He truly was one of those individuals we can point to whose unique contribution helped to turn the tide of war," he wrote, adding, "The debt of gratitude he is owed makes it all the more horrifying, therefore, that he was treated so inhumanely."

Turing is considered one of Britain's greatest mathematicians, a genius who is credited with inventing the Bombe, a code-breaking machine that deciphered messages encoded by German Enigma machines during World War II.

He went on to develop the Turing machine, a theory that automatic computation cannot solve all mathematical problems, which is considered the basis of modern computing.

Don't Miss

- Petition seeks apology for Enigma code-breaker Turing
- Leaders mark 70th anniversary of WWII

Last month, the curious lack of public recognition for Turing's contribution to the war effort and computing in general motivated computer programmer John Graham-Cumming to campaign on his behalf.

The author of the "Geek Atlas," a travel guide for technology enthusiasts, started an online petition, and soon attracted high-profile signatories including scientist Richard Dawkins, actor Stephen Fry, author Ian McEwan and philosopher A.C. Grayling.

"I was surprised by both the number of people who signed and the fast response from the government," Graham-Cumming told CNN. He said the Prime Minister had called him personally to relay news of the apology.

Stories about calls for a British apology were carried in newspapers in France, Switzerland, Spain, Austria, Portugal Poland and the Czech Republic. Supporters set up an international petition which attracted more than 10,000 signatures.

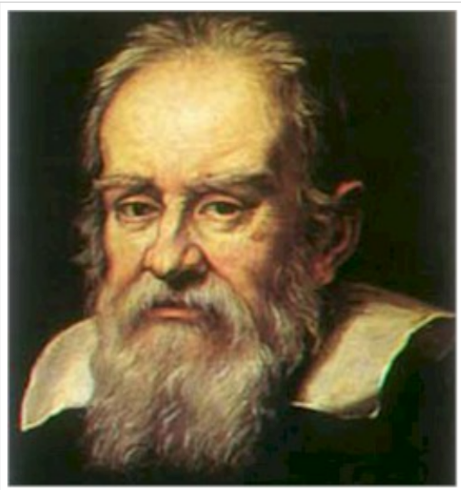
Another famous belated apology:

For The First Time (or the last time): 1992: Catholic Church apologizes to Galileo, who died in 1642 - Mozilla Firefox

http://4thefirsttime.blogspot.com/2007/09/1992-catholic-church-apologizes-to.h

Monday, September 10, 2007

1992: Catholic Church apologizes to Galileo, who died in 1642



In 1610, Century Italian astronomer/mathematician /inventor Galileo Galilei used a a telescope he built to observe the solar system, and deduced that the planets orbit the sun, not the earth.

This contradicted Church teachings, and some of the clergy accused Galileo of heresy. One friar went to the Inquisition, the Church court that investigated charges of heresy, and formally accused Galileo. (In 1600, a man named Giordano Bruno was convicted of being a heretic for believing that the earth moved around the Sun, and that there were many planets throughout the universe where life existed. Bruno was burnt to death.)

Galileo moved on to other projects. He started writing about ocean tides, but instead of writing a scientific paper, he found it much more interesting to have an imaginary conversation among three fictional characters. One character, who would support Galileo's side of the argument, was brilliant. Another character would be open to either side of the argument. The final character, named Simplicio, was dogmatic and foolish, representing all of Galileo's enemies who ignored any evidence that Galileo was right. Soon, Galileo wrote up a sim dialogue called "Dialogue on the Two Great Systems of the W This book talked about the Copernican system.

"Dialogue" was an immediate hit with the public, but not, of course, with the Church. The pope suspected that he was the model for Simplicio. He ordered the book banned, and also ordered Galileo to appear before the Inquisition in Rome for the crime of teaching the Copernican theory after being ordered not to do so.

Galileo was 68 years old and sick. Threatened with torture, he publicly confessed that he had been wrong to have said that the Earth moves around the Sun. Legend then has it that after his confession, Galileo quietly whispered "And yet, it moves."

Unlike many less famous prisoners, Galileo was allowed to live under house arrest. Until his death in 1642, he continued to investigate science, and even published a book on force and motion after he had become blind.

The Church eventually lifted the ban on Galileo's Dialogue in 1822, when it was common knowledge that the Earth was not the center of the Universe. Still later, there were statements by the Vatican Council in the early 1960's and in 1979 that implied that Galileo was pardoned, and that he had suffered at the hands of the Church. Finally, in 1992, three years after Galileo Galilei's namesake spacecraft had been launched on its way to Jupiter, the Vatican formally and publicly cleared Galileo of any wrongdoing.

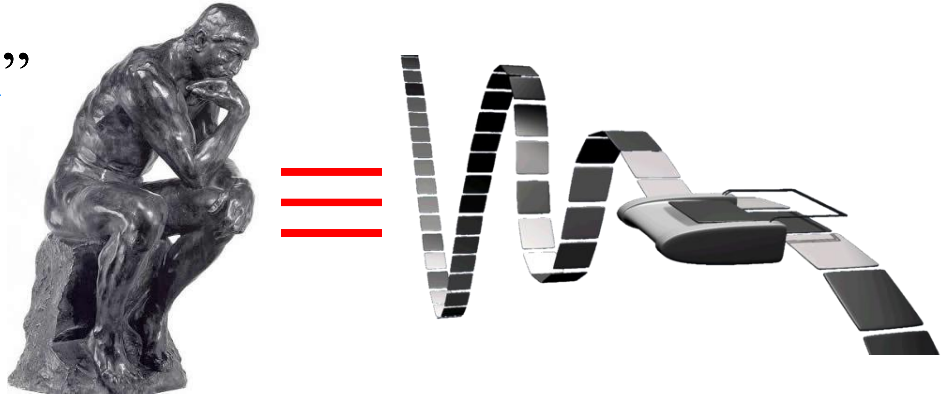
(info from NASA and the H
Susterma

Theorem: A late apology is better than no apology.
Corollary: But sooner is better!

Turing's Seminal Paper

"On Computable Numbers, with an Application to the Entscheidungsproblem", Proceedings of the London Mathematical Society, 1937, pp. 230-265.

- One of the most influential & significant papers ever!
- First formal model of "computation"
- First ever definition of "algorithm"
- Invented "Turing machines"
- Introduced "computational universality" i.e., "programmable"!
- Proved the undecidability of halting problem
- Explicates the Church-Turing Thesis



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. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

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.

have valuable applications. In particular, it is shown (§ 11) that the Hilbertian Entscheidungsproblem can have no solution.

In a recent paper Alonzo Church† has introduced an idea of "effective calculability", which is equivalent to my "computability", but is very differently defined. Church also reaches similar conclusions about the Entscheidungsproblem‡. The proof of equivalence between "computability" and "effective calculability" is outlined in an appendix to the present paper.

1. *Computing machines.*

We have said that the computable numbers are those whose decimals are calculable by finite means. This requires rather more explicit definition. No real attempt will be made to justify the definitions given until we reach § 9. For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited.

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions $q_1, q_2, \dots, q_R$ which will be called "m-configurations". The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its m-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair $q_n, \mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (*i.e.* bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the m-configuration may be changed. Some of the symbols written down

† Alonzo Church, "An unsolvable problem of elementary number theory", *American J. of Math.*, 58 (1936), 345–363.

‡ Alonzo Church, "A note on the Entscheidungsproblem", *J. of Symbolic Logic*, 1 (1936), 40–41.

will form the sequence of figures which is the decimal of the real number which is being computed. The others are just rough notes to "assist the memory". It will only be these rough notes which will be liable to erasure.

It is my contention that these operations include all those which are used in the computation of a number. The defence of this contention will be easier when the theory of the machines is familiar to the reader. In the next section I therefore proceed with the development of the theory and assume that it is understood what is meant by "machine", "tape", "scanned", etc.

2. *Definitions.*

Turing

~~Automatic~~ machines.

If at each stage the motion of a machine (in the sense of § 1) is *completely* determined by the configuration, we shall call the machine an "automatic machine" (or a-machine).

For some purposes we might use machines (choice machines or c-machines) whose motion is only partially determined by the configuration (hence the use of the word "possible" in § 1). When such a machine reaches one of these ambiguous configurations, it cannot go on until some arbitrary choice has been made by an external operator. This would be the case if we were using machines to deal with axiomatic systems. In this paper I deal only with automatic machines, and will therefore often omit the prefix a-.

Computing machines.

If an a-machine prints two kinds of symbols, of which the first kind (called figures) consists entirely of 0 and 1 (the others being called symbols of the second kind), then the machine will be called a computing machine. If the machine is supplied with a blank tape and set in motion, starting from the correct initial m-configuration, the subsequence of the symbols printed by it which are of the first kind will be called the *sequence computed by the machine*. The real number whose expression as a binary decimal is obtained by prefacing this sequence by a decimal point is called the *number computed by the machine*.

At any stage of the motion of the machine, the number of the scanned square, the complete sequence of all symbols on the tape, and the m-configuration will be said to describe the *complete configuration* at that stage. The changes of the machine and tape between successive complete configurations will be called the *moves* of the machine.

Circular and circle-free machines.

If a computing machine never writes down more than a finite number of symbols of the first kind, it will be called *circular*. Otherwise it is said to be *circle-free*.

A machine will be circular if it reaches a configuration from which there is no possible move, or if it goes on moving, and possibly printing symbols of the second kind, but cannot print any more symbols of the first kind. The significance of the term "circular" will be explained in § 8.

Computable sequences and numbers.

A sequence is said to be computable if it can be computed by a circle-free machine. A number is computable if it differs by an integer from the number computed by a circle-free machine.

We shall avoid confusion by speaking more often of computable sequences than of computable numbers.

3. *Examples of computing machines.*

I. A machine can be constructed to compute the sequence 010101.... The machine is to have the four m-configurations "$\mathfrak{b}$", "$\mathfrak{c}$", "$\mathfrak{f}$", "$\mathfrak{e}$" and is capable of printing "0" and "1". The behaviour of the machine is described in the following table in which "R" means "the machine moves so that it scans the square immediately on the right of the one it was scanning previously". Similarly for "L". "E" means "the scanned symbol is erased" and "P" stands for "prints". This table (and all succeeding tables of the same kind) is to be understood to mean that for a configuration described in the first two columns the operations in the third column are carried out successively, and the machine then goes over into the m-configuration described in the last column. When the second column is left blank, it is understood that the behaviour of the third and fourth columns applies for any symbol and for no symbol. The machine starts in the m-configuration $\mathfrak{b}$ with a blank tape.

Configuration		*Behaviour*	
m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{e}$
$\mathfrak{e}$	None	$P1, R$	$\mathfrak{f}$
$\mathfrak{f}$	None	R	$\mathfrak{b}$

If (contrary to the description in § 1) we allow the letters L, R to appear more than once in the operations column we can simplify the table considerably.

m-config.	*symbol*	*operations*	*final m-config.*
𝔟	None	$P0$	𝔟
	0	$R, R, P1$	𝔟
	1	$R, R, P0$	𝔟

II. As a slightly more difficult example we can construct a machine to compute the sequence 001011011101111011111 .... The machine is to be capable of five m-configurations, viz. "𝔬", "𝔮", "𝔭", "𝔣", "𝔟" and of printing "ə", "x", "0", "1". The first three symbols on the tape will be "əə0"; the other figures follow on alternate squares. On the intermediate squares we never print anything but "x". These letters serve to "keep the place" for us and are erased when we have finished with them. We also arrange that in the sequence of figures on alternate squares there shall be no blanks.

Configuration		*Behaviour*	
m-config.	*symbol*	*operations*	*final m-config.*
𝔟		Pə$, R, P$ə$, R, P0, R, R, P0, L, L$	𝔬
𝔬	1	R, Px, L, L, L	𝔬
	0		𝔮
𝔮	Any (0 or 1)	R, R	𝔮
	None	$P1, L$	𝔭
𝔭	x	E, R	𝔮
	ə	R	𝔣
	None	L, L	𝔭
𝔣	Any	R, R	𝔣
	None	$P0, L, L$	𝔬

To illustrate the working of this machine a table is given below of the first few complete configurations. These complete configurations are described by writing down the sequence of symbols which are on the tape, with the m-configuration written below the scanned symbol. The successive complete configurations are separated by colons.


```
  : ə ə 0   0 : ə ə 0   0 : ə ə 0   0 : ə ə 0   0     : ə ə 0   0   1 :
𝔟          𝔬           𝔮                 𝔮              𝔮                 𝔭
ə ə 0   0   1 : ə ə 0   0   1 : ə ə 0   0   1 : ə ə 0   0   1 :
      𝔭           𝔭                 𝔣                       𝔣
ə ə 0   0   1 : ə ə 0   0   1     : ə ə 0   0   1   0 :
            𝔣                 𝔣               𝔬
ə ə 0   0   1 x 0 : ....
        𝔬
```

This table could also be written in the form

$$𝔟 : ə\ ə\ 𝔬\ 0 \quad 0 : ə\ ə\ 𝔮\ 0 \quad 0 : \ldots, \qquad (C)$$

in which a space has been made on the left of the scanned symbol and the m-configuration written in this space. This form is less easy to follow, but we shall make use of it later for theoretical purposes.

The convention of writing the figures only on alternate squares is very useful: I shall always make use of it. I shall call the one sequence of alternate squares F-squares and the other sequence E-squares. The symbols on E-squares will be liable to erasure. The symbols on F-squares form a continuous sequence. There are no blanks until the end is reached. There is no need to have more than one E-square between each pair of F-squares: an apparent need of more E-squares can be satisfied by having a sufficiently rich variety of symbols capable of being printed on E-squares. If a symbol β is on an F-square S and a symbol α is on the E-square next on the right of S, then S and β will be said to be *marked* with α. The process of printing this α will be called marking β (or S) with α.

4. *Abbreviated tables.*

There are certain types of process used by nearly all machines, and these, in some machines, are used in many connections. These processes include copying down sequences of symbols, comparing sequences, erasing all symbols of a given form, etc. Where such processes are concerned we can abbreviate the tables for the m-configurations considerably by the use of "skeleton tables". In skeleton tables there appear capital German letters and small Greek letters. These are of the nature of "variables". By replacing each capital German letter throughout by an m-configuration and each small Greek letter by a symbol, we obtain the table for an m-configuration.

The skeleton tables are to be regarded as nothing but abbreviations: they are not essential. So long as the reader understands how to obtain the complete tables from the skeleton tables, there is no need to give any exact definitions in this connection.

Let us consider an example:

m-config.	*Symbol*	*Behaviour*	*Final m-config.*
$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$	ə	L	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	not ə	L	$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$
$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	α		$\mathfrak{C}$
	not α	R	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	None	R	$\mathfrak{f}_2(\mathfrak{C}, \mathfrak{B}, \alpha)$
$\mathfrak{f}_2(\mathfrak{C}, \mathfrak{B}, \alpha)$	α		$\mathfrak{C}$
	not α	R	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	None	R	$\mathfrak{B}$

From the m-configuration $\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$ the machine finds the symbol of form α which is farthest to the left (the "first α") and the m-configuration then becomes $\mathfrak{C}$. If there is no α then the m-configuration becomes $\mathfrak{B}$.

If we were to replace $\mathfrak{C}$ throughout by $\mathfrak{q}$ (say), $\mathfrak{B}$ by $\mathfrak{r}$, and α by x, we should have a complete table for the m-configuration $\mathfrak{f}(\mathfrak{q}, \mathfrak{r}, x)$. $\mathfrak{f}$ is called an "m-configuration function" or "m-function".

The only expressions which are admissible for substitution in an m-function are the m-configurations and symbols of the machine. These have to be enumerated more or less explicitly: they may include expressions such as $\mathfrak{p}(\mathfrak{e}, x)$; indeed they must if there are any m-functions used at all. If we did not insist on this explicit enumeration, but simply stated that the machine had certain m-configurations (enumerated) and all m-configurations obtainable by substitution of m-configurations in certain m-functions, we should usually get an infinity of m-configurations; *e.g.*, we might say that the machine was to have the m-configuration $\mathfrak{q}$ and all m-configurations obtainable by substituting an m-configuration for $\mathfrak{C}$ in $\mathfrak{p}(\mathfrak{C})$. Then it would have $\mathfrak{q}$, $\mathfrak{p}(\mathfrak{q})$, $\mathfrak{p}\big(\mathfrak{p}(\mathfrak{q})\big)$, $\mathfrak{p}\Big(\mathfrak{p}\big(\mathfrak{p}(\mathfrak{q})\big)\Big)$, ... as m-configurations.

Our interpretation rule then is this. We are given the names of the m-configurations of the machine, mostly expressed in terms of m-functions. We are also given skeleton tables. All we want is the complete table for the m-configurations of the machine. This is obtained by repeated substitution in the skeleton tables.

Further examples.

(In the explanations the symbol "$\rightarrow$" is used to signify "the machine goes into the m-configuration. . . .")

$\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{f}\big(\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha\big)$
$\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$		E	$\mathfrak{C}$

From $\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$ the first α is erased and $\rightarrow \mathfrak{C}$. If there is no $\alpha \rightarrow \mathfrak{B}$.

$\mathfrak{e}(\mathfrak{B}, \alpha)$			$\mathfrak{e}\big(\mathfrak{e}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha\big)$

From $\mathfrak{e}(\mathfrak{B}, \alpha)$ all letters α are erased and $\rightarrow \mathfrak{B}$.

The last example seems somewhat more difficult to interpret than most. Let us suppose that in the list of m-configurations of some machine there appears $\mathfrak{e}(\mathfrak{b}, x)$ ($= \mathfrak{q}$, say). The table is

	$\mathfrak{e}(\mathfrak{b}, x)$	$\mathfrak{e}\big(\mathfrak{e}(\mathfrak{b}, x), \mathfrak{b}, x\big)$
or	$\mathfrak{q}$	$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$.

Or, in greater detail:

$\mathfrak{q}$		$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$
$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$		$\mathfrak{f}\big(\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x), \mathfrak{b}, x\big)$
$\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$	E	$\mathfrak{q}$.

In this we could replace $\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$ by $\mathfrak{q}'$ and then give the table for $\mathfrak{f}$ (with the right substitutions) and eventually reach a table in which no m-functions appeared.

$\mathfrak{pe}(\mathfrak{C}, \beta)$			$\mathfrak{f}\big(\mathfrak{pe}_1(\mathfrak{C}, \beta), \mathfrak{C}, ə\big)$
$\mathfrak{pe}_1(\mathfrak{C}, \beta)$	Any	R, R	$\mathfrak{pe}_1(\mathfrak{C}, \beta)$
	None	$P\beta$	$\mathfrak{C}$

From $\mathfrak{pe}$ $(\mathfrak{C}, \beta)$ the machine prints β at the end of the sequence of symbols and $\rightarrow \mathfrak{C}$.

$\mathfrak{l}(\mathfrak{C})$	L	$\mathfrak{C}$
$\mathfrak{r}(\mathfrak{C})$	R	$\mathfrak{C}$
$\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}\big(\mathfrak{l}(\mathfrak{C}), \mathfrak{B}, \alpha\big)$
$\mathfrak{f}''(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}\big(\mathfrak{r}(\mathfrak{C}), \mathfrak{B}, \alpha\big)$

From $\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$ it does the same as for $\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$ but moves to the left before $\rightarrow \mathfrak{C}$.

$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}'\big(\mathfrak{c}_1(\mathfrak{C}), \mathfrak{B}, \alpha\big)$
$\mathfrak{c}_1(\mathfrak{C})$	β	$\mathfrak{pe}(\mathfrak{C}, \beta)$

$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$. The machine writes at the end the first symbol marked α and $\rightarrow \mathfrak{C}$.

The last line stands for the totality of lines obtainable from it by replacing β by any symbol which may occur on the tape of the machine concerned.

Transition function

$\mathfrak{ce}(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{c}\big(\mathfrak{ce}(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha\big)$
$\mathfrak{ce}(\mathfrak{B}, \alpha)$			$\mathfrak{ce}\big(\mathfrak{ce}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha\big)$

$\mathfrak{ce}(\mathfrak{B}, \alpha)$. The machine copies down in order at the end all symbols marked α and erases the letters α; $\rightarrow \mathfrak{B}$.

$\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \beta)$			$\mathfrak{f}\big(\mathfrak{re}_1(\mathfrak{C}, \mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha\big)$
$\mathfrak{re}_1(\mathfrak{C}, \mathfrak{B}, \alpha, \beta)$		$E, P\beta$	$\mathfrak{C}$
$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$			$\mathfrak{re}\big(\mathfrak{re}(\mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha, \beta\big)$

$\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \beta)$. The machine replaces the first α by β and $\rightarrow \mathfrak{C} \rightarrow \mathfrak{B}$ if there is no α.

$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$. The machine replaces all letters α by β; $\rightarrow \mathfrak{B}$.

$\mathfrak{cr}(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{c}\big(\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \alpha), \mathfrak{B}, \alpha\big)$
$\mathfrak{cr}(\mathfrak{B}, \alpha)$			$\mathfrak{cr}\big(\mathfrak{cr}(\mathfrak{B}, \alpha), \mathfrak{re}(\mathfrak{B}, \alpha, \alpha), \alpha\big)$

$\mathfrak{cr}(\mathfrak{B}, \alpha)$ differs from $\mathfrak{ce}(\mathfrak{B}, \alpha)$ only in that the letters α are not erased. The m-configuration $\mathfrak{cr}(\mathfrak{B}, \alpha)$ is taken up when no letters "α" are on the tape.

$\mathfrak{cp}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$			$\mathfrak{f}'\big(\mathfrak{cp}_1(\mathfrak{C}_1 \mathfrak{A}, \beta), \mathfrak{f}(\mathfrak{A}, \mathfrak{E}, \beta), \alpha\big)$
$\mathfrak{cp}_1(\mathfrak{C}, \mathfrak{A}, \beta)$	γ		$\mathfrak{f}'\big(\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma), \mathfrak{A}, \beta\big)$
$\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma)$	γ		$\mathfrak{C}$
	not γ		$\mathfrak{A}$.

The first symbol marked α and the first marked β are compared. If there is neither α nor β, $\rightarrow \mathfrak{E}$. If there are both and the symbols are alike, $\rightarrow \mathfrak{C}$. Otherwise $\rightarrow \mathfrak{A}$.

$$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta) \qquad \mathfrak{cp}\Big(\mathfrak{e}\big(\mathfrak{e}(\mathfrak{C}, \mathfrak{C}, \beta), \mathfrak{C}, \alpha\big), \mathfrak{A}, \mathfrak{E}, \alpha, \beta\Big)$$

$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$ differs from $\mathfrak{cp}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$ in that in the case when there is similarity the first α and β are erased.

$$\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta) \qquad \mathfrak{cpe}\big(\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta), \mathfrak{A}, \mathfrak{E}, \alpha, \beta\big).$$

$\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta)$. The sequence of symbols marked α is compared with the sequence marked β. $\rightarrow \mathfrak{E}$ if they are similar. Otherwise $\rightarrow \mathfrak{A}$. Some of the symbols α and β are erased.

$\mathfrak{q}(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$
	None	R	$\mathfrak{q}_1(\mathfrak{C})$
$\mathfrak{q}_1(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$
	None		$\mathfrak{C}$
$\mathfrak{q}(\mathfrak{C}, \alpha)$			$\mathfrak{q}\big(\mathfrak{q}_1(\mathfrak{C}, \alpha)\big)$
$\mathfrak{q}_1(\mathfrak{C}, \alpha)$	α		$\mathfrak{C}$
	not α	L	$\mathfrak{q}_1(\mathfrak{C}, \alpha)$

$\mathfrak{q}(\mathfrak{C}, \alpha)$. The machine finds the last symbol of form α. $\rightarrow \mathfrak{C}$.

$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$			$\mathfrak{pe}\big(\mathfrak{pe}(\mathfrak{C}, \beta), \alpha\big)$

$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$. The machine prints $\alpha\, \beta$ at the end.

$\mathfrak{ce}_2(\mathfrak{B}, \alpha, \beta)$			$\mathfrak{ce}\big(\mathfrak{ce}(\mathfrak{B}, \beta), \alpha\big)$
$\mathfrak{ce}_3(\mathfrak{B}, \alpha, \beta, \gamma)$			$\mathfrak{ce}\big(\mathfrak{ce}_2(\mathfrak{B}, \beta, \gamma), \alpha\big)$

$\mathfrak{ce}_3(\mathfrak{B}, \alpha, \beta, \gamma)$. The machine copies down at the end first the symbols marked α, then those marked β, and finally those marked γ; it erases the symbols α, β, γ.

$\mathfrak{e}(\mathfrak{C})$	ə	R	$\mathfrak{e}_1(\mathfrak{C})$
	Not ə	L	$\mathfrak{e}(\mathfrak{C})$
$\mathfrak{e}_1(\mathfrak{C})$	Any	R, E, R	$\mathfrak{e}_1(\mathfrak{C})$
	None		$\mathfrak{C}$

From $\mathfrak{e}(\mathfrak{C})$ the marks are erased from all marked symbols. $\rightarrow \mathfrak{C}$.

5. *Enumeration of computable sequences.*

A computable sequence γ is determined by a description of a machine which computes γ. Thus the sequence 001011011101111... is determined by the table on p. 234, and, in fact, any computable sequence is capable of being described in terms of such a table.

It will be useful to put these tables into a kind of standard form. In the first place let us suppose that the table is given in the same form as the first table, for example, I on p. 233. That is to say, that the entry in the operations column is always of one of the forms $E : E, R : E, L : Pa : Pa, R : Pa, L : R : L :$ or no entry at all. The table can always be put into this form by introducing more m-configurations. Now let us give numbers to the m-configurations, calling them $q_1, \ldots, q_R$, as in §1. The initial m-configuration is always to be called q_1. We also give numbers to the symbols $S_1, \ldots, S_m$

and, in particular, blank $= S_0$, $0 = S_1$, $1 = S_2$. The lines of the table are now of form

m-config.	*Symbol*	*Operations*	*Final m-config.*	
q_i	S_j	PS_k, L	q_m	(N_1)
q_i	S_j	PS_k, R	q_m	(N_2)
q_i	S_j	PS_k	q_m	(N_3)

Lines such as

$$q_i \qquad S_j \qquad E, R \qquad q_m$$

are to be written as

$$q_i \qquad S_j \qquad PS_0, R \qquad q_m$$

and lines such as

$$q_i \qquad S_j \qquad R \qquad q_m$$

to be written as

$$q_i \qquad S_j \qquad PS_j, R \qquad q_m$$

In this way we reduce each line of the table to a line of one of the forms (N_1), (N_2), (N_3).

From each line of form (N_1) let us form an expression $q_i S_j S_k L q_m$; from each line of form (N_2) we form an expression $q_i S_j S_k R q_m$; and from each line of form (N_3) we form an expression $q_i S_j S_k N q_m$.

Let us write down all expressions so formed from the table for the machine and separate them by semi-colons. In this way we obtain a complete description of the machine. In this description we shall replace q_i by the letter "D" followed by the letter "A" repeated i times, and S_j by "D" followed by "C" repeated j times. This new description of the machine may be called the *standard description* (S.D). It is made up entirely from the letters "A", "C", "D", "L", "R", "N", and from ";".

If finally we replace "A" by "1", "C" by "2", "D" by "3", "L" by "4", "R" by "5", "N" by "6", and ";" by "7" we shall have a description of the machine in the form of an arabic numeral. The integer represented by this numeral may be called a *description number* (D.N) of the machine. The D.N determine the S.D and the structure of the

machine uniquely. The machine whose D.N is n may be described as $\mathcal{M}(n)$.

To each computable sequence there corresponds at least one description number, while to no description number does there correspond more than one computable sequence. The computable sequences and numbers are therefore enumerable.

Let us find a description number for the machine I of § 3. When we rename the m-configurations its table becomes:

q_1	S_0	PS_1, R	q_2
q_2	S_0	PS_0, R	q_3
q_3	S_0	PS_2, R	q_4
q_4	S_0	PS_0, R	q_1

Other tables could be obtained by adding irrelevant lines such as

$$q_1 \qquad S_1 \qquad PS_1, R \qquad q_2$$

Our first standard form would be

$$q_1 S_0 S_1 R q_2;\ q_2 S_0 S_0 R q_3;\ q_3 S_0 S_2 R q_4;\ q_4 S_0 S_0 R q_1;.$$

The standard description is

DADDCRDAA;*DAADDRDAAA*;

DAAADDCCRDAAAA;*DAAAADDRDA*;

A description number is

31332531173113353111731113322531111731111335317

and so is

3133253117311335311173111332253111173111133531731323253117

A number which is a description number of a circle-free machine will be called a *satisfactory* number. In § 8 it is shown that there can be no general process for determining whether a given number is satisfactory or not.

6. *The universal computing machine.*

It is possible to invent a single machine which can be used to compute any computable sequence. If this machine $\mathcal{U}$ is supplied with a tape on the beginning of which is written the S.D of some computing machine $\mathcal{M}$, then $\mathfrak{U}$ will compute the same sequence as $\mathfrak{M}$. In this section I explain in outline the behaviour of the machine. The next section is devoted to giving the complete table for $\mathfrak{U}$.

Let us first suppose that we have a machine $\mathfrak{M}'$ which will write down on the F-squares the successive complete configurations of $\mathfrak{M}$. These might be expressed in the same form as on p. 235, using the second description, (C), with all symbols on one line. Or, better, we could transform this description (as in § 5) by replacing each m-configuration by "D" followed by "A" repeated the appropriate number of times, and by replacing each symbol by "D" followed by "C" repeated the appropriate number of times. The numbers of letters "A" and "C" are to agree with the numbers chosen in § 5, so that, in particular, "0" is replaced by "DC", "1" by "DCC", and the blanks by "D". These substitutions are to be made after the complete configurations have been put together, as in (C). Difficulties arise if we do the substitution first. In each complete configuration the blanks would all have to be replaced by "D", so that the complete configuration would not be expressed as a finite sequence of symbols.

If in the description of the machine II of § 3 we replace "$\mathfrak{o}$" by "DAA", "ə" by "$DCCC$", "$\mathfrak{q}$" by "$DAAA$", then the sequence (C) becomes:

$$DA:DCCCDCCCDAADCDDC:DCCCDCCCDAAADCDDC:\ldots \quad (C_1)$$

(This is the sequence of symbols on F-squares.)

It is not difficult to see that if $\mathfrak{M}$ can be constructed, then so can $\mathfrak{M}'$. The manner of operation of $\mathfrak{M}'$ could be made to depend on having the rules of operation (*i.e.*, the S.D) of $\mathfrak{M}$ written somewhere within itself (*i.e.* within $\mathfrak{M}'$); each step could be carried out by referring to these rules. We have only to regard the rules as being capable of being taken out and exchanged for others and we have something very akin to the universal machine.

One thing is lacking: at present the machine $\mathfrak{M}'$ prints no figures. We may correct this by printing between each successive pair of complete configurations the figures which appear in the new configuration but not in the old. Then (C_1) becomes

$$DDA:0:0:DCCCDCCCDAADCDDC:DCCC.\ldots \quad (C_2)$$

It is not altogether obvious that the E-squares leave enough room for the necessary "rough work", but this is, in fact, the case.

The sequences of letters between the colons in expressions such as (C_1) may be used as standard descriptions of the complete configurations. When the letters are replaced by figures, as in § 5, we shall have a numerical description of the complete configuration, which may be called its description number.

7. *Detailed description of the universal machine.*

A table is given below of the behaviour of this universal machine. The m-configurations of which the machine is capable are all those occurring in the first and last columns of the table, together with all those which occur when we write out the unabbreviated tables of those which appear in the table in the form of m-functions. *E.g.*, $\mathfrak{e}(\mathfrak{anf})$ appears in the table and is an m-function. Its unabbreviated table is (see p. 239)

$\mathfrak{e}(\mathfrak{anf})$	ə	R	$\mathfrak{e}_1(\mathfrak{anf})$
	not ə	L	$\mathfrak{e}(\mathfrak{anf})$
$\mathfrak{e}_1(\mathfrak{anf})$	Any	R, E, R	$\mathfrak{e}_1(\mathfrak{anf})$
	None		$\mathfrak{anf}$

Consequently $\mathfrak{e}_1(\mathfrak{anf})$ is an m-configuration of $\mathfrak{U}$.

When $\mathfrak{U}$ is ready to start work the tape running through it bears on it the symbol ə on an F-square and again ə on the next E-square; after this, on F-squares only, comes the S.D of the machine followed by a double colon "::" (a single symbol, on an F-square). The S.D consists of a number of instructions, separated by semi-colons.

Each instruction consists of five consecutive parts

(i) "D" followed by a sequence of letters "A". This describes the relevant m-configuration.

(ii) "D" followed by a sequence of letters "C". This describes the scanned symbol.

(iii) "D" followed by another sequence of letters "C". This describes the symbol into which the scanned symbol is to be changed.

(iv) "L", "R", or "N", describing whether the machine is to move to left, right, or not at all.

(v) "D" followed by a sequence of letters "A". This describes the final m-configuration.

The machine $\mathfrak{U}$ is to be capable of printing "A", "C", "D", "0", "1", "u", "v", "w", "x", "y", "z". The S.D is formed from ";", "A", "C", "D", "L", "R", "N".

Subsidiary skeleton table.

$\mathfrak{con}(\mathfrak{C}, \alpha)$	Not A	R, R	$\mathfrak{con}(\mathfrak{C}, \alpha)$
	A	$L, P\alpha, R$	$\mathfrak{con}_1(\mathfrak{C}, \alpha)$
$\mathfrak{con}_1(\mathfrak{C}, \alpha)$	A	$R, P\alpha, R$	$\mathfrak{con}_1(\mathfrak{C}, \alpha)$
	D	$R, P\alpha, R$	$\mathfrak{con}_2(\mathfrak{C}, \alpha)$
$\mathfrak{con}_2(\mathfrak{C}, \alpha)$	C	$R, P\alpha, R$	$\mathfrak{con}_2(\mathfrak{C}, \alpha)$
	Not C	R, R	$\mathfrak{C}$

$\mathfrak{con}(\mathfrak{C}, \alpha)$. Starting from an F-square, S say, the sequence C of symbols describing a configuration closest on the right of S is marked out with letters α. $\rightarrow \mathfrak{C}$.

$\mathfrak{con}(\mathfrak{C}, \ )$. In the final configuration the machine is scanning the square which is four squares to the right of the last square of C. C is left unmarked.

The table for $\mathfrak{U}$.

$\mathfrak{b}$			$\mathfrak{f}(\mathfrak{b}_1, \mathfrak{b}_1, ::)$
$\mathfrak{b}_1$		$R, R, P:, R, R, PD, R, R, PA$	$\mathfrak{anf}$
$\mathfrak{anf}$			$\mathfrak{g}(\mathfrak{anf}_1, :)$
$\mathfrak{anf}_1$			$\mathfrak{con}(\mathfrak{fom}, y)$
$\mathfrak{fom}$	;	R, Pz, L	$\mathfrak{con}(\mathfrak{fmp}, x)$
	z	L, L	$\mathfrak{fom}$
	not z nor ;	L	$\mathfrak{fom}$
$\mathfrak{fmp}$			$\mathfrak{cpe}(\mathfrak{e}(\mathfrak{fom}, x, y), \mathfrak{sim}, x, y)$

$\mathfrak{b}$. The machine prints $:DA$ on the F-squares after $::$ $\rightarrow \mathfrak{anf}$.

$\mathfrak{anf}$. The machine marks the configuration in the last complete configuration with y. $\rightarrow \mathfrak{fom}$.

$\mathfrak{fom}$. The machine finds the last semi-colon not marked with z. It marks this semi-colon with z and the configuration following it with x.

$\mathfrak{fmp}$. The machine compares the sequences marked x and y. It erases all letters x and y. $\rightarrow \mathfrak{sim}$ if they are alike. Otherwise $\rightarrow \mathfrak{fom}$.

$\mathfrak{anf}$. Taking the long view, the last instruction relevant to the last configuration is found. It can be recognised afterwards as the instruction following the last semi-colon marked z. $\rightarrow \mathfrak{sim}$.

$\mathfrak{sim}$			$\mathfrak{f}'(\mathfrak{sim}_1, \mathfrak{sim}_1, z)$
$\mathfrak{sim}_1$			$\mathfrak{con}(\mathfrak{sim}_2, \ )$
$\mathfrak{sim}_2$	A		$\mathfrak{sim}_3$
	not A	R, Pu, R, R, R	$\mathfrak{sim}_2$
$\mathfrak{sim}_3$	not A	L, Py	$\mathfrak{e}(\mathfrak{mk}, z)$
	A	L, Py, R, R, R	$\mathfrak{sim}_3$
$\mathfrak{mk}$			$\mathfrak{g}(\mathfrak{mk}, :)$
$\mathfrak{mk}_1$	not A	R, R	$\mathfrak{mk}_1$
	A	L, L, L, L	$\mathfrak{mk}_2$
$\mathfrak{mk}_2$	C	R, Px, L, L, L	$\mathfrak{mk}_2$
	:		$\mathfrak{mk}_4$
	D	R, Px, L, L, L	$\mathfrak{mk}_3$
$\mathfrak{mk}_3$	not :	R, Pv, L, L, L	$\mathfrak{mk}_3$
	:		$\mathfrak{mk}_4$
$\mathfrak{mk}_4$			$\mathfrak{con}\big(\mathfrak{l}(\mathfrak{l}(\mathfrak{mk}_5)), \ \big)$
$\mathfrak{mk}_5$	Any	R, Pw, R	$\mathfrak{mk}_5$
	None	$P:$	$\mathfrak{sh}$
$\mathfrak{sh}$			$\mathfrak{f}(\mathfrak{sh}_1, \mathfrak{inst}, u)$
$\mathfrak{sh}_1$		L, L, L	$\mathfrak{sh}_2$
$\mathfrak{sh}_2$	D	R, R, R, R	$\mathfrak{sh}_2$
	not D		$\mathfrak{inst}$
$\mathfrak{sh}_3$	C	R, R	$\mathfrak{sh}_4$
	not C		$\mathfrak{inst}$
$\mathfrak{sh}_4$	C	R, R	$\mathfrak{sh}_5$
	not C		$\mathfrak{pe}_2(\mathfrak{inst}, 0, :)$
$\mathfrak{sh}_5$	C		$\mathfrak{inst}$
	not C		$\mathfrak{pe}_2(\mathfrak{inst}, 1, :)$

$\mathfrak{sim}$. The machine marks out the instructions. That part of the instructions which refers to operations to be carried out is marked with u, and the final m-configuration with y. The letters z are erased.

$\mathfrak{mk}$. The last complete configuration is marked out into four sections. The configuration is left unmarked. The symbol directly preceding it is marked with x. The remainder of the complete configuration is divided into two parts, of which the first is marked with v and the last with w. A colon is printed after the whole. $\rightarrow \mathfrak{sh}$.

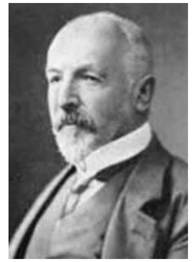
$\mathfrak{sh}$. The instructions (marked u) are examined. If it is found that they involve "Print 0" or "Print 1", then $0:$ or $1:$ is printed at the end.

Universal Turing machine!

Basis for the "stored program" concept!

$\mathfrak{inst}$			$\mathfrak{g}(\mathfrak{l}(\mathfrak{inst}_1), u)$
$\mathfrak{inst}_1$	α	R, E	$\mathfrak{inst}_1(\alpha)$
$\mathfrak{inst}_1(L)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, y, x, u, w)$
$\mathfrak{inst}_1(R)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, x, u, y, w)$
$\mathfrak{inst}_1(N)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, x, y, u, w)$
$\mathfrak{ov}$			$\mathfrak{e}(\mathfrak{anf})$

$\mathfrak{inst}$. The next complete configuration is written down, carrying out the marked instructions. The letters u, v, w, x, y are erased. $\rightarrow \mathfrak{anf}$.



8. *Application of the diagonal process.*

It may be thought that arguments which prove that the real numbers are not enumerable would also prove that the computable numbers and sequences cannot be enumerable*. It might, for instance, be thought that the limit of a sequence of computable numbers must be computable. This is clearly only true if the sequence of computable numbers is defined by some rule.

Or we might apply the diagonal process. "If the computable sequences are enumerable, let α_n be the n-th computable sequence, and let $\phi_n(m)$ be the m-th figure in α_n. Let β be the sequence with $1-\phi_n(n)$ as its n-th figure. Since β is computable, there exists a number K such that $1-\phi_n(n) = \phi_K(n)$ all n. Putting $n = K$, we have $1 = 2\phi_K(K)$, *i.e.* 1 is even. This is impossible. The computable sequences are therefore not enumerable".

The fallacy in this argument lies in the assumption that β is computable. It would be true if we could enumerate the computable sequences by finite means, but the problem of enumerating computable sequences is equivalent to the problem of finding out whether a given number is the D.N of a circle-free machine, and we have no general process for doing this in a finite number of steps. In fact, by applying the diagonal process argument correctly, we can show that there cannot be any such general process.

The simplest and most direct proof of this is by showing that, if this general process exists, then there is a machine which computes β. This proof, although perfectly sound, has the disadvantage that it may leave the reader with a feeling that "there must be something wrong". The proof which I shall give has not this disadvantage, and gives a certain insight into the significance of the idea "circle-free". It depends not on constructing β, but on constructing β', whose n-th figure is $\phi_n(n)$.

* *Cf.* Hobson, *Theory of functions of a real variable* (2nd ed., 1921), 87, 88.

Let us suppose that there is such a process; that is to say, that we can invent a machine $\mathfrak{D}$ which, when supplied with the S.D of any computing machine $\mathcal{M}$ will test this S.D and if $\mathcal{M}$ is circular will mark the S.D with the symbol "u" and if it is circle-free will mark it with "s". By combining the machines $\mathfrak{D}$ and $\mathfrak{U}$ we could construct a machine $\mathfrak{H}$ to compute the sequence β'. The machine $\mathfrak{D}$ may require a tape. We may suppose that it uses the E-squares beyond all symbols on F-squares, and that when it has reached its verdict all the rough work done by $\mathfrak{D}$ is erased.

The machine $\mathfrak{H}$ has its motion divided into sections. In the first $N-1$ sections, among other things, the integers $1, 2, \ldots, N-1$ have been written down and tested by the machine $\mathfrak{D}$. A certain number, say $R(N-1)$, of them have been found to be the D.N's of circle-free machines. In the N-th section the machine $\mathfrak{D}$ tests the number N. If N is satisfactory, *i.e.*, if it is the D.N of a circle-free machine, then $R(N) = 1 + R(N-1)$ and the first $R(N)$ figures of the sequence of which a D.N is N are calculated. The $R(N)$-th figure of this sequence is written down as one of the figures of the sequence β' computed by $\mathfrak{H}$. If N is not satisfactory, then $R(N) = R(N-1)$ and the machine goes on to the $(N+1)$-th section of its motion.

From the construction of $\mathfrak{H}$ we can see that $\mathfrak{H}$ is circle-free. Each section of the motion of $\mathfrak{H}$ comes to an end after a finite number of steps. For, by our assumption about $\mathfrak{D}$, the decision as to whether N is satisfactory is reached in a finite number of steps. If N is not satisfactory, then the N-th section is finished. If N is satisfactory, this means that the machine $\mathcal{M}(N)$ whose D.N is N is circle-free, and therefore its $R(N)$-th figure can be calculated in a finite number of steps. When this figure has been calculated and written down as the $R(N)$-th figure of β', the N-th section is finished. Hence $\mathfrak{H}$ is circle-free.

Now let K be the D.N of $\mathfrak{H}$. What does $\mathfrak{H}$ do in the K-th section of its motion? It must test whether K is satisfactory, giving a verdict "s" or "u". Since K is the D.N of $\mathfrak{H}$ and since $\mathfrak{H}$ is circle-free, the verdict cannot be "u". On the other hand the verdict cannot be "s". For if it were, then in the K-th section of its motion $\mathfrak{H}$ would be bound to compute the first $R(K-1)+1 = R(K)$ figures of the sequence computed by the machine with K as its D.N and to write down the $R(K)$-th as a figure of the sequence computed by $\mathfrak{H}$. The computation of the first $R(K)-1$ figures would be carried out all right, but the instructions for calculating the $R(K)$-th would amount to "calculate the first $R(K)$ figures computed by H and write down the $R(K)$-th". This $R(K)$-th figure would never be found. *I.e.*, $\mathfrak{H}$ is circular, contrary both to what we have found in the last paragraph and to the verdict "s". Thus both verdicts are impossible and we conclude that there can be no machine $\mathfrak{D}$.

We can show further that *there can be no machine* $\mathcal{E}$ *which, when supplied with the S.D of an arbitrary machine* $\mathcal{M}$, *will determine whether* $\mathcal{M}$ *ever prints a given symbol* (0 *say*).

We will first show that, if there is a machine $\mathcal{E}$, then there is a general process for determining whether a given machine $\mathcal{M}$ prints 0 infinitely often. Let $\mathcal{M}_1$ be a machine which prints the same sequence as $\mathcal{M}$, except that in the position where the first 0 printed by $\mathcal{M}$ stands, $\mathcal{M}_1$ prints $\bar{0}$. $\mathcal{M}_2$ is to have the first two symbols 0 replaced by $\bar{0}$, and so on. Thus, if $\mathcal{M}$ were to print

$$A\,B\,A\,0\,1\,A\,A\,B\,0\,0\,1\,0\,A\,B\ldots,$$

then $\mathcal{M}_1$ would print

$$A\,B\,A\,\bar{0}\,1\,A\,A\,B\,0\,0\,1\,0\,A\,B\ldots$$

and $\mathcal{M}_2$ would print

$$A\,B\,A\,\bar{0}\,1\,A\,A\,B\,\bar{0}\,0\,1\,0\,A\,B\ldots.$$

Now let $\mathcal{F}$ be a machine which, when supplied with the S.D of $\mathcal{M}$, will write down successively the S.D of $\mathcal{M}$, of $\mathcal{M}_1$, of $\mathcal{M}_2$, ... (there is such a machine). We combine $\mathcal{F}$ with $\mathcal{E}$ and obtain a new machine, $\mathcal{G}$. In the motion of $\mathcal{G}$ first $\mathcal{F}$ is used to write down the S.D of $\mathcal{M}$, and then $\mathcal{E}$ tests it, : 0 : is written if it is found that $\mathcal{M}$ never prints 0; then $\mathcal{F}$ writes the S.D of $\mathcal{M}_1$, and this is tested, : 0 : being printed if and only if $\mathcal{M}_1$ never prints 0, and so on. Now let us test $\mathcal{G}$ with $\mathcal{E}$. If it is found that $\mathcal{G}$ never prints 0, then $\mathcal{M}$ prints 0 infinitely often; if $\mathcal{G}$ prints 0 sometimes, then $\mathcal{M}$ does not print 0 infinitely often.

Similarly there is a general process for determining whether $\mathcal{M}$ prints 1 infinitely often. By a combination of these processes we have a process for determining whether $\mathcal{M}$ prints an infinity of figures, *i.e.* we have a process for determining whether $\mathcal{M}$ is circle-free. There can therefore be no machine $\mathcal{E}$.

The expression "there is a general process for determining ..." has been used throughout this section as equivalent to "there is a machine which will determine ...". This usage can be justified if and only if we can justify our definition of "computable". For each of these "general process" problems can be expressed as a problem concerning a general process for determining whether a given integer n has a property $G(n)$ [*e.g.* $G(n)$ might mean "n is satisfactory" or "n is the Gödel representation of a provable formula"], and this is equivalent to computing a number whose n-th figure is 1 if $G(n)$ is true and 0 if it is false.

Church-Turing thesis!

9. *The extent of the computable numbers.*

No attempt has yet been made to show that the "computable" numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. The real question at issue is "What are the possible processes which can be carried out in computing a number?"

The arguments which I shall use are of three kinds.

(*a*) A direct appeal to intuition.

(*b*) A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal).

(*c*) Giving examples of large classes of numbers which are computable.

Once it is granted that computable numbers are all "computable", several other propositions of the same character follow. In particular, it follows that, if there is a general process for determining whether a formula of the Hilbert function calculus is provable, then the determination can be carried out by a machine.

I. [Type (*a*)]. This argument is only an elaboration of the ideas of § 1.

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, *i.e.* on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent†. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. Thus an Arabic numeral such as

† If we regard a symbol as literally printed on a square we may suppose that the square is $0 \leqslant x \leqslant 1$, $0 \leqslant y \leqslant 1$. The symbol is defined as a set of points in this square, viz. the set occupied by printer's ink. If these sets are restricted to be measurable, we can define the "distance" between two symbols as the cost of transforming one symbol into the other if the cost of moving unit area of printer's ink unit distance is unity, and there is an infinite supply of ink at $x = 2$, $y = 0$. With this topology the symbols form a conditionally compact space.

17 or 999999999999999 is normally treated as a single symbol. Similarly in any European language words are treated as single symbols (Chinese, however, attempts to have an enumerable infinity of symbols). The differences from our point of view between the single and compound symbols is that the compound symbols, if they are too lengthy, cannot be observed at one glance. This is in accordance with experience. We cannot tell at a glance whether 9999999999999999 and 999999999999999 are the same.

The behaviour of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will be "arbitrarily close" and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.

Let us imagine the operations performed by the computer to be split up into "simple operations" which are so elementary that it is not easy to imagine them further divided. Every such operation consists of some change of the physical system consisting of the computer and his tape. We know the state of the system if we know the sequence of symbols on the tape, which of these are observed by the computer (possibly with a special order), and the state of mind of the computer. We may suppose that in a simple operation not more than one symbol is altered. Any other changes can be split up into simple changes of this kind. The situation in regard to the squares whose symbols may be altered in this way is the same as in regard to the observed squares. We may, therefore, without loss of generality, assume that the squares whose symbols are changed are always "observed" squares.

Besides these changes of symbols, the simple operations must include changes of distribution of observed squares. The new observed squares must be immediately recognisable by the computer. I think it is reasonable to suppose that they can only be squares whose distance from the closest of the immediately previously observed squares does not exceed a certain fixed amount. Let us say that each of the new observed squares is within L squares of an immediately previously observed square.

In connection with "immediate recognisability", it may be thought that there are other kinds of square which are immediately recognisable. In particular, squares marked by special symbols might be taken as immediately recognisable. Now if these squares are marked only by single symbols there can be only a finite number of them, and we should not upset our theory by adjoining these marked squares to the observed squares. If, on the other hand, they are marked by a sequence of symbols, we cannot regard the process of recognition as a simple process. This is a fundamental point and should be illustrated. In most mathematical papers the equations and theorems are numbered. Normally the numbers do not go beyond (say) 1000. It is, therefore, possible to recognise a theorem at a glance by its number. But if the paper was very long, we might reach Theorem 157767733443477; then, further on in the paper, we might find "... hence (applying Theorem 157767733443477) we have ...". In order to make sure which was the relevant theorem we should have to compare the two numbers figure by figure, possibly ticking the figures off in pencil to make sure of their not being counted twice. If in spite of this it is still thought that there are other "immediately recognisable" squares, it does not upset my contention so long as these squares can be found by some process of which my type of machine is capable. This idea is developed in III below.

The simple operations must therefore include:

(*a*) Changes of the symbol on one of the observed squares.

(*b*) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.

It may be that some of these changes necessarily involve a change of state of mind. The most general single operation must therefore be taken to be one of the following:

(*A*) A possible change (*a*) of symbol together with a possible change of state of mind.

(*B*) A possible change (*b*) of observed squares, together with a possible change of state of mind.

The operation actually performed is determined, as has been suggested on p. 250, by the state of mind of the computer and the observed symbols. In particular, they determine the state of mind of the computer after the operation is carried out.

We may now construct a machine to do the work of this computer. To each state of mind of the computer corresponds an "m-configuration" of the machine. The machine scans B squares corresponding to the B squares observed by the computer. In any move the machine can change a symbol on a scanned square or can change any one of the scanned squares to another square distant not more than L squares from one of the other scanned squares. The move which is done, and the succeeding configuration, are determined by the scanned symbol and the m-configuration. The machines just described do not differ very essentially from computing machines as defined in § 2, and corresponding to any machine of this type a computing machine can be constructed to compute the same sequence, that is to say the sequence computed by the computer.

II. [Type (b)].

If the notation of the Hilbert functional calculus† is modified so as to be systematic, and so as to involve only a finite number of symbols, it becomes possible to construct an automatic‡ machine $\mathfrak{K}$, which will find all the provable formulae of the calculus§.

Now let α be a sequence, and let us denote by $G_\alpha(x)$ the proposition "The x-th figure of α is 1", so that|| $-G_\alpha(x)$ means "The x-th figure of α is 0". Suppose further that we can find a set of properties which define the sequence α and which can be expressed in terms of $G_\alpha(x)$ and of the propositional functions $N(x)$ meaning "x is a non-negative integer" and $F(x, y)$ meaning "$y = x+1$". When we join all these formulae together conjunctively, we shall have a formula, $\mathfrak{A}$ say, which defines α. The terms of $\mathfrak{A}$ must include the necessary parts of the Peano axioms, viz.,

$$(\exists u)N(u) \,\&\, (x)\Big(N(x) \rightarrow (\exists y)F(x, y)\Big) \,\&\, \Big(F(x, y) \rightarrow N(y)\Big),$$

which we will abbreviate to P.

When we say "$\mathfrak{A}$ defines α", we mean that $-\mathfrak{A}$ is not a provable formula, and also that, for each n, one of the following formulae (A_n) or (B_n) is provable.

$$\mathfrak{A} \,\&\, F^{(n)} \rightarrow G_\alpha(u^{(n)}), \qquad (A_n)¶$$

$$\mathfrak{A} \,\&\, F^{(n)} \rightarrow \Big(-G_\alpha(u^{(n)})\Big), \qquad (B_n),$$

where $F^{(n)}$ stands for $F(u, u') \,\&\, F(u', u'') \,\&\, \ldots F(u^{(n-1)}, u^{(n)})$.

† The expression "the functional calculus" is used throughout to mean the *restricted* Hilbert functional calculus.

‡ It is most natural to construct first a choice machine (§ 2) to do this. But it is then easy to construct the required automatic machine. We can suppose that the choices are always choices between two possibilities 0 and 1. Each proof will then be determined by a sequence of choices $i_1, i_2, \ldots, i_n$ ($i_1 = 0$ or 1, $i_2 = 0$ or 1, ..., $i_n = 0$ or 1), and hence the number $2^n + i_1 2^{n-1} + i_2 2^{n-2} + \ldots + i_n$ completely determines the proof. The automatic machine carries out successively proof 1, proof 2, proof 3, ...

§ The author has found a description of such a machine.

|| The negation sign is written before an expression and not over it.

¶ A sequence of r primes is denoted by $^{(r)}$.

I say that α is then a computable sequence: a machine $\mathfrak{K}_\alpha$ to compute α can be obtained by a fairly simple modification of $\mathfrak{K}$.

We divide the motion of $\mathfrak{K}_\alpha$ into sections. The n-th section is devoted to finding the n-th figure of α. After the $(n-1)$-th section is finished a double colon :: is printed after all the symbols, and the succeeding work is done wholly on the squares to the right of this double colon. The first step is to write the letter "A" followed by the formula (A_n) and then "B" followed by (B_n). The machine $\mathfrak{K}_\alpha$ then starts to do the work of $\mathfrak{K}$, but whenever a provable formula is found, this formula is compared with (A_n) and with (B_n). If it is the same formula as (A_n), then the figure "1" is printed, and the n-th section is finished. If it is (B_n), then "0" is printed and the section is finished. If it is different from both, then the work of $\mathfrak{K}$ is continued from the point at which it had been abandoned. Sooner or later one of the formulae (A_n) or (B_n) is reached; this follows from our hypotheses about α and $\mathfrak{A}$, and the known nature of $\mathfrak{K}$. Hence the n-th section will eventually be finished. $\mathfrak{K}_\alpha$ is circle-free; α is computable.

It can also be shown that the numbers α definable in this way by the use of axioms include all the computable numbers. This is done by describing computing machines in terms of the function calculus.

It must be remembered that we have attached rather a special meaning to the phrase "$\mathfrak{A}$ defines α". The computable numbers do not include all (in the ordinary sense) definable numbers. Let δ be a sequence whose n-th figure is 1 or 0 according as n is or is not satisfactory. It is an immediate consequence of the theorem of § 8 that δ is not computable. It is (so far as we know at present) possible that any assigned number of figures of δ can be calculated, but not by a uniform process. When sufficiently many figures of δ have been calculated, an essentially new method is necessary in order to obtain more figures.

III. This may be regarded as a modification of I or as a corollary of II.

We suppose, as in I, that the computation is carried out on a tape; but we avoid introducing the "state of mind" by considering a more physical and definite counterpart of it. It is always possible for the computer to break off from his work, to go away and forget all about it, and later to come back and go on with it. If he does this he must leave a note of instructions (written in some standard form) explaining how the work is to be continued. This note is the counterpart of the "state of mind". We will suppose that the computer works in such a desultory manner that he never does more than one step at a sitting. The note of instructions must enable him to carry out one step and write the next note. Thus the state of progress of the computation at any stage is completely determined by the note of instructions and the symbols on the tape. That is, the state of the system may be described by a single expression (sequence of symbols), consisting of the symbols on the tape followed by Δ (which we suppose not to appear elsewhere) and then by the note of instructions. This expression may be called the "state formula". We know that the state formula at any given stage is determined by the state formula before the last step was made, and we assume that the relation of these two formulae is expressible in the functional calculus. In other words, we assume that there is an axiom $\mathfrak{A}$ which expresses the rules governing the behaviour of the computer, in terms of the relation of the state formula at any stage to the state formula at the preceding stage. If this is so, we can construct a machine to write down the successive state formulae, and hence to compute the required number.

10. *Examples of large classes of numbers which are computable.*

It will be useful to begin with definitions of a computable function of an integral variable and of a computable variable, etc. There are many equivalent ways of defining a computable function of an integral variable. The simplest is, possibly, as follows. If γ is a computable sequence in which 0 appears infinitely† often, and n is an integer, then let us define $\xi(\gamma, n)$ to be the number of figures 1 between the n-th and the $(n+1)$-th figure 0 in γ. Then $\phi(n)$ is computable if, for all n and some γ, $\phi(n) = \xi(\gamma, n)$. An equivalent definition is this. Let $H(x, y)$ mean $\phi(x) = y$. Then, if we can find a contradiction-free axiom $\mathfrak{A}_\phi$, such that $\mathfrak{A}_\phi \to P$, and if for each integer n there exists an integer N, such that

$$\mathfrak{A}_\phi \;\&\; F^{(N)} \to H(u^{(n)}, u^{(\phi(n))}),$$

and such that, if $m \neq \phi(n)$, then, for some N',

$$\mathfrak{A}_\phi \;\&\; F^{(N')} \to \big(-H(u^{(n)}, u^{(m)})\big),$$

then ϕ may be said to be a computable function.

We cannot define general computable functions of a real variable, since there is no general method of describing a real number, but we can define a computable function of a computable variable. If n is satisfactory, let γ_n be the number computed by $\mathcal{M}(n)$, and let

$$\alpha_n = \tan\big(\pi(\gamma_n - \tfrac{1}{2})\big),$$

† If $\mathcal{M}$ computes γ, then the problem whether $\mathcal{M}$ prints 0 infinitely often is of the same character as the problem whether $\mathcal{M}$ is circle-free.

unless $\gamma_n = 0$ or $\gamma_n = 1$, in either of which cases $\alpha_n = 0$. Then, as n runs through the satisfactory numbers, α_n runs through the computable numbers†. Now let $\phi(n)$ be a computable function which can be shown to be such that for any satisfactory argument its value is satisfactory‡. Then the function f, defined by $f(\alpha_n) = \alpha_{\phi(n)}$, is a computable function and all computable functions of a computable variable are expressible in this form.

Similar definitions may be given of computable functions of several variables, computable-valued functions of an integral variable, etc.

I shall enunciate a number of theorems about computability, but I shall prove only (ii) and a theorem similar to (iii).

Subroutines!

(i) A computable function of a computable function of an integral or computable variable is computable.

(ii) Any function of an integral variable defined recursively in terms of computable functions is computable. *I.e.* if $\phi(m, n)$ is computable, and r is some integer, then $\eta(n)$ is computable, where

$$\eta(0) = r,$$

$$\eta(n) = \phi\big(n, \eta(n-1)\big).$$

(iii) If $\phi(m, n)$ is a computable function of two integral variables, then $\phi(n, n)$ is a computable function of n.

(iv) If $\phi(n)$ is a computable function whose value is always 0 or 1, then the sequence whose n-th figure is $\phi(n)$ is computable.

Dedekind's theorem does not hold in the ordinary form if we replace "real" throughout by "computable". But it holds in the following form:

(v) If $G(\alpha)$ is a propositional function of the computable numbers and

$$(a) \quad (\exists\alpha)(\exists\beta)\{G(\alpha) \;\&\; \big(-G(\beta)\big)\},$$

$$(b) \quad G(\alpha) \;\&\; \big(-G(\beta)\big) \to (\alpha < \beta),$$

and there is a general process for determining the truth value of $G(\alpha)$, then

† A function α_n may be defined in many other ways so as to run through the computable numbers.

‡ Although it is not possible to find a general process for determining whether a given number is satisfactory, it is often possible to show that certain classes of numbers are satisfactory.

there is a computable number ξ such that

$$G(\alpha) \to \alpha \leqslant \xi,$$

$$-G(\alpha) \to \alpha \geqslant \xi.$$

In other words, the theorem holds for any section of the computables such that there is a general process for determining to which class a given number belongs.

Owing to this restriction of Dedekind's theorem, we cannot say that a computable bounded increasing sequence of computable numbers has a computable limit. This may possibly be understood by considering a sequence such as

$$-1, -\tfrac{1}{2}, -\tfrac{1}{4}, -\tfrac{1}{8}, -\tfrac{1}{16}, \tfrac{1}{2}, \ldots.$$

On the other hand, (v) enables us to prove

(vi) If α and β are computable and $\alpha < \beta$ and $\phi(\alpha) < 0 < \phi(\beta)$, where $\phi(\alpha)$ is a computable increasing continuous function, then there is a unique computable number γ, satisfying $\alpha < \gamma < \beta$ and $\phi(\gamma) = 0$.

Computable convergence.

We shall say that a sequence β_n of computable numbers *converges computably* if there is a computable integral valued function $N(\epsilon)$ of the computable variable ϵ, such that we can show that, if $\epsilon > 0$ and $n > N(\epsilon)$ and $m > N(\epsilon)$, then $|\beta_n - \beta_m| < \epsilon$.

We can then show that

(vii) A power series whose coefficients form a computable sequence of computable numbers is computably convergent at all computable points in the interior of its interval of convergence.

(viii) The limit of a computably convergent sequence is computable.

And with the obvious definition of "uniformly computably convergent":

(ix) The limit of a uniformly computably convergent computable sequence of computable functions is a computable function. Hence

(x) The sum of a power series whose coefficients form a computable sequence is a computable function in the interior of its interval of convergence.

From (viii) and $\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \ldots)$ we deduce that π is computable.

From $e = 1 + 1 + \dfrac{1}{2!} + \dfrac{1}{3!} + \ldots$ we deduce that e is computable.

From (vi) we deduce that all real algebraic numbers are computable.

From (vi) and (x) we deduce that the real zeros of the Bessel functions are computable.

Proof of (ii).

Let $H(x, y)$ mean "$\eta(x) = y$", and let $K(x, y, z)$ mean "$\phi(x, y) = z$". $\mathfrak{A}_\phi$ is the axiom for $\phi(x, y)$. We take $\mathfrak{A}_\eta$ to be

$$\mathfrak{A}_\phi \mathbin{\&} P \mathbin{\&} \big(F(x, y) \to G(x, y)\big) \mathbin{\&} \big(G(x, y) \mathbin{\&} G(y, z) \to G(x, z)\big)$$

$$\mathbin{\&} \big(F^{(r)} \to H(u, u^{(r)})\big) \mathbin{\&} \big(F(v, w) \mathbin{\&} H(v, x) \mathbin{\&} K(w, x, z) \to H(w, z)\big)$$

$$\mathbin{\&} \Big[H(w, z) \mathbin{\&} G(z, t) \vee G(t, z) \to \big(-H(w, t)\big)\Big].$$

I shall not give the proof of consistency of $\mathfrak{A}_\eta$. Such a proof may be constructed by the methods used in Hilbert and Bernays, *Grundlagen der Mathematik* (Berlin, 1934), p. 209 *et seq*. The consistency is also clear from the meaning.

Suppose that, for some n, N, we have shown

$$\mathfrak{A}_\eta \mathbin{\&} F^{(N)} \to H(u^{(n-1)}, u^{(\eta(n-1))}),$$

then, for some M,

$$\mathfrak{A}_\phi \mathbin{\&} F^{(M)} \to K(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}),$$

$$\mathfrak{A}_\eta \mathbin{\&} F^{(M)} \to F(u^{(n-1)}, u^{(n)}) \mathbin{\&} H(u^{(n-1)}, u^{(\eta(n-1))})$$

$$\mathbin{\&} K(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}),$$

and

$$\mathfrak{A}_\eta \mathbin{\&} F^{(M)} \to [F(u^{(n-1)}, u^{(n)}) \mathbin{\&} H(u^{(n-1)}, u^{(\eta(n-1))})$$

$$\mathbin{\&} K(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}) \to H(u^{(n)}, u^{(\eta(n))})].$$

Hence $$\mathfrak{A}_\eta \mathbin{\&} F^{(M)} \to H(u^{(n)}, u^{(\eta(n))}).$$

Also $$\mathfrak{A}_\eta \mathbin{\&} F^{(r)} \to H(u, u^{(\eta(0))}).$$

Hence for each n some formula of the form

$$\mathfrak{A}_\eta \mathbin{\&} F^{(M)} \to H(u^{(n)}, u^{(\eta(n))})$$

is provable. Also, if $M' \geqslant M$ and $M' \geqslant m$ and $m \neq \eta(u)$, then

$$\mathfrak{A}_\eta \mathbin{\&} F^{(M')} \to G(u^{\eta((n))}, u^{(m)}) \vee G(u^{(m)}, u^{(\eta(n))})$$

and

$$\mathfrak{A}_\eta \,\&\, F^{(M')} \to \Big[\{G(u^{(\eta(n))}, u^{(m)}) \vee G(u^{(m)}, u^{(\eta(n))}) \,\&\, H(u^{(n)}, u^{(\eta(n))})\} \to \big(-H(u^{(n)}, u^{(m)})\big) \Big].$$

Hence $$\mathfrak{A}_\eta \,\&\, F^{(M')} \to \big(-H(u^{(n)}, u^{(m)})\big).$$

The conditions of our second definition of a computable function are therefore satisfied. Consequently η is a computable function.

Proof of a modified form of (iii).

Suppose that we are given a machine $\mathfrak{N}$, which, starting with a tape bearing on it əə followed by a sequence of any number of letters "F" on F-squares and in the m-configuration $\mathfrak{b}$, will compute a sequence γ_n depending on the number n of letters "F". If $\phi_n(m)$ is the m-th figure of γ_n, then the sequence β whose n-th figure is $\phi_n(n)$ is computable.

We suppose that the table for $\mathfrak{N}$ has been written out in such a way that in each line only one operation appears in the operations column. We also suppose that Ξ, Θ, $\bar{0}$, and $\bar{1}$ do not occur in the table, and we replace ə throughout by Θ, 0 by $\bar{0}$, and 1 by $\bar{1}$. Further substitutions are then made. Any line of form

$\mathfrak{A}$	α	$P\bar{0}$	$\mathfrak{B}$

we replace by

$\mathfrak{A}$	α	$P\bar{0}$	$\mathfrak{re}(\mathfrak{B}, \mathfrak{u}, h, k)$

and any line of the form

$\mathfrak{A}$	α	$P\bar{1}$	$\mathfrak{B}$

by

$\mathfrak{A}$	α	$P\bar{1}$	$\mathfrak{re}(\mathfrak{B}, \mathfrak{v}, h, k)$

and we add to the table the following lines:

$\mathfrak{u}$		$\mathfrak{pe}(\mathfrak{u}_1, 0)$
$\mathfrak{u}_1$	$R, Pk, R, P\Theta, R, P\Theta$	$\mathfrak{u}_2$
$\mathfrak{u}_2$		$\mathfrak{re}(\mathfrak{u}_3, \mathfrak{u}_3, k, h)$
$\mathfrak{u}_3$		$\mathfrak{pe}(\mathfrak{u}_2, F)$

and similar lines with $\mathfrak{v}$ for $\mathfrak{u}$ and 1 for 0 together with the following line

$\mathfrak{c}$	$R, P\Xi, R, Ph$	$\mathfrak{b}$.

We then have the table for the machine $\mathfrak{N}'$ which computes β. The initial m-configuration is $\mathfrak{c}$, and the initial scanned symbol is the second ə.

11. *Application to the Entscheidungsproblem.*

The results of §8 have some important applications. In particular, they can be used to show that the Hilbert Entscheidungsproblem can have no solution. For the present I shall confine myself to proving this particular theorem. For the formulation of this problem I must refer the reader to Hilbert and Ackermann's *Grundzüge der Theoretischen Logik* (Berlin, 1931), chapter 3.

I propose, therefore, to show that there can be no general process for determining whether a given formula $\mathfrak{A}$ of the functional calculus **K** is provable, *i.e.* that there can be no machine which, supplied with any one $\mathfrak{A}$ of these formulae, will eventually say whether $\mathfrak{A}$ is provable.

It should perhaps be remarked that what I shall prove is quite different from the well-known results of Gödel†. Gödel has shown that (in the formalism of Principia Mathematica) there are propositions $\mathfrak{A}$ such that neither $\mathfrak{A}$ nor $-\mathfrak{A}$ is provable. As a consequence of this, it is shown that no proof of consistency of Principia Mathematica (or of **K**) can be given within that formalism. On the other hand, I shall show that there is no general method which tells whether a given formula $\mathfrak{A}$ is provable in **K**, or, what comes to the same, whether the system consisting of **K** with $-\mathfrak{A}$ adjoined as an extra axiom is consistent.

If the negation of what Gödel has shown had been proved, *i.e.* if, for each $\mathfrak{A}$, either $\mathfrak{A}$ or $-\mathfrak{A}$ is provable, then we should have an immediate solution of the Entscheidungsproblem. For we can invent a machine $\mathcal{K}$ which will prove consecutively all provable formulae. Sooner or later $\mathcal{K}$ will reach either $\mathfrak{A}$ or $-\mathfrak{A}$. If it reaches $\mathfrak{A}$, then we know that $\mathfrak{A}$ is provable. If it reaches $-\mathfrak{A}$, then, since **K** is consistent (Hilbert and Ackermann, p. 65), we know that $\mathfrak{A}$ is not provable.

Owing to the absence of integers in **K** the proofs appear somewhat lengthy. The underlying ideas are quite straightforward.

Corresponding to each computing machine $\mathcal{M}$ we construct a formula Un $(\mathcal{M})$ and we show that, if there is a general method for determining whether Un $(\mathcal{M})$ is provable, then there is a general method for determining whether $\mathcal{M}$ ever prints 0.

The interpretations of the propositional functions involved are as follows:

$R_{S_l}(x, y)$ is to be interpreted as "in the complete configuration x (of $\mathcal{M}$) the symbol on the square y is S".

† *Loc. cit.*

$I(x, y)$ is to be interpreted as "in the complete configuration x the square y is scanned".

$K_{q_m}(x)$ is to be interpreted as "in the complete configuration x the m-configuration is q_m.

$F(x, y)$ is to be interpreted as "y is the immediate successor of x".

Inst $\{q_i S_j S_k L q_l\}$ is to be an abbreviation for

$$(x, y, x', y')\Big\{\Big(R_{S_j}(x, y) \,\&\, I(x, y) \,\&\, K_{q_i}(x) \,\&\, F(x, x') \,\&\, F(y', y)\Big)$$
$$\to \Big(I(x', y') \,\&\, R_{S_k}(x', y) \,\&\, K_{q_l}(x')$$
$$\&\, (z)\Big[F(y', z) \vee \Big(R_{S_j}(x, z) \to R_{S_k}(x', z)\Big)\Big]\Big)\Big\}.$$

$$\text{Inst}\,\{q_i S_j S_k R q_l\} \quad \text{and} \quad \text{Inst}\,\{q_i S_j S_k N q_l\}$$

are to be abbreviations for other similarly constructed expressions.

Let us put the description of $\mathfrak{M}$ into the first standard form of § 6. This description consists of a number of expressions such as "$q_i S_j S_k L q_l$" (or with R or N substituted for L). Let us form all the corresponding expressions such as Inst $\{q_i S_j S_k L q_l\}$ and take their logical sum. This we call Des $(\mathfrak{M})$.

The formula Un $(\mathfrak{M})$ is to be

$$(\exists u)\Big[N(u) \,\&\, (x)\Big(N(x) \to (\exists x')\, F(x, x')\Big)$$
$$\&\, (y, z)\Big(F(y, z) \to N(y) \,\&\, N(z)\Big) \,\&\, (y)\, R_{S_0}(u, y)$$
$$\&\, I(u, u) \,\&\, K_{q_1}(u) \,\&\, \text{Des}(\mathfrak{M})\Big]$$
$$\to (\exists s)(\exists t)\,[N(s) \,\&\, N(t) \,\&\, R_{S_1}(s, t)].$$

$[N(u) \,\&\, \ldots \,\&\, \text{Des}\,(\mathfrak{M})]$ may be abbreviated to $A(\mathfrak{M})$.

When we substitute the meanings suggested on p. 259–60 we find that Un $(\mathfrak{M})$ has the interpretation "in some complete configuration of $\mathfrak{M}$, S_1 (*i.e.* 0) appears on the tape". Corresponding to this I prove that

(*a*) If S_1 appears on the tape in some complete configuration of $\mathfrak{M}$, then Un $(\mathfrak{M})$ is provable.

(*b*) If Un $(\mathfrak{M})$ is provable, then S_1 appears on the tape in some complete configuration of $\mathfrak{M}$.

When this has been done, the remainder of the theorem is trivial.

LEMMA 1. *If S_1 appears on the tape in some complete configuration of $\mathfrak{M}$, then* Un $(\mathfrak{M})$ *is provable.*

We have to show how to prove Un $(\mathfrak{M})$. Let us suppose that in the n-th complete configuration the sequence of symbols on the tape is $S_{r(n,0)}, S_{r(n,1)}, \ldots, S_{r(n,n)}$, followed by nothing but blanks, and that the scanned symbol is the $i(n)$-th, and that the m-configuration is $q_{k(n)}$. Then we may form the proposition

$$R_{S_{r(n,0)}}(u^{(n)}, u) \,\&\, R_{S_{r(n,1)}}(u^{(n)}, u') \,\&\, \ldots \,\&\, R_{S_{r(n,n)}}(u^{(n)}, u^{(n)})$$
$$\&\, I(u^{(n)}, u^{(i(n))}) \,\&\, K_{q_{k(n)}}(u^{(n)})$$
$$\&\, (y)F\Big((y, u') \vee F(u, y) \vee F(u', y) \vee \ldots \vee F(u^{(n-1)}, y) \vee R_{S_0}(u^{(n)}, y)\Big),$$

which we may abbreviate to CC_n.

As before, $F(u, u') \,\&\, F(u', u'') \,\&\, \ldots \,\&\, F(u^{(r-1)}, u^{(r)})$ is abbreviated to $F^{(r)}$.

I shall show that all formulae of the form $A(\mathfrak{M}) \,\&\, F^{(n)} \to CC_n$ (abbreviated to CF_n) are provable. The meaning of CF_n is "The n-th complete configuration of $\mathfrak{M}$ is so and so", where "so and so" stands for the actual n-th complete configuration of $\mathfrak{M}$. That CF_n should be provable is therefore to be expected.

CF_0 is certainly provable, for in the complete configuration the symbols are all blanks, the m-configuration is q_1, and the scanned square is u, *i.e.* CC_0 is

$$(y)\, R_{S_0}(u, y) \,\&\, I(u, u) \,\&\, K_{q_1}(u).$$

$A(\mathfrak{M}) \to CC_0$ is then trivial.

We next show that $CF_n \to CF_{n+1}$ is provable for each n. There are three cases to consider, according as in the move from the n-th to the $(n+1)$-th configuration the machine moves to left or to right or remains stationary. We suppose that the first case applies, *i.e.* the machine moves to the left. A similar argument applies in the other cases. If $r\big(n, i(n)\big) = a$, $r\big(n+1, i(n+1)\big) = c$, $k\big(i(n)\big) = b$, and $k\big(i(n+1)\big) = d$, then Des $(\mathfrak{M})$ must include Inst $\{q_a S_b S_d L q_c\}$ as one of its terms, *i.e.*

$$\text{Des}\,(\mathfrak{M}) \to \text{Inst}\,\{q_a S_b S_d L q_c\}.$$

Hence $\quad A(\mathfrak{M}) \,\&\, F^{(n+1)} \to \text{Inst}\,\{q_a S_b S_d L q_c\} \,\&\, F^{(n+1)}.$

But $\quad \text{Inst}\{q_a S_b S_d L q_c\} \,\&\, F^{(n+1)} \to (CC_n \to CC_{n+1})$

is provable, and so therefore is

$$A(\mathfrak{M}) \,\&\, F^{(n+1)} \to (CC_n \to CC_{n+1})$$

and $$\left(A(\mathcal{M}) \,\&\, F^{(n)} \rightarrow CC_n\right) \rightarrow \left(A(\mathcal{M}) \,\&\, F^{(n+1)} \rightarrow CC_{n+1}\right),$$

i.e. $$CF_n \rightarrow CF_{n+1}.$$

CF_n is provable for each n. Now it is the assumption of this lemma that S_1 appears somewhere, in some complete configuration, in the sequence of symbols printed by $\mathcal{M}$; that is, for some integers N, K, CC_N has $R_{S_1}(u^{(N)}, u^{(K)})$ as one of its terms, and therefore $CC_N \rightarrow R_{S_1}(u^{(N)}, u^{(K)})$ is provable. We have then

$$CC_N \rightarrow R_{S_1}(u^{(N)}, u^{(K)})$$

and $$A(\mathcal{M}) \,\&\, F^{(N)} \rightarrow CC^N.$$

We also have

$$(\exists u)\, A(\mathcal{M}) \rightarrow (\exists u)(\exists u') \ldots (\exists u^{(N')}) \left(A(\mathcal{M}) \,\&\, F^{(N)}\right),$$

where $N' = \max(N, K)$. And so

$$(\exists u)\, A(\mathcal{M}) \rightarrow (\exists u)\,(\exists u') \ldots (\exists u^{(N')})\, R_{S_1}(u^{(N)}, u^{(K)}),$$

$$(\exists u)\, A(\mathcal{M}) \rightarrow (\exists u^{(N)})\,(\exists u^{(K)})\, R_{S_1}(u^{(N)}, u^{(K)}),$$

$$(\exists u)\, A(\mathcal{M}) \rightarrow (\exists s)\,(\exists t)\, R_{S_1}(s, t),$$

i.e. $\mathrm{Un}(\mathcal{M})$ is provable.

This completes the proof of Lemma 1.

LEMMA 2. *If* $\mathrm{Un}(\mathcal{M})$ *is provable, then* S_1 *appears on the tape in some complete configuration of* $\mathcal{M}$.

If we substitute any propositional functions for function variables in a provable formula, we obtain a true proposition. In particular, if we substitute the meanings tabulated on pp. 259–260 in $\mathrm{Un}(\mathcal{M})$, we obtain a true proposition with the meaning "S_1 appears somewhere on the tape in some complete configuration of $\mathcal{M}$".

We are now in a position to show that the Entscheidungsproblem cannot be solved. Let us suppose the contrary. Then there is a general (mechanical) process for determining whether $\mathrm{Un}(\mathcal{M})$ is provable. By Lemmas 1 and 2, this implies that there is a process for determining whether $\mathcal{M}$ ever prints 0, and this is impossible, by § 8. Hence the Entscheidungsproblem cannot be solved.

In view of the large number of particular cases of solutions of the Entscheidungsproblem for formulae with restricted systems of quantors, it is interesting to express $\mathrm{Un}(\mathcal{M})$ in a form in which all quantors are at the beginning. $\mathrm{Un}(\mathcal{M})$ is, in fact, expressible in the form

$$(u)\,(\exists x)\,(w)\,(\exists u_1) \ldots (\exists u_n)\, \mathfrak{B}, \tag{I}$$

where $\mathfrak{B}$ contains no quantors, and $n = 6$. By unimportant modifications we can obtain a formula, with all essential properties of $\mathrm{Un}(\mathcal{M})$, which is of form (I) with $n = 5$.

Added 28 August, 1936.

APPENDIX.

Computability and effective calculability

The theorem that all effectively calculable (λ-definable) sequences are computable and its converse are proved below in outline. It is assumed that the terms "well-formed formula" (W.F.F.) and "conversion" as used by Church and Kleene are understood. In the second of these proofs the existence of several formulae is assumed without proof; these formulae may be constructed straightforwardly with the help of, *e.g.*, the results of Kleene in "A theory of positive integers in formal logic", *American Journal of Math.*, 57 (1935), 153-173, 219-244.

The W.F.F. representing an integer n will be denoted by N_n. We shall say that a sequence γ whose n-th figure is $\phi_\gamma(n)$ is λ-definable or effectively calculable if $1+\phi_\gamma(u)$ is a λ-definable function of n, *i.e.* if there is a W.F.F. M_γ such that, for all integers n,

$$\{M_\gamma\}\,(N_n) \ \mathrm{conv}\ N_{\phi_\gamma(n)+1},$$

i.e. $\{M_\gamma\}\,(N_n)$ is convertible into $\lambda xy\,.\,x\big(x(y)\big)$ or into $\lambda xy\,.\,x(y)$ according as the n-th figure of λ is 1 or 0.

To show that every λ-definable sequence γ is computable, we have to show how to construct a machine to compute γ. For use with machines it is convenient to make a trivial modification in the calculus of conversion. This alteration consists in using x, x', x'', ... as variables instead of a, b, c, ... We now construct a machine $\mathcal{L}$ which, when supplied with the formula M_γ, writes down the sequence γ. The construction of $\mathcal{L}$ is somewhat similar to that of the machine $\mathcal{K}$ which proves all provable formulae of the functional calculus. We first construct a choice machine $\mathcal{L}_1$, which, if supplied with a W.F.F., M say, and suitably manipulated, obtains any formula into which M is convertible. $\mathcal{L}_1$ can then be modified so as to yield an automatic machine $\mathcal{L}_2$ which obtains successively all the formulae into which M is convertible (cf. foot-note p. 252). The machine $\mathcal{L}$ includes $\mathcal{L}_2$ as a part. The motion of the machine $\mathcal{L}$ when supplied with the formula M_γ is divided into sections of which the n-th is devoted to finding the n-th figure of γ. The first stage in this n-th section is the formation of $\{M_\gamma\}(N_n)$. This formula is then supplied to the machine $\mathcal{L}_2$, which converts it successively into various other formulae. Each formula into which it is convertible eventually appears, and each, as it is found, is compared with

$$\lambda x\Big[\lambda x'\Big[\{x\}\big(\{x\}(x')\big)\Big]\Big], \quad i.e.\ N_2,$$

and with $$\lambda x\Big[\lambda x'[\{x\}(x')]\Big], \quad i.e.\ N_1.$$

If it is identical with the first of these, then the machine prints the figure 1 and the n-th section is finished. If it is identical with the second, then 0 is printed and the section is finished. If it is different from both, then the work of $\mathcal{L}_2$ is resumed. By hypothesis, $\{M_\gamma\}(N_n)$ is convertible into one of the formulae N_2 or N_1; consequently the n-th section will eventually be finished, *i.e.* the n-th figure of γ will eventually be written down.

To prove that every computable sequence γ is λ-definable, we must show how to find a formula M_γ such that, for all integers n,

$$\{M_\gamma\}(N_n) \text{ conv } N_{1+\phi_\gamma(n)}.$$

Let $\mathcal{M}$ be a machine which computes γ and let us take some description of the complete configurations of $\mathcal{M}$ by means of numbers, *e.g.* we may take the D.N of the complete configuration as described in §6. Let $\xi(n)$ be the D.N of the n-th complete configuration of $\mathcal{M}$. The table for the machine $\mathcal{M}$ gives us a relation between $\xi(n+1)$ and $\xi(n)$ of the form

$$\xi(n+1) = \rho_\gamma\big(\xi(n)\big),$$

where ρ_γ is a function of very restricted, although not usually very simple, form: it is determined by the table for $\mathcal{M}$. ρ_γ is λ-definable (I omit the proof of this), *i.e.* there is a W.F.F. A_γ such that, for all integers n,

$$\{A_\gamma\}(N_{\xi(n)}) \text{ conv } N_{\xi(n+1)}.$$

Let U stand for

$$\lambda u\Big[\Big\{\{u\}(A_\gamma)\Big\}(N_r)\Big],$$

where $r = \xi(0)$; then, for all integers n,

$$\{U_\gamma\}(N_n) \text{ conv } N_{\xi(n)}.$$

It may be proved that there is a formula V such that

$$\Big\{\{V\}(N_{\xi(n+1)})\Big\}(N_{\xi(n)}) \begin{cases} \text{conv } N_1 & \text{if, in going from the } n\text{-th to the } (n+1)\text{-th complete configuration, the figure 0 is printed.} \\ \text{conv } N_2 & \text{if the figure 1 is printed.} \\ \text{conv } N_3 & \text{otherwise.} \end{cases}$$

Let W_γ stand for

$$\lambda u\Big[\Big\{\{V\}\Big(\{A_\gamma\}\big(\{U_\gamma\}(u)\big)\Big)\Big\}\Big(\{U_\gamma\}(u)\Big)\Big],$$

so that, for each integer n,

$$\Big\{\{V\}(N_{\xi(n+1)})\Big\}(N_{\xi(n)}) \text{ conv } \{W_\gamma\}(N_n),$$

and let Q be a formula such that

$$\Big\{\{Q\}(W_\gamma)\Big\}(N_s) \text{ conv } N_{r(s)},$$

where $r(s)$ is the s-th integer q for which $\{W_\gamma\}(N_q)$ is convertible into either N_1 or N_2. Then, if M_γ stands for

$$\lambda w\Big[\{W_\gamma\}\Big(\Big\{\{Q\}(W_\gamma)\Big\}(w)\Big)\Big],$$

it will have the required property†.

The Graduate College,
Princeton University,
New Jersey, U.S.A.

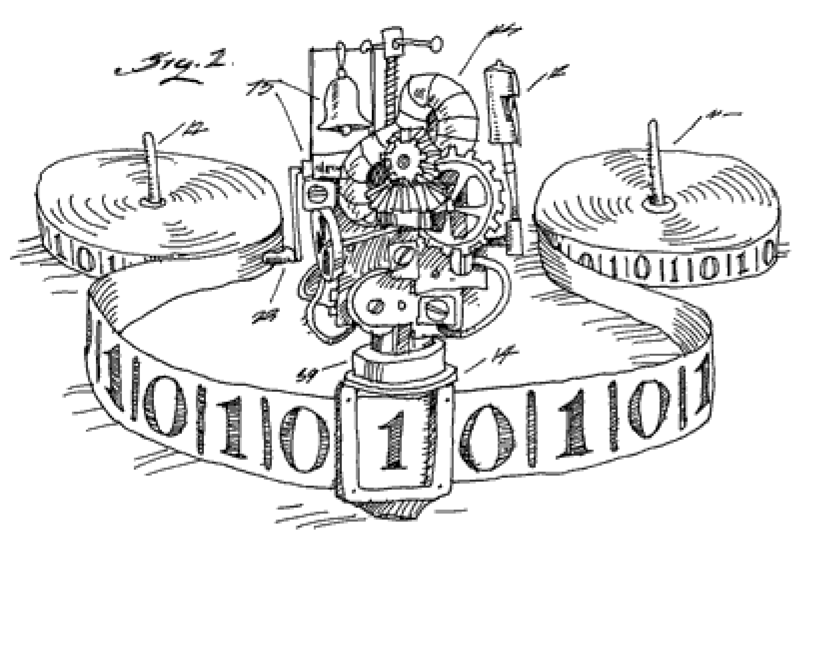
† In a complete proof of the λ-definability of computable sequences it would be best to modify this method by replacing the numerical description of the complete configurations by a description which can be handled more easily with our apparatus. Let us choose certain integers to represent the symbols and the m-configurations of the machine. Suppose that in a certain complete configuration the numbers representing the successive symbols on the tape are $s_1 s_2 \ldots s_n$, that the m-th symbol is scanned, and that the m-configuration has the number t; then we may represent this complete configuration by the formula

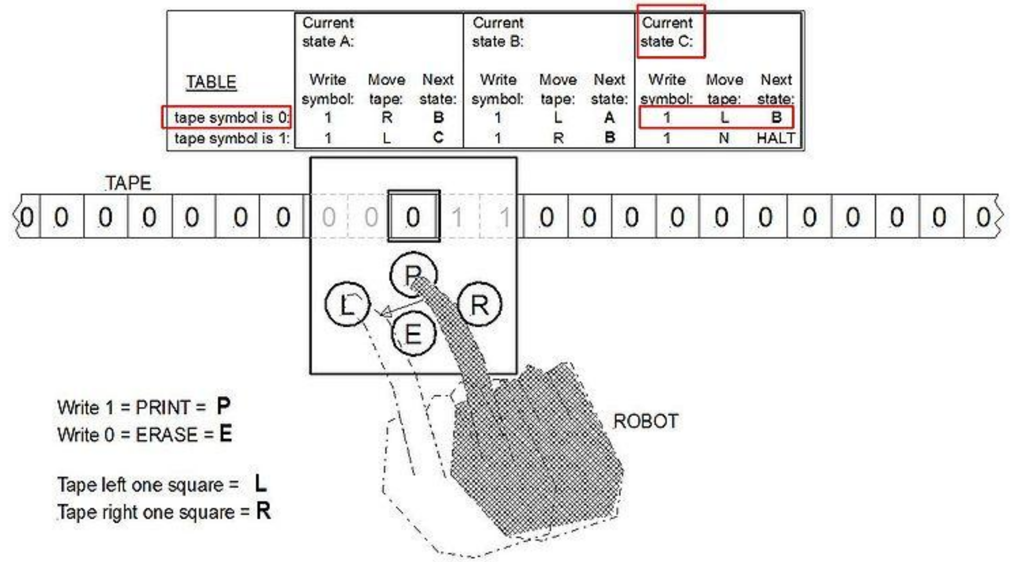
$$\Big[[N_{s_1}, N_{s_2}, \ldots, N_{s_{m-1}}],\ [N_t, N_{s_m}],\ [N_{s_{m+1}}, \ldots, N_{s_n}]\Big],$$

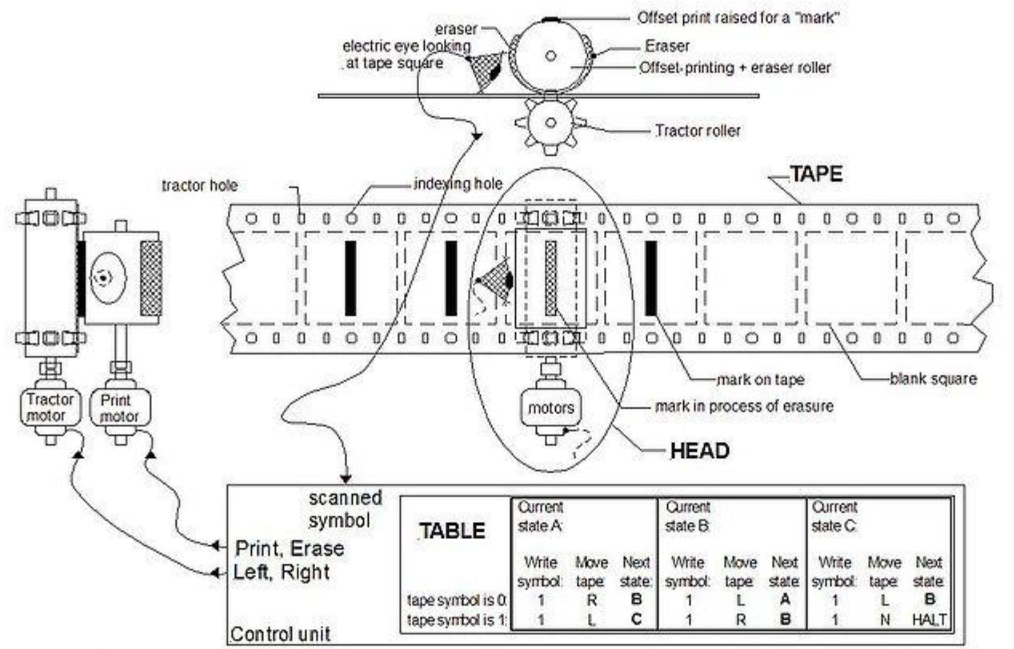
where $$[a, b] \text{ stands for } \lambda u\Big[\Big\{\{u\}(a)\Big\}(b)\Big],$$

$$[a, b, c] \text{ stands for } \lambda u\Big[\Big\{\Big\{\{u\}(a)\Big\}(b)\Big\}(c)\Big],$$

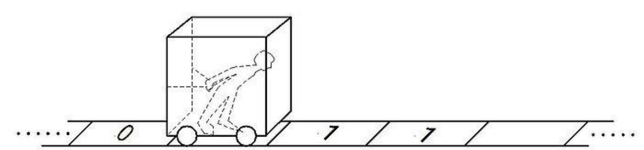
etc.







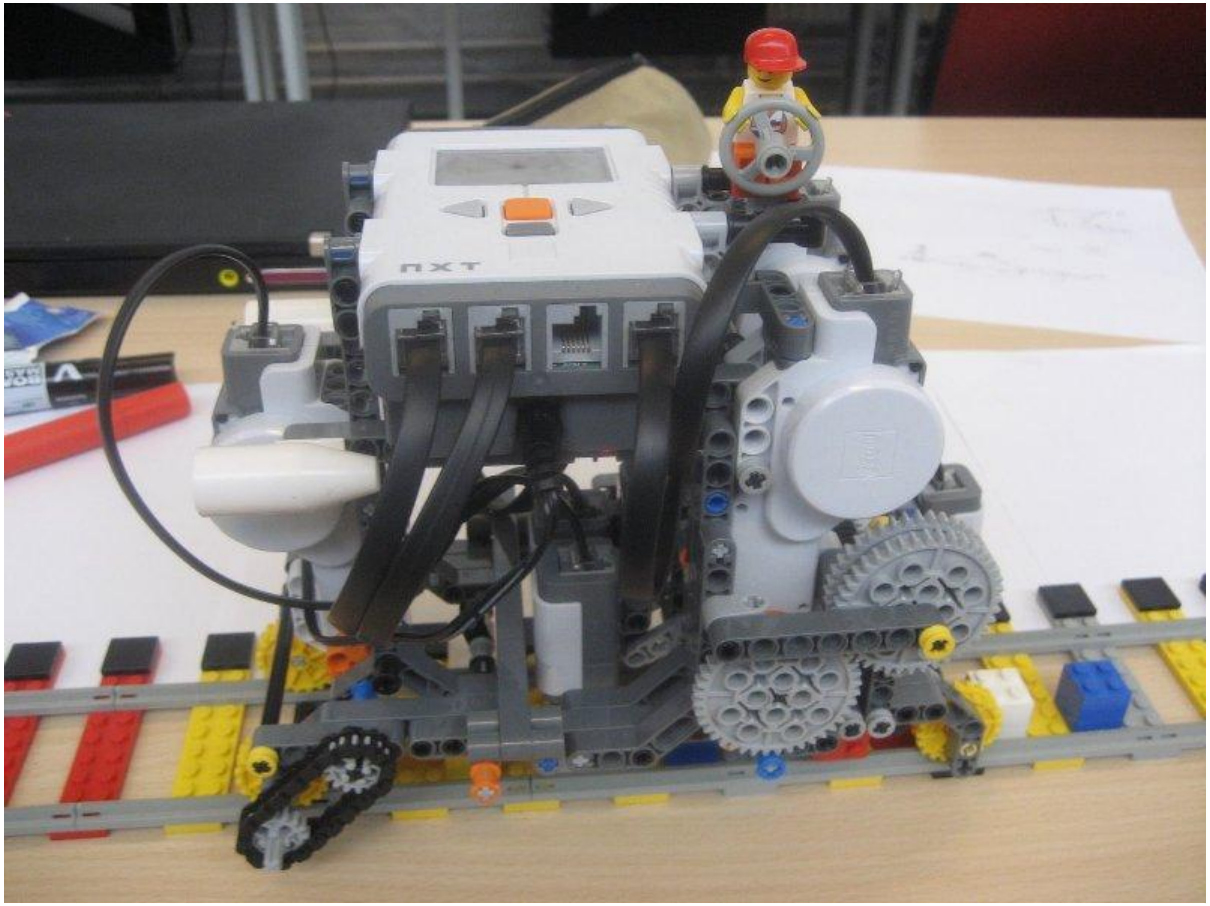
A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

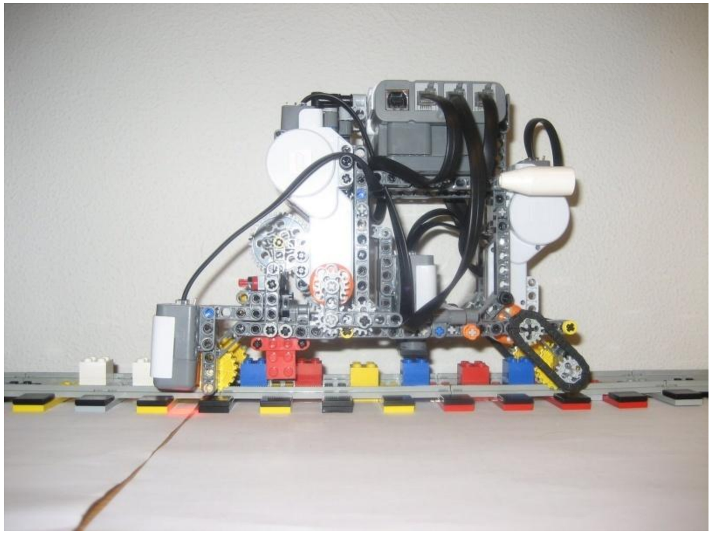


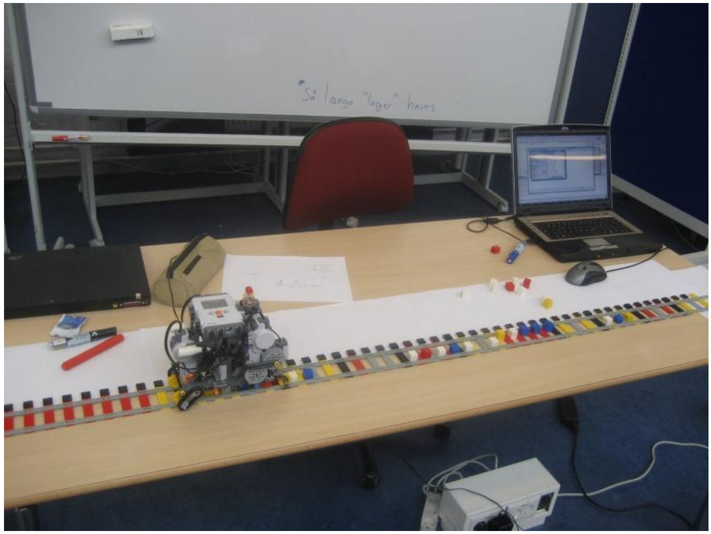
Turing's insight:

simple local actions can lead to arbitrarily complex computations!

Lego Turing Machines

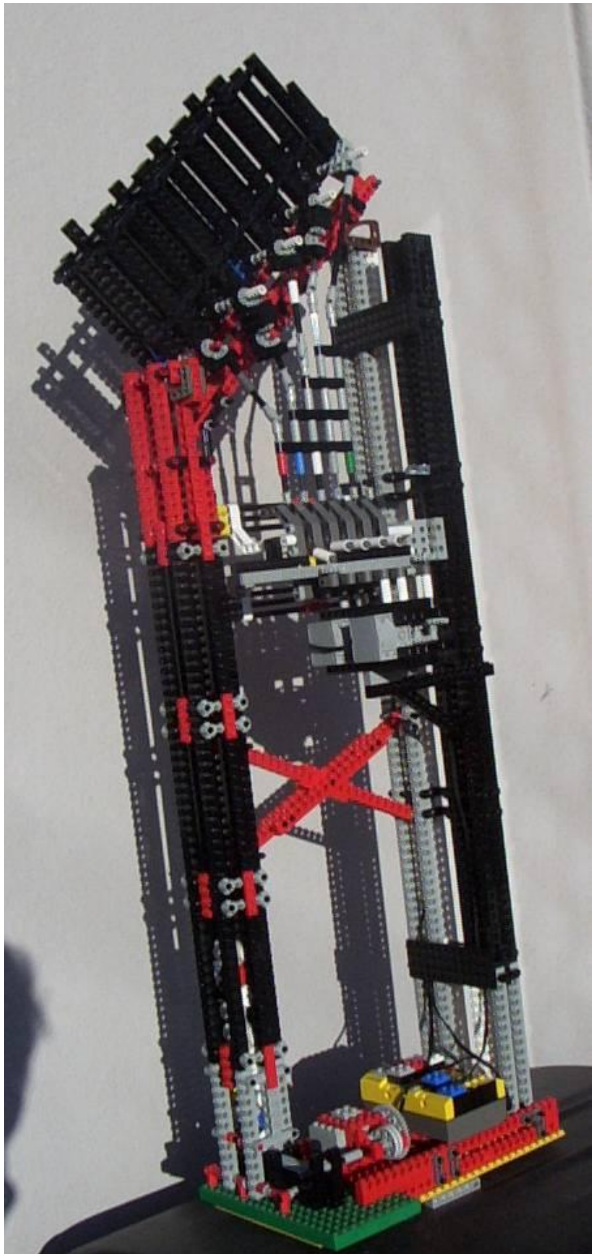


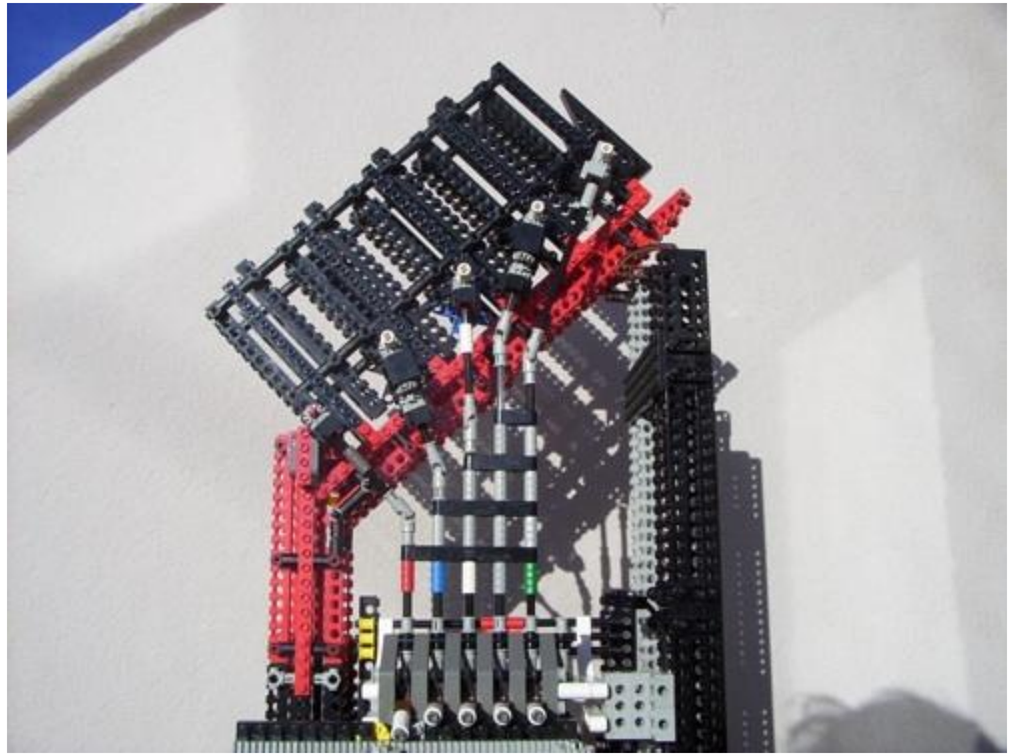




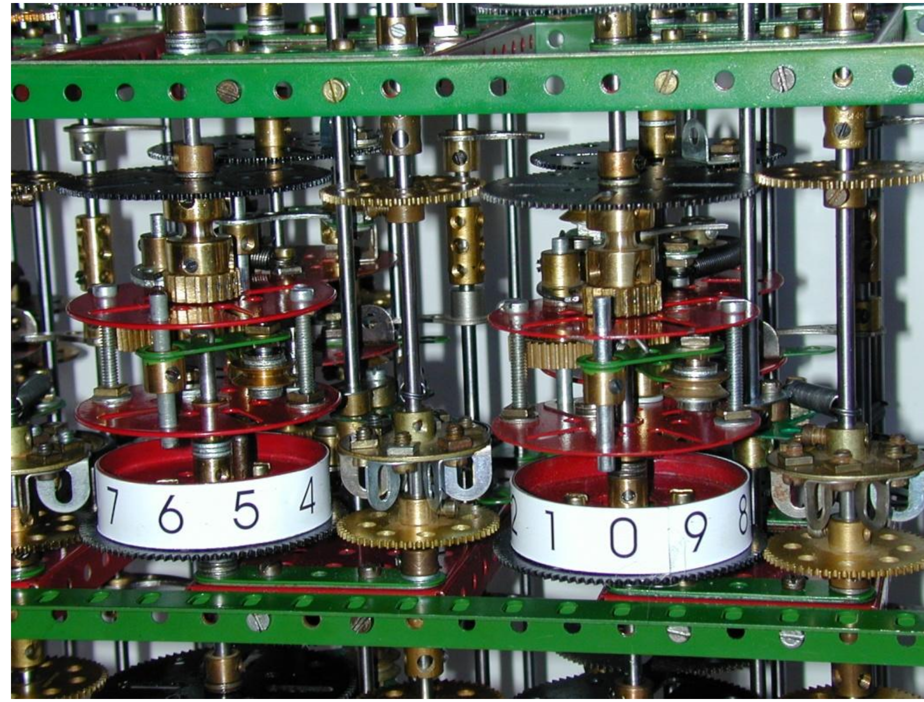
See: http://www.youtube.com/watch?v=cYw2ewoO6c4

Lego Turing Machines



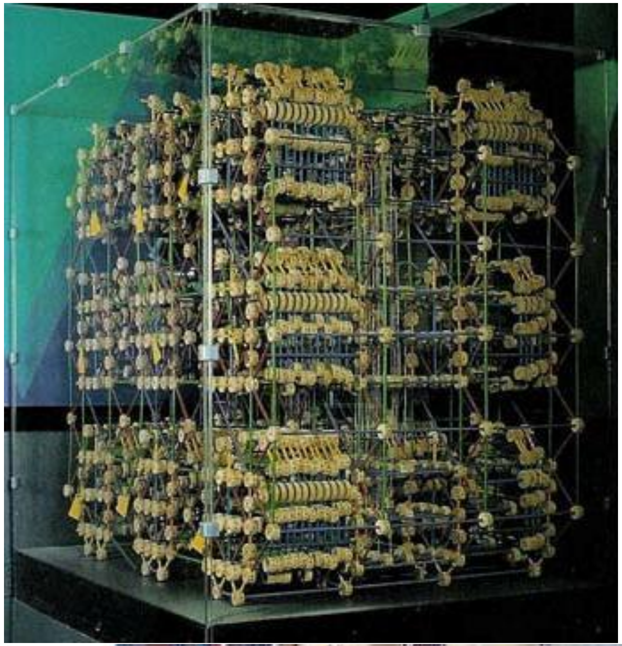


“Mechano” Computers

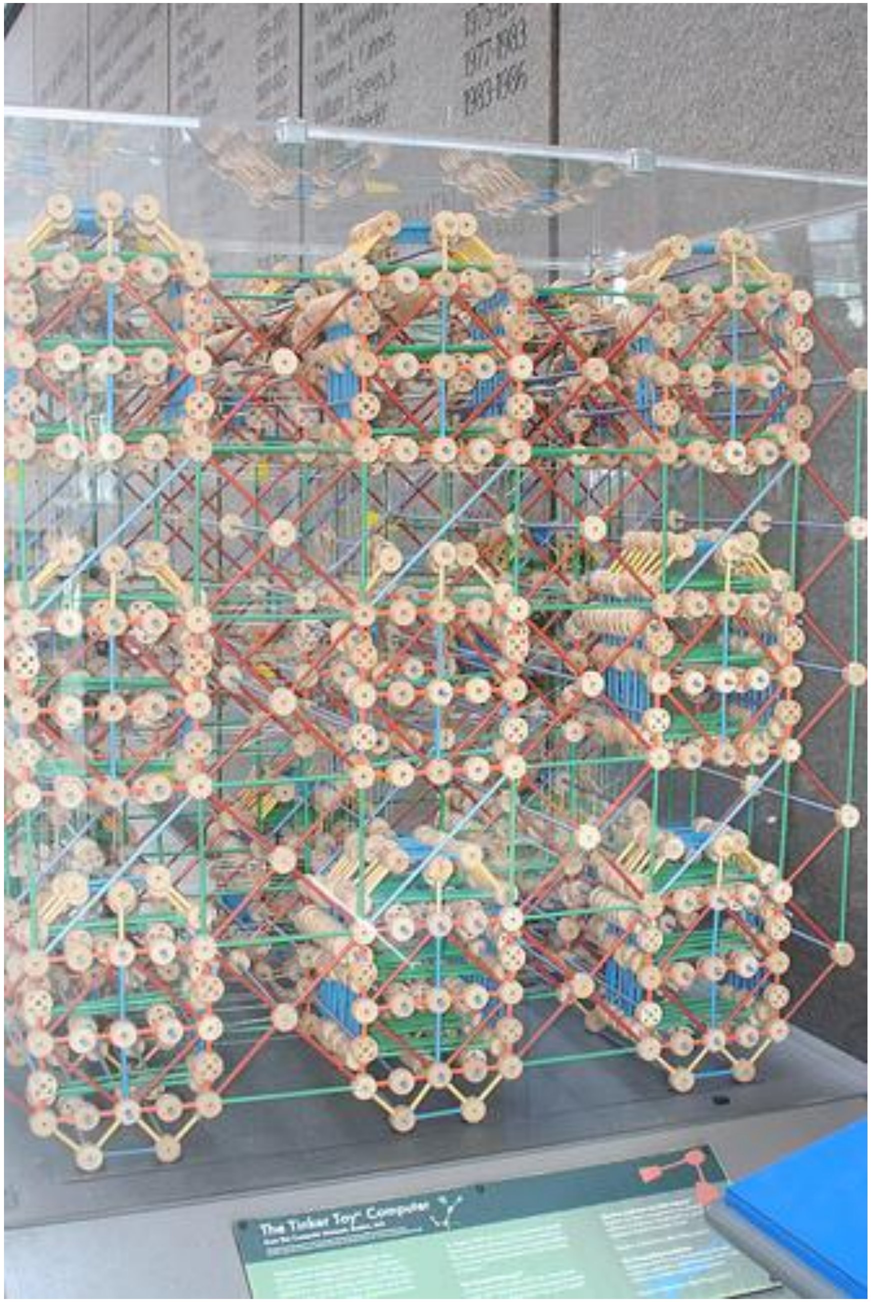


Babbage’s difference engine

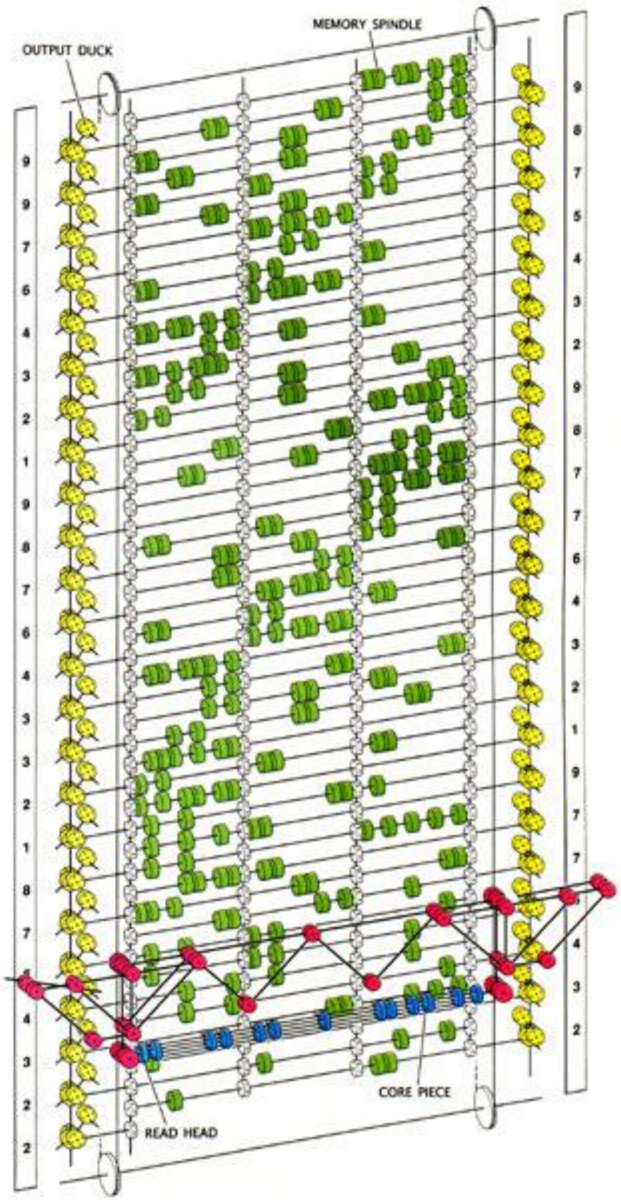
Tinker Toy Computers



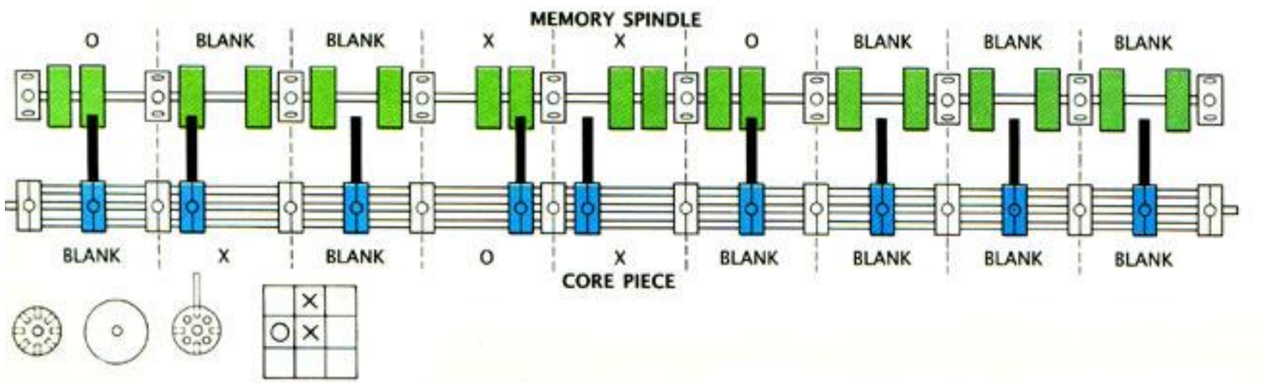
Plays
tic-tac-toe!



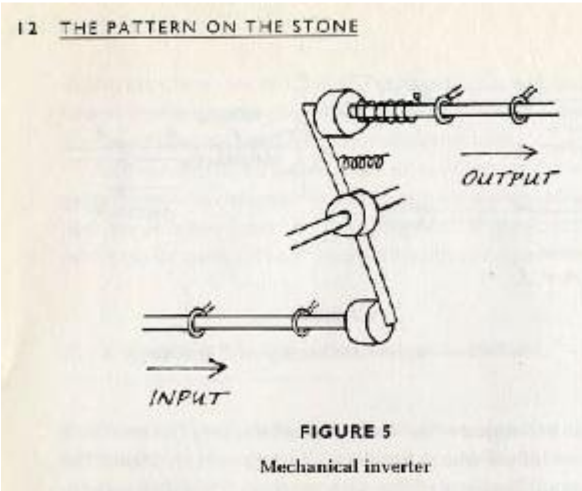
Tinker Toy Computers

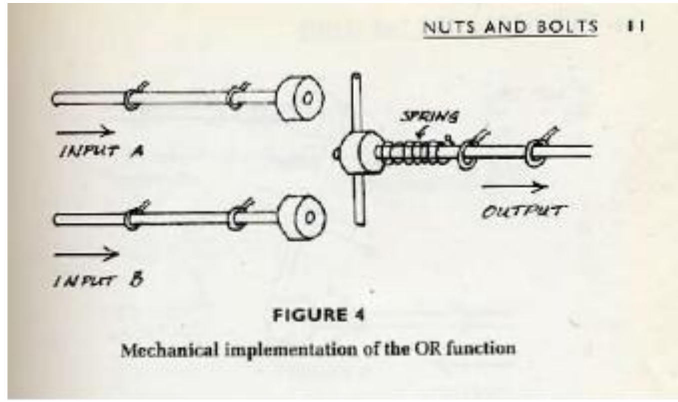


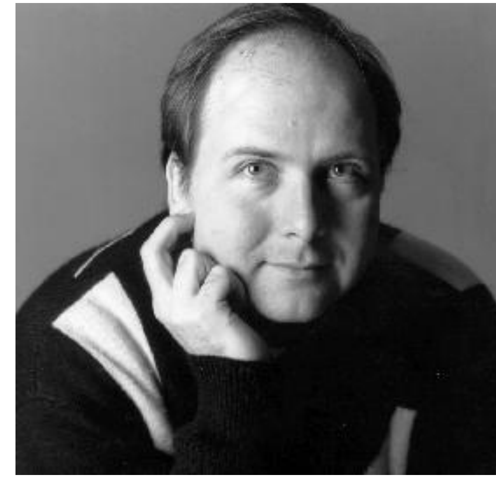
The Tinkertoy computer: ready for a game of tic-tac-toe



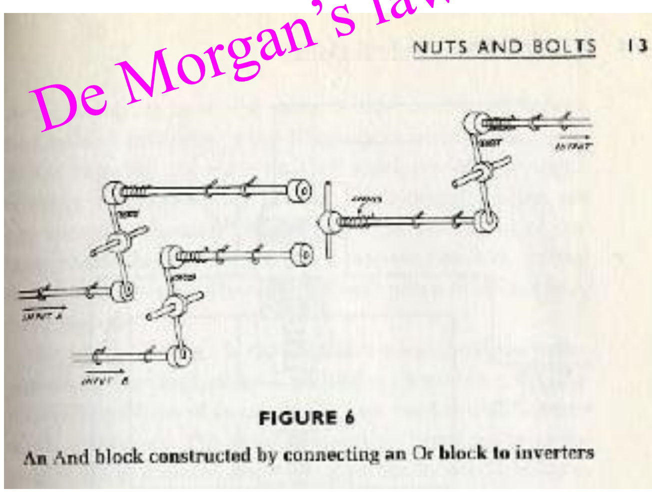
Mechanical Computers

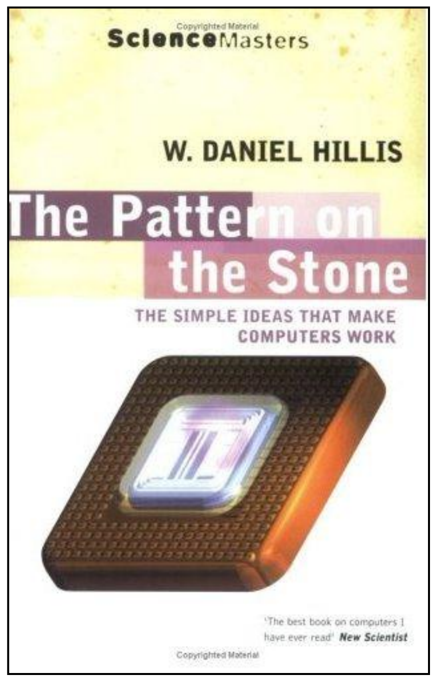




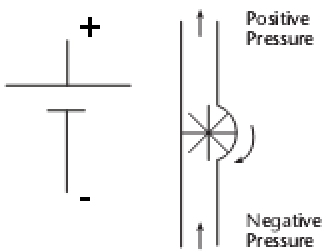


De Morgan's law!

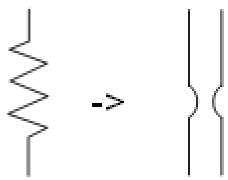




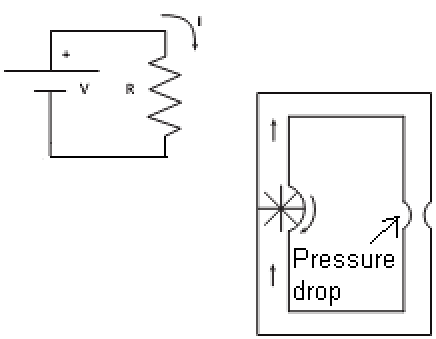
Hydraulic Computers



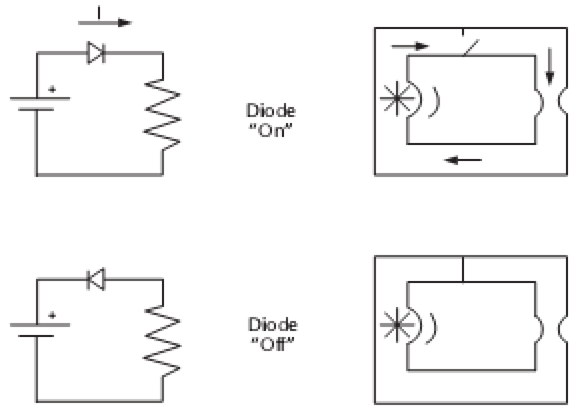
Voltage source
or inductor



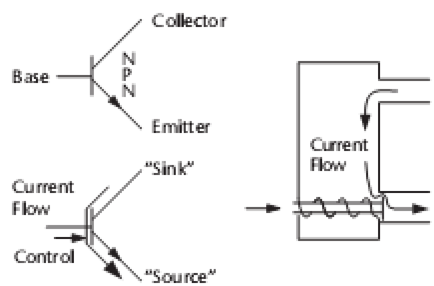
Resistor



Simple circuit



Diode



Transistor

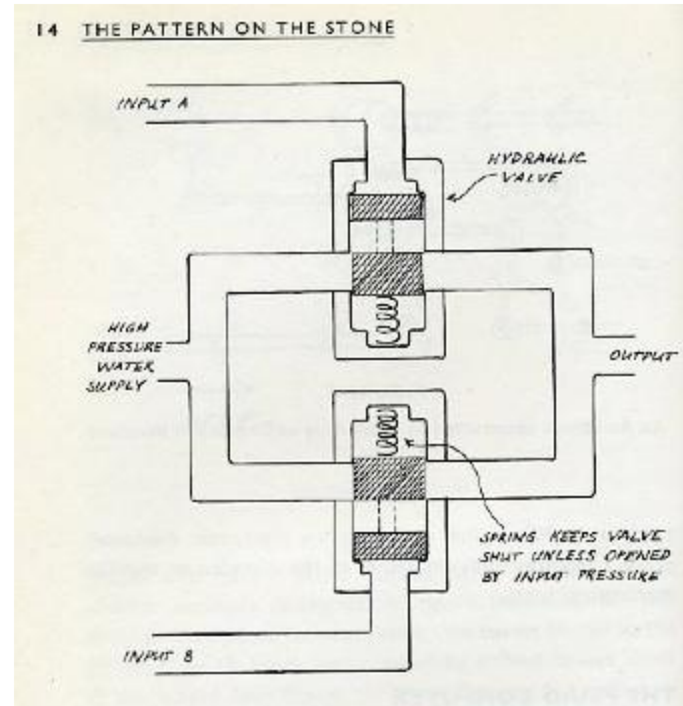
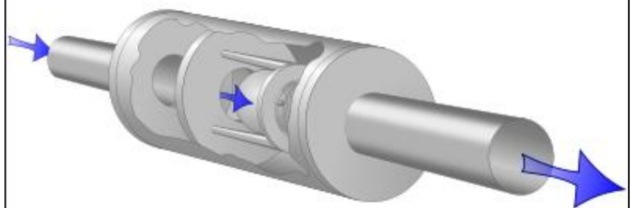


FIGURE 7

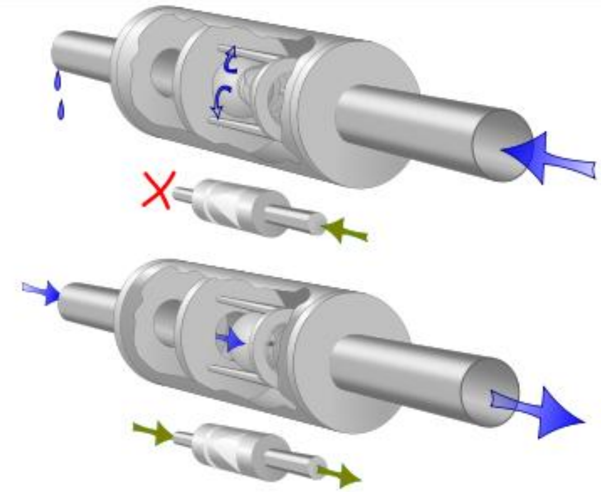
An Or block built with hydraulic valves

Hydraulic Computers

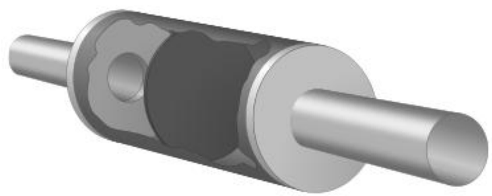
Wire



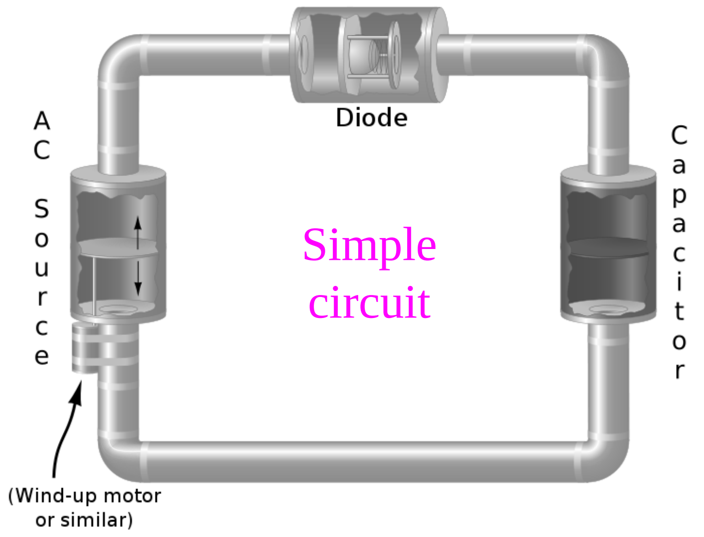
Diode



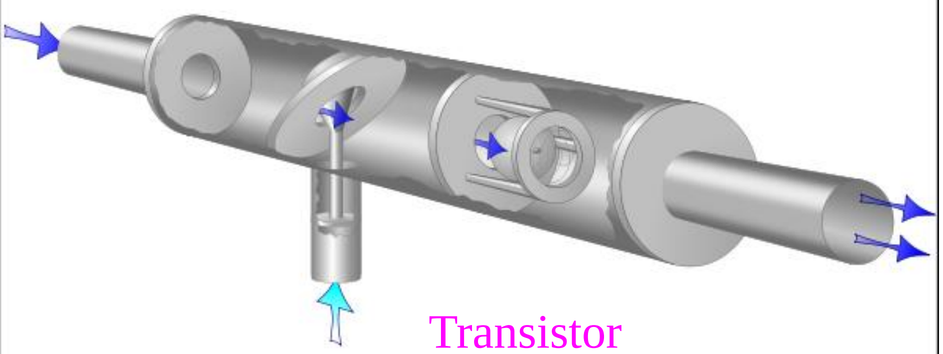
Resistor



Capacitor

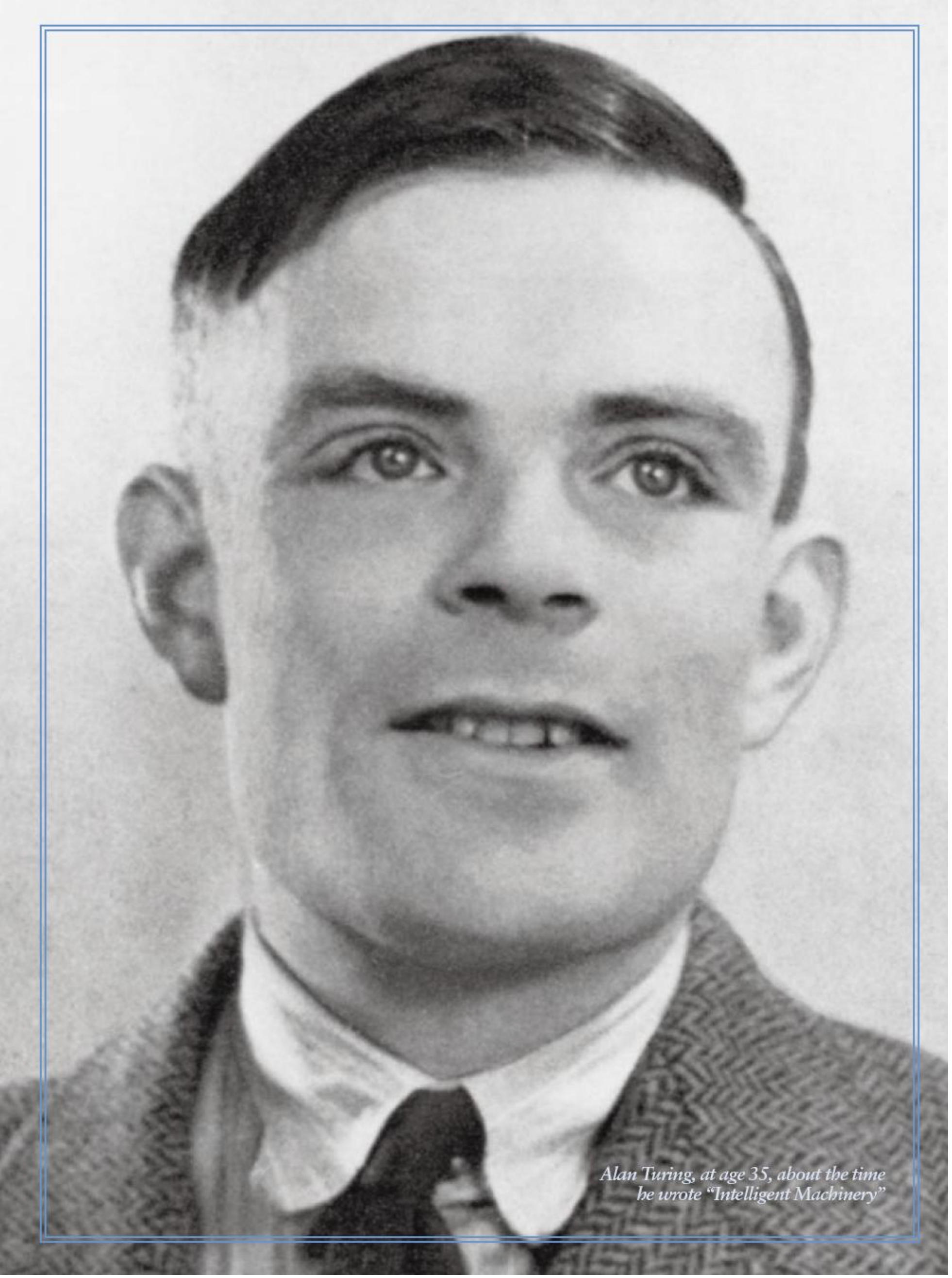


Simple circuit



Transistor

Theorem: fluid-based "circuits" are Turing-complete / universal!



Alan Turing, at age 35, about the time he wrote "Intelligent Machinery"

Alan Turing's Forgotten Ideas in Computer Science

Well known for the machine, test and thesis that bear his name, the British genius also anticipated neural-network computers and "hypercomputation"

by B. Jack Copeland and Diane Proudfoot

Alan Mathison Turing conceived of the modern computer in 1935. Today all digital computers are, in essence, "Turing machines." The British mathematician also pioneered the field of artificial intelligence, or AI, proposing the famous and widely debated Turing test as a way of determining whether a suitably programmed computer can think. During World War II, Turing was instrumental in breaking the German Enigma code in part of a top-secret British operation that historians say shortened the war in Europe by two years. When he died at the age of 41, Turing was doing the earliest work on what would now be called artificial life, simulating the chemistry of biological growth.

Throughout his remarkable career, Turing had no great interest in publicizing his ideas. Consequently, important aspects of his work have been neglected or forgotten over the years. In particular, few people—even those knowledgeable about computer science—are familiar with Turing's fascinating anticipation of connectionism, or neuronlike computing. Also neglected are his groundbreaking theoretical concepts in the exciting area of "hypercomputation." According to some experts, hypercomputers might one day solve problems heretofore deemed intractable.

The Turing Connection

Digital computers are superb number crunchers. Ask them to predict a rocket's trajectory or calculate the financial figures for a large multinational corporation, and they can churn out the answers in seconds. But seemingly simple actions that people routinely perform, such as recognizing a face or reading handwriting, have been devilishly tricky to program. Perhaps the networks of neurons that make up the brain have a natural facility for such tasks that standard computers lack. Scientists have thus been investigating computers modeled more closely on the human brain.

Connectionism is the emerging science of computing with networks of artificial neurons. Currently researchers usually simulate the neurons and their interconnections within an ordinary digital computer (just as engineers create virtual models of aircraft wings and skyscrapers). A training algorithm that runs on the computer adjusts the connections between the neurons, honing the network into a special-purpose machine dedicated to some particular function, such as forecasting international currency markets.

Modern connectionists look back to Frank Rosenblatt, who published the first of many papers on the topic in 1957, as the founder of their approach. Few realize that Turing had already investigated connectionist networks as early as 1948, in a little-known paper entitled "Intelligent Machinery."

Written while Turing was working for the National Physical Laboratory in London, the manuscript did not meet with his employer's approval. Sir Charles Darwin, the rather headmasterly director of the laboratory and grandson of the great English naturalist, dismissed it as a "schoolboy essay." In reality, this farsighted paper was the first manifesto of the field of artificial intelligence. In the work—which remained unpublished until 1968, 14 years after Turing's death—the British mathematician not only set out the fundamentals of connectionism but also brilliantly introduced many of the concepts that were later to become central to AI, in some cases after reinvention by others.

In the paper, Turing invented a kind of neural network that he called a "B-type unorganized machine," which consists of artificial neurons and devices that modify the connections between them. B-type machines may contain any number of neurons connected in any pattern but are always subject to the restriction that each neuron-to-neuron connection must pass through a modifier device.

Few realize that Turing had already investigated connectionist networks as early as 1948.

All connection modifiers have two training fibers. Applying a pulse to one of them sets the modifier to "pass mode," in which an input—either 0 or 1—passes through unchanged and becomes the output. A pulse on the other fiber places the modifier in "interrupt mode," in which the output is always 1, no matter what the input is. In this state the modifier destroys all information attempting to pass along the connection to which it is attached.

Once set, a modifier will maintain its function (either "pass" or "interrupt") unless it receives a pulse on the other training fiber. The presence of these ingenious connection modifiers enables the training of a B-type unorganized machine by means of what Turing called "appropriate interference, mimicking education." Actually, Turing theorized that "the cortex of an infant is an unorganized machine, which can be organized by suitable interfering training."

Each of Turing's model neurons has two input fibers, and the output of a neuron is a simple logical function of its two inputs. Every neuron in the network executes the same logical operation of "not and" (or NAND): the output is 1 if either of the inputs is 0. If both inputs are 1, then the output is 0.

Turing selected NAND because every other logical (or Boolean) operation can be accomplished by groups of NAND neurons. Furthermore, he showed that even the connection modifiers themselves can be built out of NAND neurons. Thus, Turing specified a network made up of nothing more than NAND neurons and their connecting fibers—about the simplest possible model of the cortex.

In 1958 Rosenblatt defined the theoretical basis of connectionism in one succinct statement: "Stored information takes the form of new connections, or transmission channels in the nervous system (or the creation of conditions which are functionally equivalent to new connections)." Because the destruction of existing connections can be functionally equivalent to the creation of new ones, researchers can build a network for accomplishing a specific task by taking one with an excess of connections and selectively destroying some of them. Both actions—destruction and creation—are employed in the training of Turing's B-types.

At the outset, B-types contain random interneural connections whose modifiers have been set by chance to either pass or interrupt. During training, unwanted connections are destroyed by switching their attached modifiers to interrupt mode. Conversely, changing a modifier from interrupt to pass in effect creates a connection. This selective culling and enlivening of connections hones the initially random network into one organized for a given job.

Turing wished to investigate other kinds of unorganized machines, and he longed to simulate a neural network and its training regimen using an ordinary digital computer. He would, he said, "allow the whole system to run for an appreciable period, and then break in as a kind of 'inspector of schools' and see what progress had been made." But his own work on neural networks was carried out shortly before the first general-purpose electronic computers became available. (It was not until 1954, the year of Turing's death, that Belmont G. Farley and Wesley A. Clark succeeded at the Massachusetts Institute of Technology in running the first computer simulation of a small neural network.)

Paper and pencil were enough, though, for Turing to show that a sufficiently large B-type neural network can be configured (via its connection modifiers) in such a way that it becomes a general-purpose computer. This discovery illuminates one of the most fundamental problems concerning human cognition.

From a top-down perspective, cognition includes complex sequential processes, often involving language or other forms of symbolic representation, as in mathematical calculation. Yet from a bottom-up view, cognition is nothing but the simple firings of neurons. Cognitive scientists face the problem of how to reconcile these very different perspectives.

Turing's discovery offers a possible solution: the cortex, by virtue of being a neural network acting as a general-purpose computer, is able to carry out the sequential, symbol-rich processing discerned in the view from the top. In 1948 this hypothesis was well ahead of its time, and today it remains among the best guesses concerning one of cognitive science's hardest problems.

Computing the Uncomputable

In 1935 Turing thought up the abstract device that has since become known as the "universal Turing machine." It consists of a limitless memory that stores both program and data and a scanner that moves back and forth through the memory, symbol by symbol, reading the information and writing additional symbols. Each of the machine's basic actions is very simple—such as "identify the symbol on which the scanner is positioned," "write '1'" and "move one position to the left." Complexity is achieved by chaining together large numbers of these basic actions. Despite its simplicity, a universal Turing machine can execute any task that can be done by the most powerful of today's computers. In fact, all modern digital computers are in essence universal Turing machines [see "Turing Machines," by John E. Hopcroft; Scientific American, May 1984].

Turing's aim in 1935 was to devise a machine—one as simple as possible—capable of any calculation that a human mathematician working in accordance with some algorithmic method could perform, given unlimited time, energy, paper and pencils, and perfect concentration. Calling a machine "universal" merely signifies that it is capable of all such calculations. As Turing himself wrote, "Electronic computers are intended to carry out any definite rule-of-thumb process which could have been done by a human operator working in a disciplined but unintelligent manner."

Such powerful computing devices notwithstanding, an intriguing question arises: Can machines be devised that are capable of accomplishing even more? The answer is that these "hypermachines" can be described on paper, but no one as yet knows whether it will be possible to build one. The field of hypercomputation is currently attracting a growing number of scientists. Some speculate that the human brain itself—the most complex information processor known—is actually a naturally occurring example of a hypercomputer.

Before the recent surge of interest in hypercomputation, any information-processing job that was known to be too difficult for universal Turing machines was written off as "uncomputable." In this sense, a hypermachine computes the uncomputable.

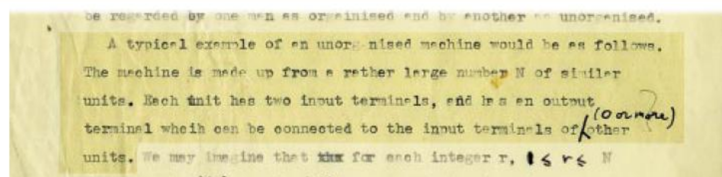
Examples of such tasks can be found in even the most straightforward areas of mathematics. For instance, given arithmetical statements picked at random, a universal Turing machine may not always be able to tell which are theorems (such as "7 + 5 = 12") and which are nontheorems (such as "every number is the sum of two even numbers"). Another type of uncomputable problem comes from geometry. A set of tiles—variously sized squares with different colored edges—"tiles the plane" if the Euclidean plane can be covered by copies of the tiles with no gaps or overlaps and with adjacent edges always the same color. Logicians William Hanf and Dale Myers of the University of Hawaii have discovered a tile set that tiles the plane only in patterns too complicated for a universal Turing machine to calculate. In the field of computer science, a universal Turing machine cannot always predict whether a given program will terminate or continue running forever. This is sometimes expressed by saying that no general-purpose programming language (Pascal, BASIC, Prolog, C and so on) can have a foolproof crash debugger: a tool that detects all bugs that could lead to crashes, including errors that result in infinite processing loops.

Turing himself was the first to investigate the idea of machines that can perform mathematical tasks too difficult

Turing's Anticipation of Connectionism

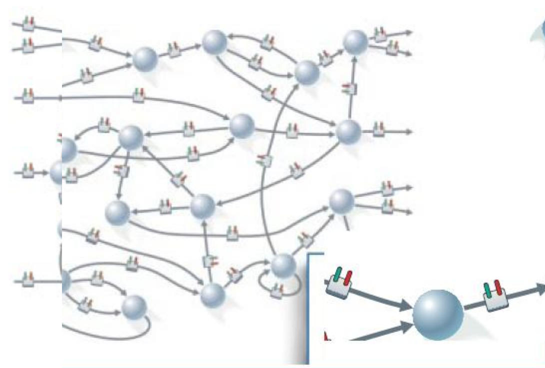
In a paper that went unpublished until 14 years after his death (*top*), Alan Turing described a network of artificial neurons connected in a random manner. In this "B-type unorganized machine" (*bottom left*), each connection passes through a modifier that is set either to allow data to pass unchanged (*green fiber*) or to destroy the transmitted information (*red fiber*). Switching the modifiers from one mode to the other enables the network to be trained. Note that each neuron has two inputs (*bottom left, inset*) and executes the simple logical operation of "not and," or NAND: if both inputs are 1, then the output is 0; otherwise the output is 1.

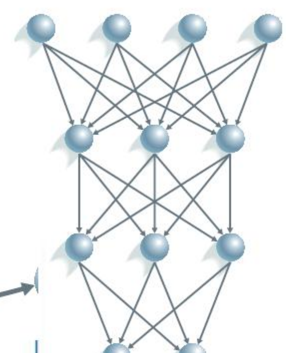
In Turing's network the neurons interconnect freely. In contrast, modern networks (*bottom center*) restrict the flow of information from layer to layer of neurons. Connectionists aim to simulate the neural networks of the brain (*bottom right*).

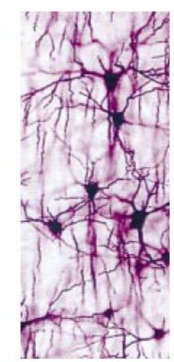


be regarded by one man as organised and by another as unorganised.

A typical example of an unorganised machine would be as follows. The machine is made up from a rather large number N of similar units. Each unit has two input terminals, and has an output terminal whcih can be connected to the input terminals of (0 or more) other units. We may imagine that for each integer r, 1 ≤ r ≤ N







TOM MOORE (*illustrations*); KING'S COLLEGE MODERN ARCHIVES, CAMBRIDGE UNIVERSITY LIBRARY (*top*); PETER ARNOLD, INC. (*bottom right*)

Using an Oracle to Compute the Uncomputable

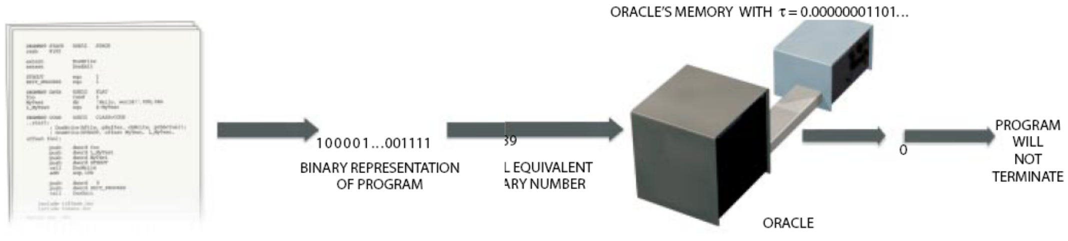
Alan Turing proved that his universal machine—and by extension, even today’s most powerful computers—could never solve certain problems. For instance, a universal Turing machine cannot always determine whether a given software program will terminate or continue running forever. In some cases, the best the universal machine can do is execute the program and wait—maybe eternally—for it to finish. But in his doctoral thesis (*below*), Turing did imagine that a machine equipped with a special “oracle” could perform this and other “uncomputable” tasks. Here is one example of how, in principle, an oracle might work.

Consider a hypothetical machine for solving the formidable “terminating program” problem (*above*). A computer program can be represented as a finite string of 1s and 0s. This sequence of digits can also be thought of as the binary representation of an integer, just as 1011011 is the equivalent of 91. The oracle’s job can then be restated as, “Given an integer that represents a program (for any computer that can be simulated by a universal Turing machine), output a ‘1’ if the program will terminate or a ‘0’ otherwise.”

The oracle consists of a perfect measuring device and a store, or memory, that contains a precise value—call it τ for Turing—of some physical quantity. (The memory might, for example, resemble a capacitor storing an exact amount of electricity.) The value of τ is an irrational number; its written representation would be an infinite string of binary digits, such as 0.00000001101…

The crucial property of τ is that its individual digits happen to represent accurately which programs terminate and which do not. So, for instance, if the integer representing a program were 8,735,439, then the oracle could by measurement obtain the 8,735,439th digit of τ (counting from left to right after the decimal point). If that digit were 0, the oracle would conclude that the program will process forever.

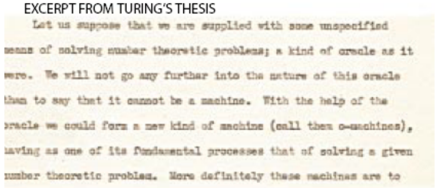
Obviously, without τ the oracle would be useless, and finding some physical variable in nature that takes this exact value might very well be impossible. So the search is on for some practicable way of implementing an oracle. If such a means were found, the impact on the field of computer science could be enormous. —*B.J.C. and D.P.*



COMPUTER PROGRAM

TOM MOORE

EXCERPT FROM TURING’S THESIS


Let us suppose that we are supplied with some unspecified means of solving number theoretic problems; a kind of oracle as it were. We will not go any further into the nature of this oracle than to say that it cannot be a machine. With the help of the oracle we could form a new kind of machine (call them o-machines), having as one of its fundamental processes that of solving a given number theoretic problem. More definitely these machines are to

PRINCETON ARCHIVES

for universal Turing machines. In his 1938 doctoral thesis at Princeton University, he described “a new kind of machine,” the “O-machine.”

An O-machine is the result of augmenting a universal Turing machine with a black box, or “oracle,” that is a mechanism for carrying out uncomputable tasks. In other respects, O-machines are similar to ordinary computers. A digitally encoded program is fed in, and the machine produces digital output from the input using a step-by-step procedure of repeated applications of the machine’s basic operations, one of which is to pass data to the oracle and register its response.

Turing gave no indication of how an oracle might work. (Neither did he explain in his earlier research how the basic actions of a universal Turing machine—for example, “identify the symbol in the scanner”—might take place.) But notional mechanisms that fulfill the specifications of an O-machine’s black box are not difficult to imagine [*see box above*]. In principle, even a suitable B-type network can compute the uncomputable, provided the activity of the neurons is desynchronized. (When a central clock keeps the neurons in step with one another, the functioning of the network can be exactly simulated by a universal Turing machine.)

Even among experts, Turing’s pioneering theoretical concept of a hypermachine has largely been forgotten.

In the exotic mathematical theory of hypercomputation, tasks such as that of distinguishing theorems from nontheorems in arithmetic are no longer uncomputable. Even a debugger that can tell whether any program written in C, for example, will enter an infinite loop is theoretically possible.

If hypercomputers can be built—and that is a big if—the potential for cracking logical and mathematical problems hitherto deemed intractable will be enormous. Indeed, computer science may be approaching one of its most significant advances since researchers wired together the first electronic embodiment of a universal Turing machine decades ago. On the other hand, work on hypercomputers may simply fizzle out for want of some way of realizing an oracle.

The search for suitable physical, chemical or biological phenomena is getting under way. Perhaps the answer will be complex molecules or other structures that link together in patterns as complicated as those discovered by Hanf and Myers. Or, as suggested by Jon Doyle of M.I.T., there may be naturally occurring equilibrating systems with discrete spectra that can be seen as carrying out, in principle, an uncomputable task, producing appropriate output (1 or 0, for example) after being bombarded with input.

Outside the confines of mathematical logic, Turing’s O-machines have largely been forgotten, and instead a myth has taken hold. According to this apocryphal account, Turing demonstrated in the mid-1930s that hypermachines are impossible. He and Alonzo Church, the logician who was Turing’s doctoral adviser at Princeton, are mistakenly credited with having enunciated a principle to the effect that a universal Turing machine can exactly simulate the behavior of any other information-processing machine. This proposition, widely but incorrectly known as the Church-Turing thesis, implies that no machine can carry out an information-processing task that lies beyond the scope of a universal Turing machine. In truth, Church and Turing claimed only that a universal Turing machine can match the behavior of any human mathematician working with paper and pencil in accordance with an algorithmic method—a considerably weaker claim that certainly does not rule out the possibility of hypermachines.

Even among those who are pursuing the goal of building hypercomputers, Turing’s pioneering theoretical contributions have been overlooked. Experts routinely talk of carrying out information processing “beyond the Turing limit” and describe themselves as attempting to “break the Turing barrier.” A recent review in *New Scientist* of this emerging field states that the new machines “fall outside Turing’s conception” and are “computers of a type never envisioned by Turing,” as if the British genius had not conceived of such devices more than half a century ago. Sadly, it appears that what has already occurred with respect to Turing’s ideas on connectionism is starting to happen all over again.

The Final Years

In the early 1950s, during the last years of his life, Turing pioneered the field of artificial life. He was trying to simulate a chemical mechanism by which the genes of a fertilized egg cell may determine the anatomical structure of the resulting animal or plant. He described this research as “not altogether unconnected” to his study of neural networks, because “brain structure has to be … achieved by the genetical embryological mechanism, and this theory that I am now working on may make clearer what restrictions this really implies.” During this period, Turing achieved the distinction of being the first to engage in the computer-assisted exploration of nonlinear dynamical systems. His theory used nonlinear differential equations to express the chemistry of growth.

But in the middle of this groundbreaking investigation, Turing died from cyanide poisoning, possibly by his own hand. On June 8, 1954, shortly before what would have been his 42nd birthday, he was found dead in his bedroom. He had left a large pile of handwritten notes and some computer programs. Decades later this fascinating material is still not fully understood.

The Authors

B. JACK COPELAND and DIANE PROUDFOOT are the directors of the Turing Project at the University of Canterbury, New Zealand, which aims to develop and apply Turing’s ideas using modern techniques. The authors are professors in the philosophy department at Canterbury, and Copeland is visiting professor of computer science at the University of Portsmouth in England. They have written numerous articles on Turing. Copeland’s *Turing’s Machines* and *The Essential Turing* are forthcoming from Oxford University Press, and his *Artificial Intelligence* was published by Blackwell in 1993. In addition to the logical study of hypermachines and the simulation of B-type neural networks, the authors are investigating the computer models of biological growth that Turing was working on at the time of his death. They are organizing a conference in London in May 2000 to celebrate the 50th anniversary of the pilot model of the Automatic Computing Engine, an electronic computer designed primarily by Turing.

Further Reading

X-Machines and the Halting Problem: Building a Super-Turing Machine. Mike Stannett in *Formal Aspects of Computing*, Vol. 2, pages 331–341; 1990.

Intelligent Machinery. Alan Turing in *Collected Works of A. M. Turing: Mechanical Intelligence*. Edited by D. C. Ince. Elsevier Science Publishers, 1992.

Computation beyond the Turing Limit. Hava T. Siegelmann in *Science*, Vol. 268, pages 545–548; April 28, 1995.

On Alan Turing’s Anticipation of Connectionism. B. Jack Copeland and Diane Proudfoot in *Synthese*, Vol. 108, No. 3, pages 361–377; March 1996.

Turing’s O-Machines, Searle, Penrose and the Brain. B. Jack Copeland in *Analysis*, Vol. 58, No. 2, pages 128–138; 1998.

The Church-Turing Thesis. B. Jack Copeland in *The Stanford Encyclopedia of Philosophy*. Edited by Edward N. Zalta. Stanford University, ISSN 1095-5054. Available at http://plato.stanford.edu on the World Wide Web.

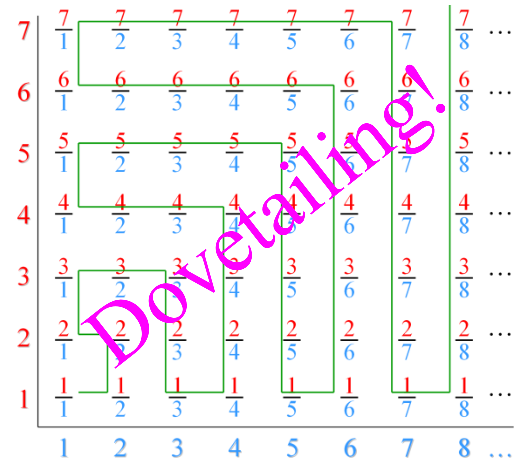
Theorem [Turing]: the set of algorithms is countable.

Proof: Sort algorithms ≡ programs by length:

1	↔	"main(){}"
⋮		⋮
9372	↔	"main(){int n; n=13;}"
⋮		⋮
10^{100}	↔	"<UNIX OS>"
⋮		⋮
10^{999}	↔	"<Windows Vista>"
⋮		⋮
$10^{10^{100}}$	↔	"<super intelligent program>"
⋮		⋮

Dovetailing!

Canonical order



⇒ set of algorithms is countable!

Theorem [Turing]: the set of functions is not countable.

Theorem: Boolean functions $\{f \mid f:\mathbb{N}\rightarrow\{0,1\}\}$ are uncountable.

Proof: Assume Boolean functions were countable; i.e., $\exists$ table containing all of f_i's and their corresponding values:

f_i	$f_i(1)$	$f_i(2)$	$f_i(3)$	$f_i(4)$	$f_i(5)$	$f_i(6)$	$f_i(7)$	$f_i(8)$	$f_i(9)$	
f_1	0	0	0	0	0	0	0	0	0	…
f_2	1	1	1	1	1	1	1	1	1	…
f_3	0	1	0	1	0	1	0	1	0	…
f_4	1	1	0	1	0	0	0	1	0	…
f_5	0	1	1	0	1	0	1	0	0	…
…	…	…	…	…	…	…	…	…	…	…
$f'(i) =$	1	0	1	0	0	…		$f':\mathbb{N}\rightarrow\{0,1\}$		

Real numbers!

Diagonalization

Non-existence proof!

But f' is missing from our table! $f' \neq f_k \ \forall\, k \in \mathbb{N}$

$\Rightarrow$ table is not a 1-1 correspondence between $\mathbb{N}$ and f_i's

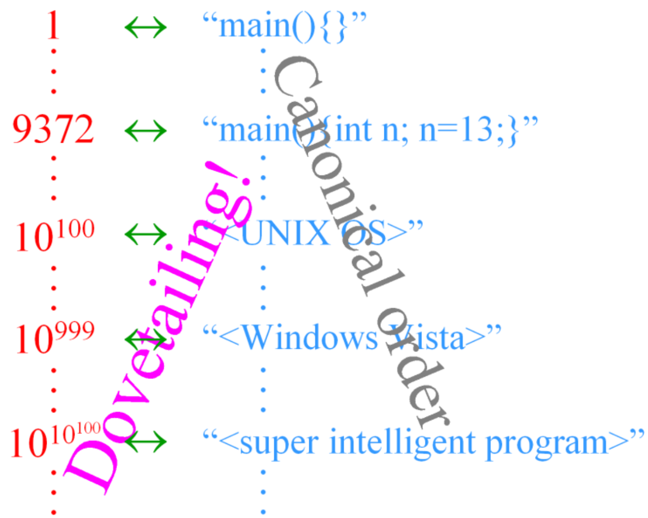
$\Rightarrow$ contradiction $\Rightarrow \{f \mid f:\mathbb{N}\rightarrow\{0,1\}\}$ is not countable!

$\Rightarrow$ There are more Boolean functions than natural numbers!

Theorem: the set of algorithms is countable.

Theorem: the set of functions is uncountable.

Theorem: the Boolean functions are uncountable.



f_i	$f_i(1)$	$f_i(2)$	$f_i(3)$	$f_i(4)$	$f_i(5)$	$f_i(6)$	$f_i(7)$	$f_i(8)$	$f_i(9)$	
f_1	0	0	0	0	0	0	0	0	0	…
f_2	1	1	1	1	1	1	1	1	1	…
f_3	0	1	0	1	0	1	0	1	0	…
f_4	1	1	0	1	0	0	0	1	0	…
f_5	0	1	1	0	1	0	1	0	0	…
…	…	…	…	…	…	…	…	…	…	…

$f'(i) =$ 1 0 1 0 0 … $f':\mathbb{N}\rightarrow\{0,1\}$

Diagonalization

Non-existence proof!

Corollary: there are "more" functions than algorithms / programs.

Corollary: some functions are not computable by any algorithm!

Corollary: most functions are not computable by any algorithm!

Corollary: there are "more" Boolean functions than algorithms.

Corollary: some Boolean functions on $\mathbb{N}$ are not computable.

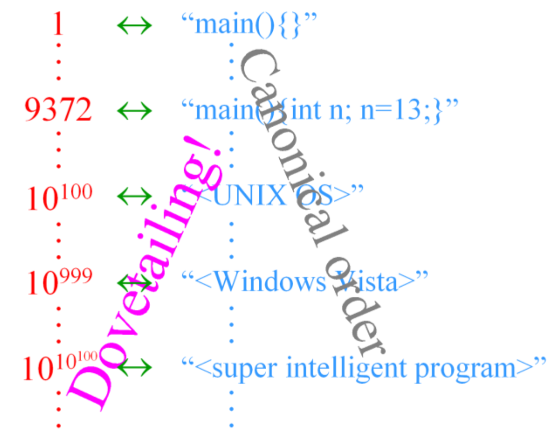
Corollary: most Boolean functions on $\mathbb{N}$ are not computable.

Theorem: most Boolean functions on ℕ are not computable.

Q: Can we find a concrete example of an uncomputable function?

A [Turing]: Yes, for example, the Halting Problem.

Definition: The Halting problem: given a program P and input I, will P halt if we ran it on I?



Define H:ℕ×ℕ→{0,1}

H(P,I)=1 if TM P halts on input I

H(P,I)=0 otherwise

Notes:

- P and I can be encoded as integers, in some canonical order.
- H is an everywhere-defined Boolean function on natural pairs.
- Alternatively, both P and I can be encoded as strings in Σ*.
- We can modify H to take only a single input: H'(2^P3^I) or H'(P$I)

Why 2^P3^I ?

What else will work?

Gödel numbering / encoding

Theorem [Turing]: the halting problem (H) is not computable.

Corollary: we can not algorithmically detect all infinite loops.

Q: Why not? E.g., do the following programs halt?

```
main()
{ int k=3; }
```

Halts!

```
main()
{ while(1) {} }
```

Runs forever!

Windows Vista

?

main()
{ Find a Fermat
triple $a^n+b^n=c^n$
with $n>2$ & stop}

Runs forever!

Open from 1637-1995!

main()
{ Find a Goldbach
integer that is not a sum
of two primes & stop}

?

Still open since 1742!

Theorem: solving the halting problem is at least as hard as solving arbitrary open mathematical problems!

Theorem [Turing]: the halting problem (H) is not computable.

Ex: the "3X+1" problem (the Ulam conjecture):

- Start with any integer X>0
- If X is even, then replace it with X/2
- If X is odd then replace it with 3X+1
- Repeat until X=1 (i.e., short cycle 4, 2, 1, ...)

Ex: 26 terminates after 10 steps

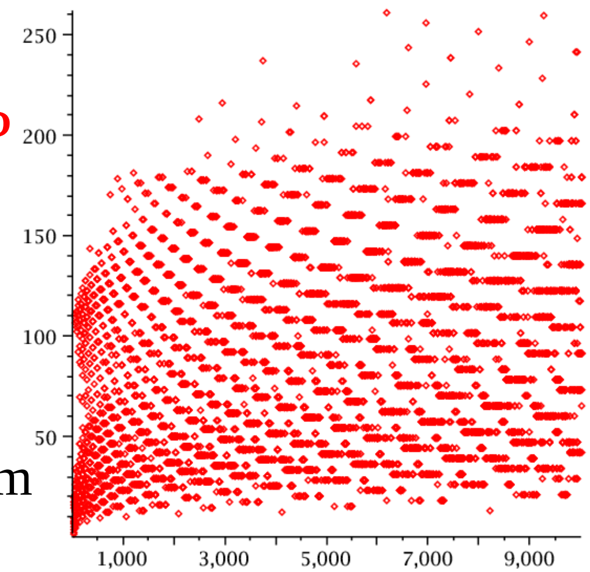
27 terminates after 111 steps

Termination verified for $X<10^{18}$

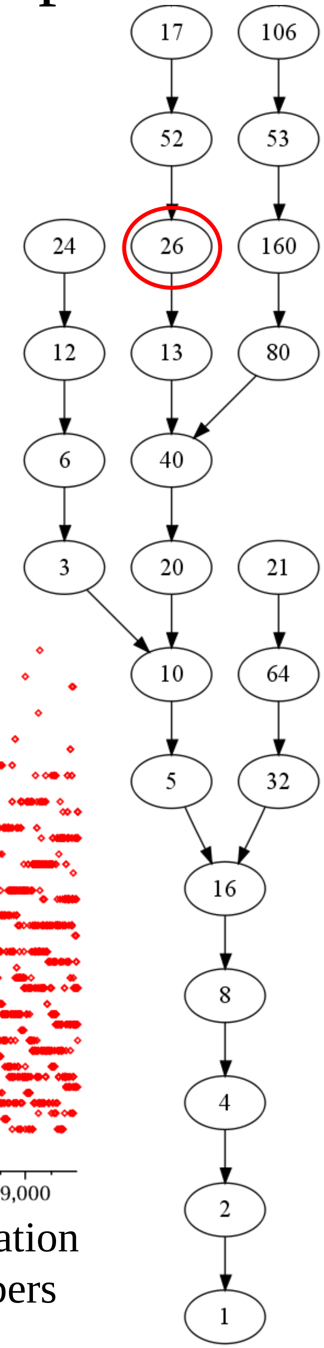
Q: Does this terminate for every X>0 ?

A: Open since 1937!

"Mathematics is not yet ready for such confusing, troubling, and hard problems." - Paul Erdős, who offered a $500 bounty for a solution to this problem



Number of steps to termination for the first 10,000 numbers



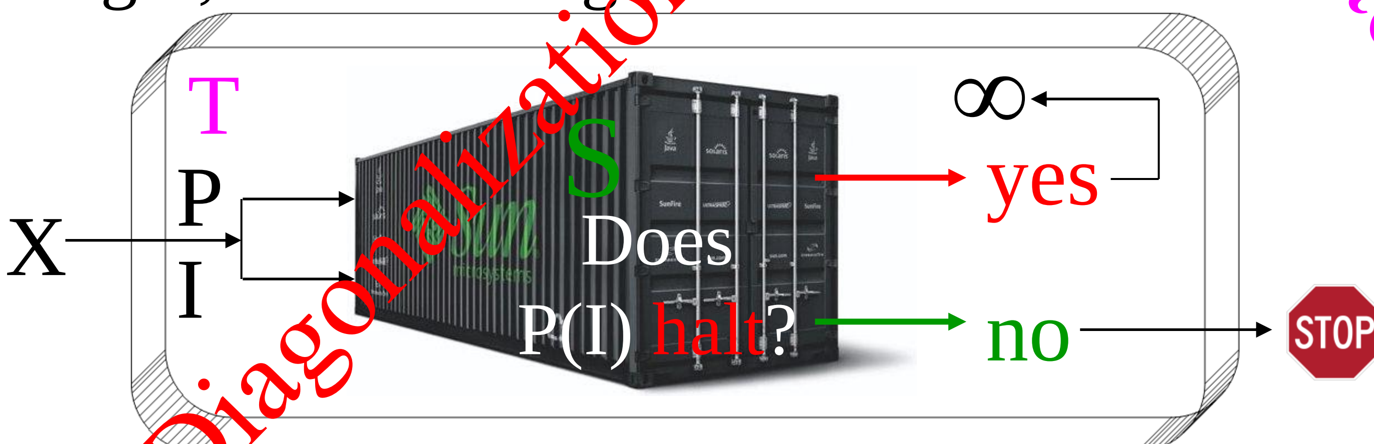
Observation: termination is in general difficult to detect!

Theorem [Turing]: the halting problem (H) is not computable.

Proof: Assume ∃ algorithm S that solves the halting problem H, that always stops with the correct answer for any P & I.

Non-existence proof!

Using S, construct algorithm / TM T:



Diagonalization

T(T) halts $\Rightarrow$ T(T) does not halt

T(T) does not halt $\Rightarrow$ T(T) halts

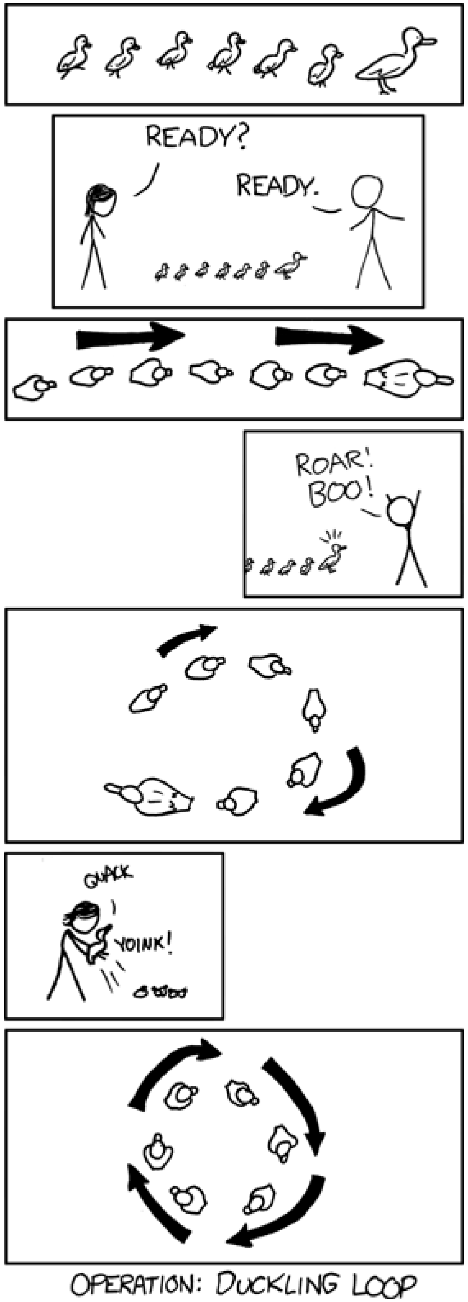
} Q $\Leftrightarrow$ ~Q $\Rightarrow$ Contradiction!

$\Rightarrow$ S cannot exist! (at least as an algorithm / program / TM)

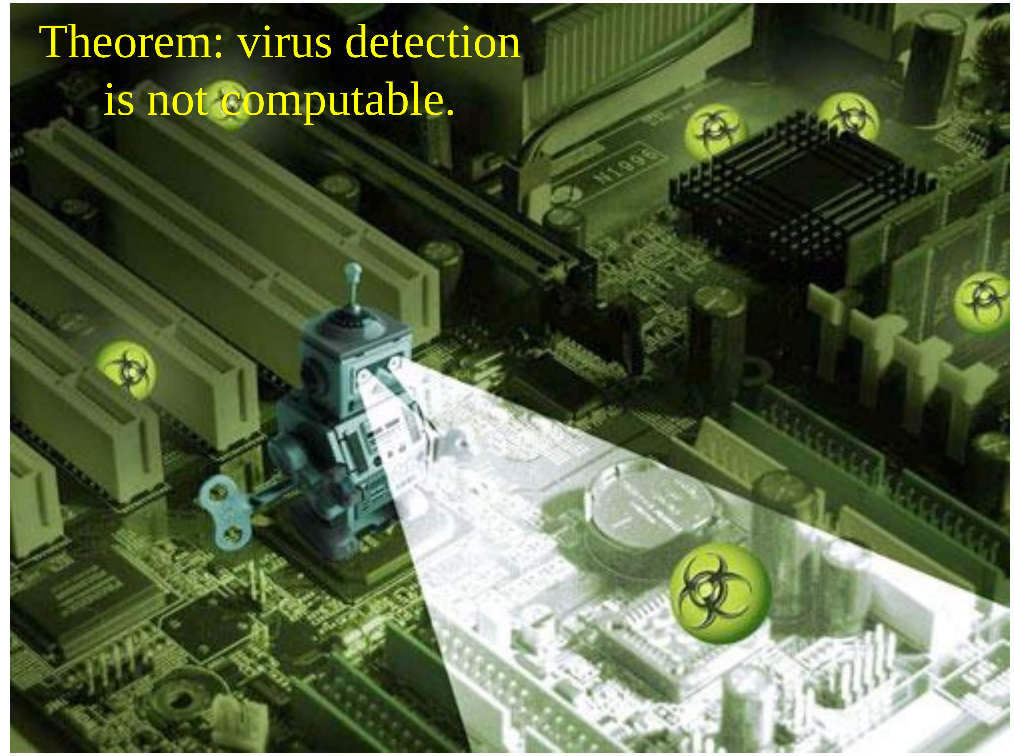
Q: When do we want to feed a program to itself in practice?
A: When we build compilers.

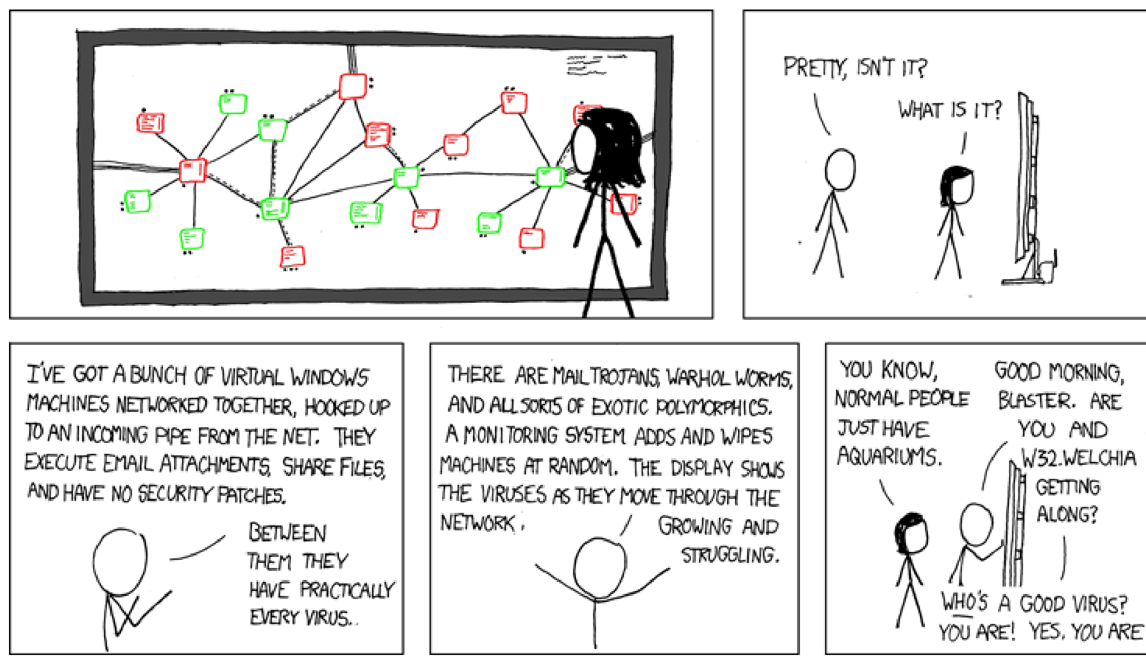
Q: Why?
A: To make them more efficient!
To boot-strap the coding in the compiler's own language!

Theorem: Infinite loop detection is not computable.

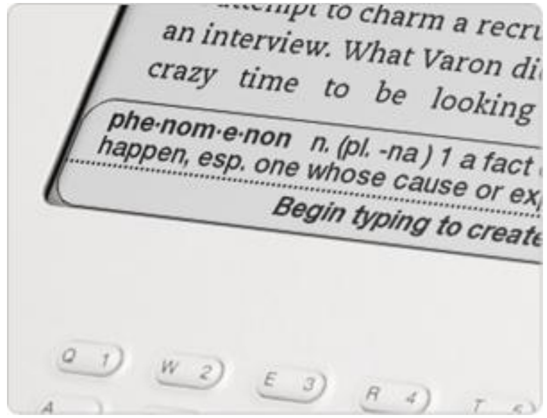


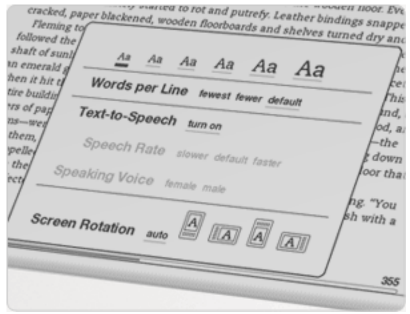
Theorem: virus detection is not computable.





One of My Favorite Turing Machines

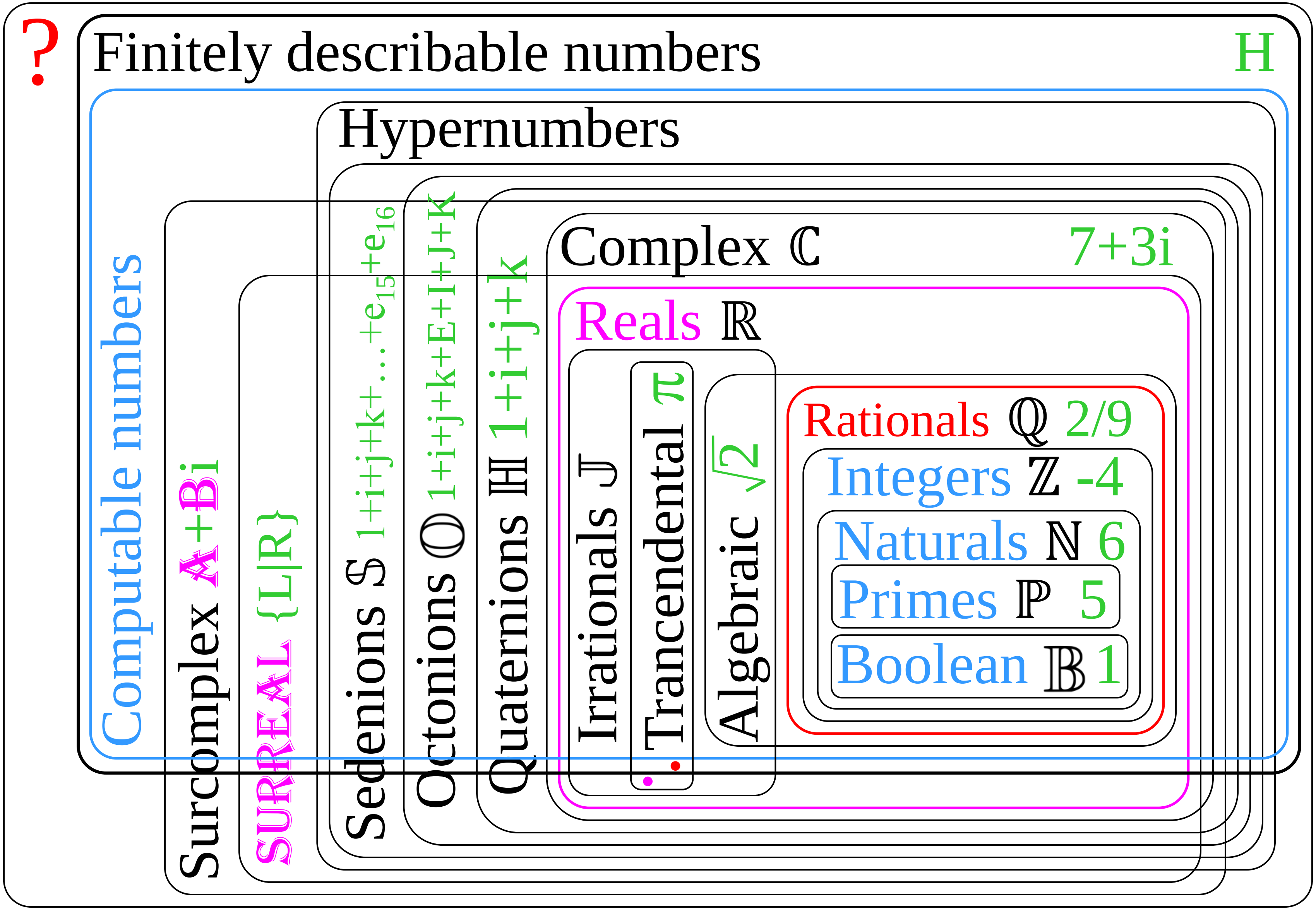




"Kindle DX" wireless reading device

- 1/3 of an inch thin, 4GB memory
- holds 3,500 books / documents
- 532 MHz ARM-11 processor
- 9.7" e-ink auto-rotate 824x1200 display
- Full PDF and text-to-speech
- 3G wireless, < 1 min / book
- 18.0 oz, battery life 4 days

Generalized Numbers



Theorem: some real numbers are not finitely describable!

Theorem: some finitely describable real numbers are not computable!

Theorem: Some real numbers are not finitely describable.

Proof: The number of finite descriptions is countable.

The number of real numbers is not countable.

$\Rightarrow$ Most real numbers do not have finite descriptions.

1 $\leftrightarrow$ “main(){}”

9372 $\leftrightarrow$ “main(){int n; n=13;}”

10^{100} $\leftrightarrow$ “<UNIX OS>”

10^{999} $\leftrightarrow$ “<Windows Vista>”

$10^{10^{100}}$ $\leftrightarrow$ “<super intelligent program>”

Dovetailing!

Canonical order

f_i	$f_i(1)$	$f_i(2)$	$f_i(3)$	$f_i(4)$	$f_i(5)$	$f_i(6)$	$f_i(7)$	$f_i(8)$	$f_i(9)$	
f_1	0	0	0	0	0	0	0	0	0	…
f_2	1	1	1	1	1	1	1	1	1	…
f_3	0	1	0	1	0	1	0	1	0	…
f_4	1	1	0	1	0	0	0	1	0	…
f_5	0	1	1	0	1	0	1	0	0	…
…	…	…	…	…	…	…	…	…	…	…
$f'(i)$ =	1	0	1	0	0	…		$f':\mathbb{N}\rightarrow\{0,1\}$		

Diagonalization

Non-existence proof!

Gödel numbering / encoding

Theorem: Some finitely describable reals are not computable.

Proof: Let $h=0.H_1H_2H_3H_4\ldots$ where $H_i=1$ if $i=2^P3^I$ for some integers P&I, and TM P halts on input I, and $H_i=0$ otherwise. Clearly $0 < h < 1$ is a real number and is finitely describable. If h was computable, then we could exploit an algorithm that computes it into solving the halting problem, a contradiction.

$\Rightarrow$ h is not computable.

Reduction / transformation!

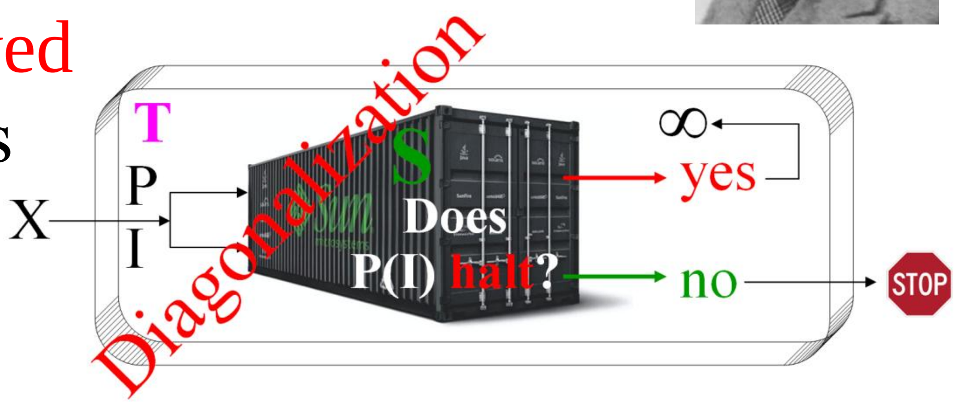
Theorem: all computable numbers are finitely describable.
Proof: A computable number can be outputted by a TM.
A TM is a (unique) finite description.

What the unsolvability of the Halting Problem means:

There is no single algorithm / program / TM that correctly solves all instances of the halting problem in finite time each.

This result does not necessarily apply if we allow:

- Incorrectness on some instances
- Infinitely large algorithm / program
- Infinite number of finite algorithms / programs
- Some instances to not be solved
- Infinite "running time" / steps
- Powerful enough oracles



Oracles

- Originated in Turing's Ph.D. thesis
- Named after the "Oracle of Apollo" at Delphi, ancient Greece

- Black-box subroutine / language
- Can compute arbitrary functions
- Instant computations "for free"
- Can greatly increase computation power of basic TMs

E.g., oracle for halting problem

The “Oracle of Omaha”

The “Oracle” of the Matrix

Turing Machines with Oracles

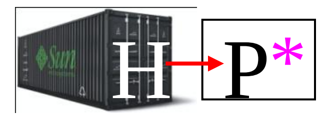
- A special case of "hyper-computation"
- Allows "what if" analysis: assumes certain undecidable languages can be recognized
- An oracle can profoundly impact the decidability & tractability of a language
- Any language / problem can be "relativized" WRT an arbitrary oracle
- Undecidability / intractability exists even for oracle machines!

Theorem [Turing]: Some problems are still not computable, even by Turing machines with an oracle for the halting problem!

Theorem [Turing]: the halting problem*(H*) is not computable.*

Proof: Assume ∃ algorithm S* that solves the halting problem H*, that always stops with the correct answer for any P* & I.

Add to P an H-oracle:

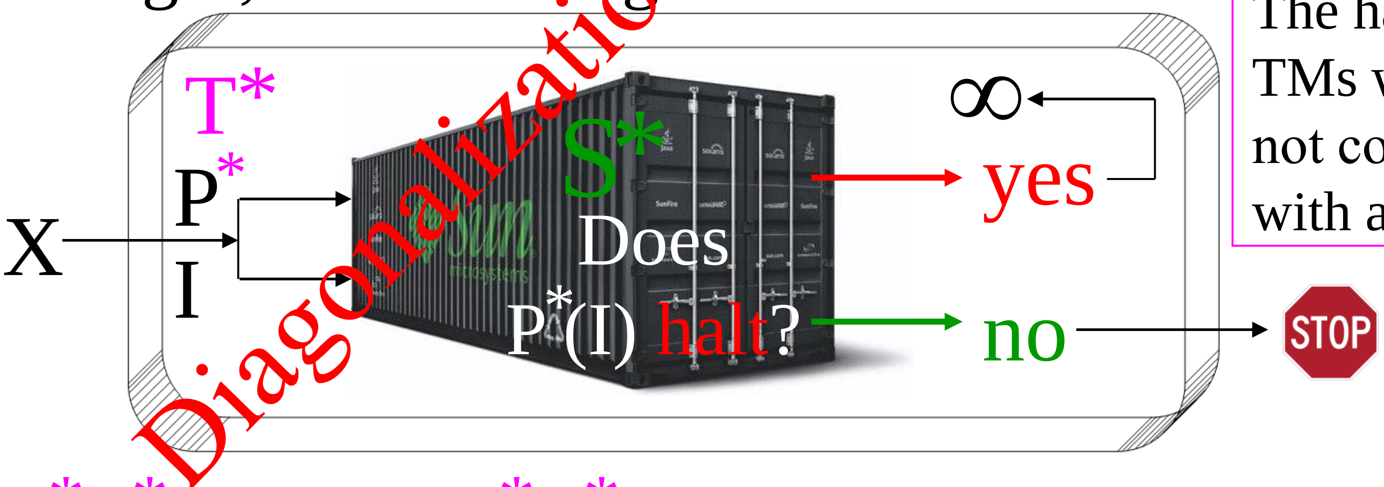


P* is "relativized" P.

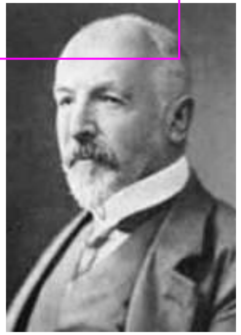
S* is "relativized" S.

T* is "relativized" T.

Using S, construct algorithm / TM T*:



The halting problem for TMs with an H-oracle is not computable by TM's with an H-oracle!



$$\left.\begin{array}{l} T^*(T^*) \text{ halts} \Rightarrow T^*(T^*) \text{ does not halt} \\ T^*(T^*) \text{ does not halt} \Rightarrow T^*(T^*) \text{ halts} \end{array}\right\} Q \Leftrightarrow \sim Q \Rightarrow \text{Contradiction!}$$

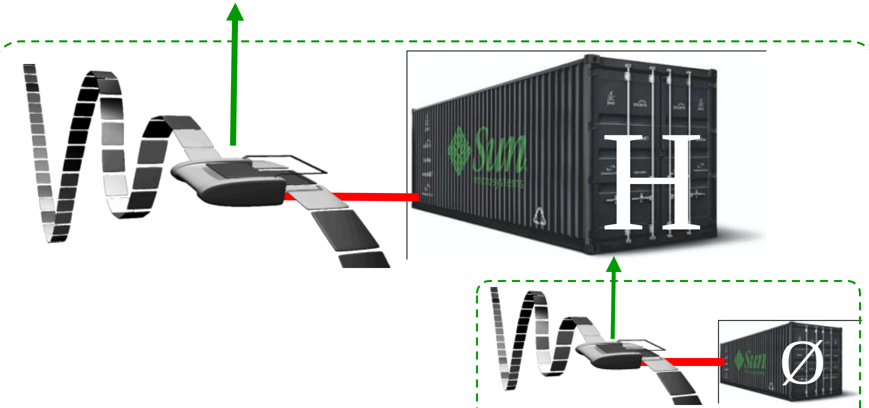
⇒ S* cannot exist! (at least as an algorithm / program / TM)

Turing Degrees

- Turing (1937); studied by Post (1944) and Kleene (1954)
- Quantifies the non-computability (i.e., algorithmic unsolvability) of (decision) problems and languages
- Some problems are "more unsolvable" than others!

Georg Cantor
1845-1918

- Defines computation "relative" to an oracle.
- "Relativized computation" - an infinite hierarchy!
- A "relativity theory of computation"!



Turing degree 2

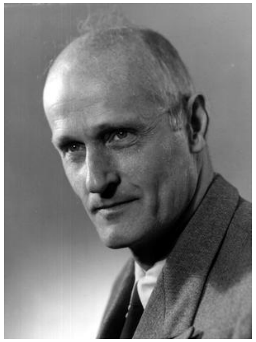
Turing degree 1

Turing degree 0

Students of Alonzo Church:

Alan Turing
1912-1954

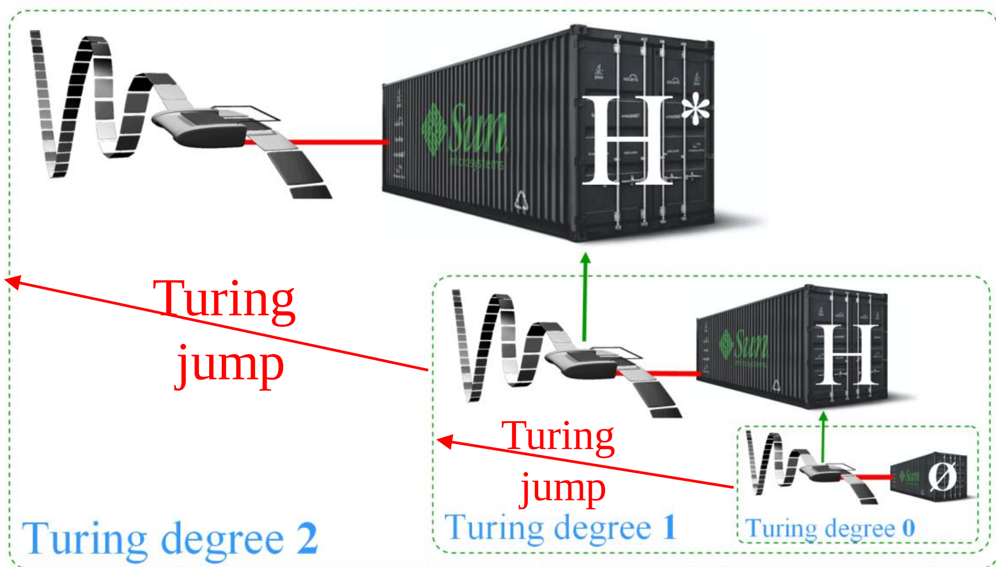
Emil Post
1897-1954



Stephen Kleene
1909-1994

Turing Degrees

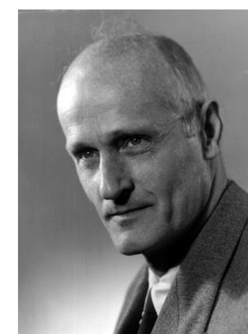
- Turing degree of a set X is the set of all Turing-equivalent (i.e., mutually-reducible) sets: an equivalence class [X]
- Turing degrees form a partial order / join-semilattice
- [0]: the unique Turing degree containing all computable sets
- For set X, the "Turing jump" operator X' is the set of indices of oracle TMs which halt when using X as an oracle
- [0']: Turing degree of the halting problem H; [0'']: Turing degree of the halting problem H* for TMs with oracle H.



Students of Alonzo Church:

Alan Turing
1912-1954

Emil Post
1897-1954

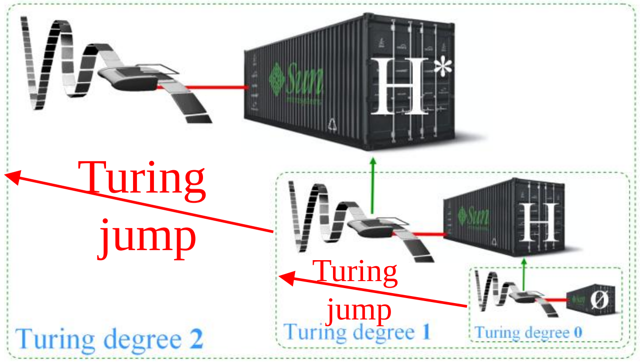


Stephen Kleene
1909-1994

Turing Degrees

- Each Turing degree is countably infinite (has exactly $\aleph_0$ sets)
- There are uncountably many ($2^{\aleph_0}$) Turing degrees
- A Turing degree X is strictly smaller than its Turing jump X'
- For a Turing degree X, the set of degrees smaller than X is countable; set of degrees larger than X is uncountable ($2^{\aleph_0}$)
- For every Turing degree X there is an incomparable degree (i.e., neither $X \geq Y$ nor $Y \geq X$ holds).
- There are $2^{\aleph_0}$ pairwise incomparable Turing degrees
- For every degree X, there is a degree D strictly between X and X' so that $X < D < X'$ (there are actually $\aleph_0$ of them)

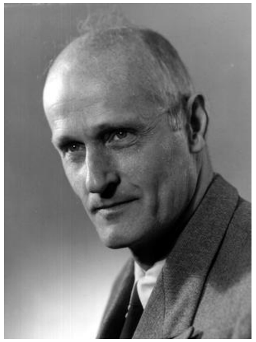
The structure of the Turing degrees semilattice is extremely complex!



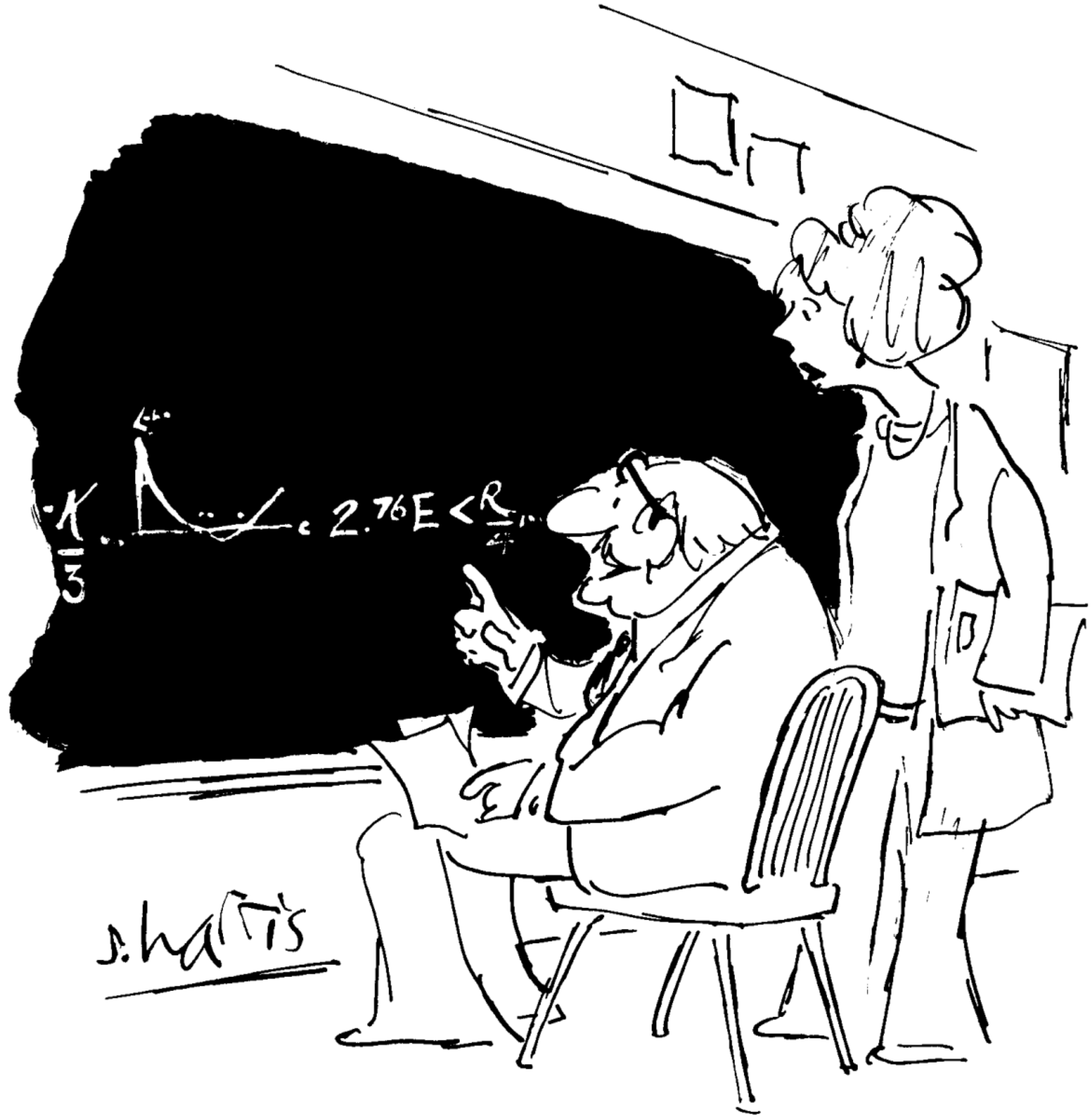
Students of Alonzo Church:

Alan Turing 1912-1954

Emil Post 1897-1954

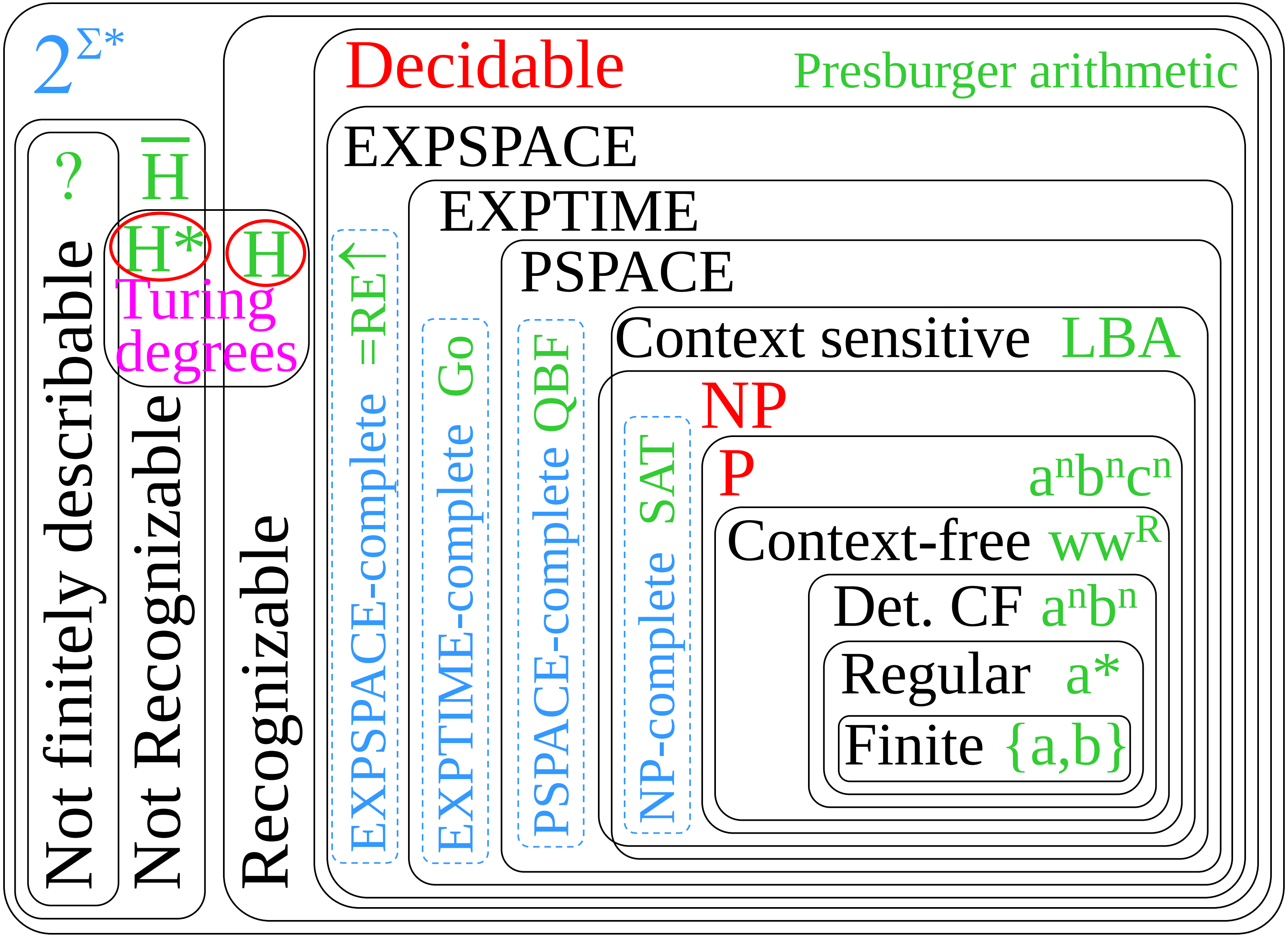


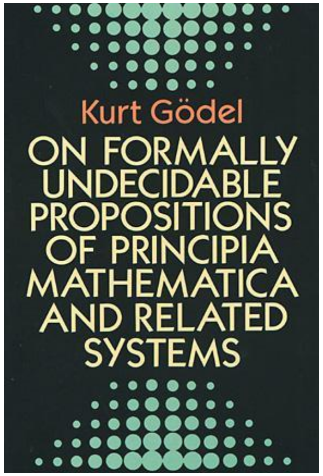
Stephen Kleene 1909-1994

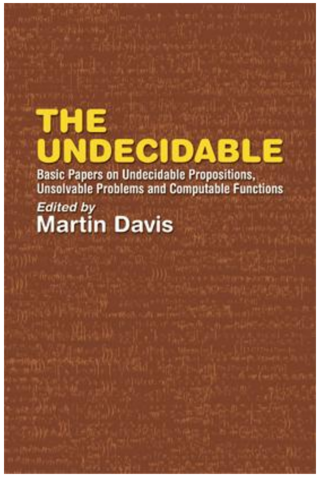


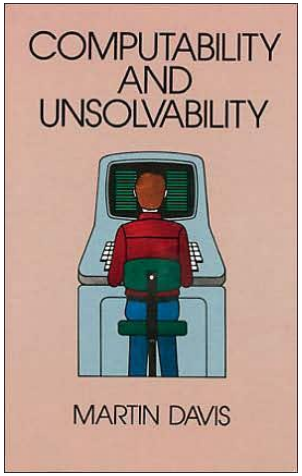
"THE BEAUTY OF THIS IS THAT IT IS ONLY OF THEORETICAL IMPORTANCE, AND THERE IS NO WAY IT CAN BE OF ANY PRACTICAL USE WHATSOEVER."

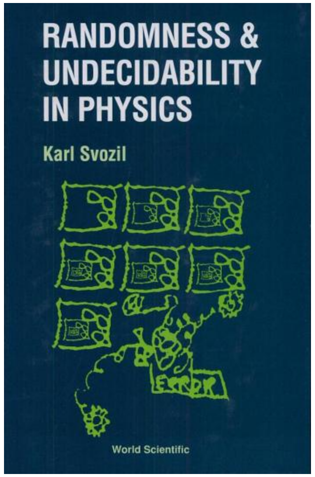
The Extended Chomsky Hierarchy

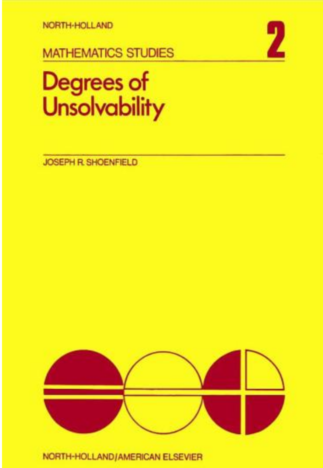


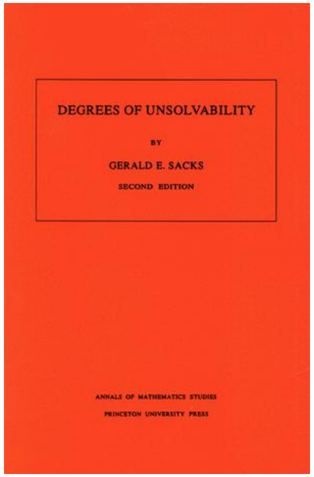

Kurt Gödel
ON FORMALLY UNDECIDABLE PROPOSITIONS OF PRINCIPIA MATHEMATICA AND RELATED SYSTEMS

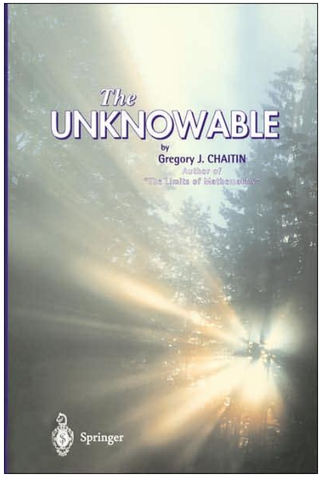

THE UNDECIDABLE
Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions
Edited by
Martin Davis

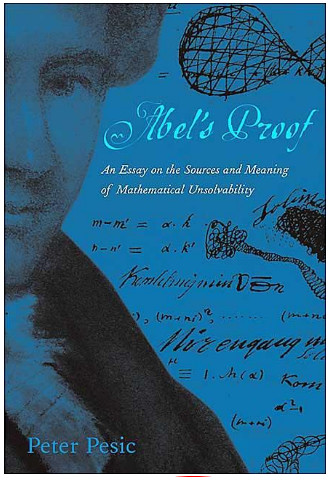

COMPUTABILITY AND UNSOLVABILITY
MARTIN DAVIS

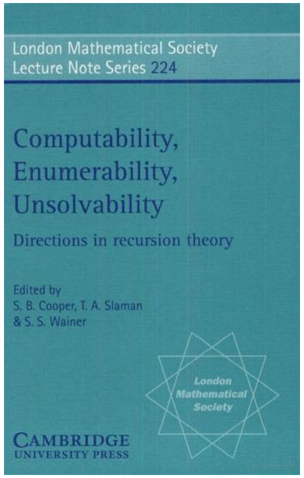

RANDOMNESS & UNDECIDABILITY IN PHYSICS
Karl Svozil
World Scientific

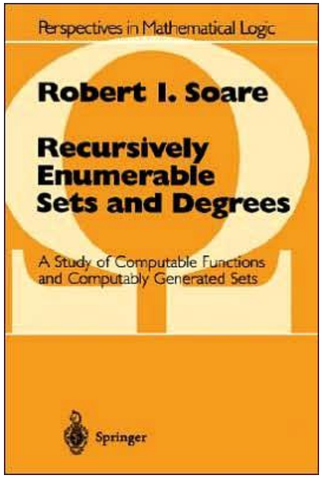

NORTH-HOLLAND
MATHEMATICS STUDIES
2
Degrees of Unsolvability
JOSEPH R. SHOENFIELD
NORTH-HOLLAND/AMERICAN ELSEVIER

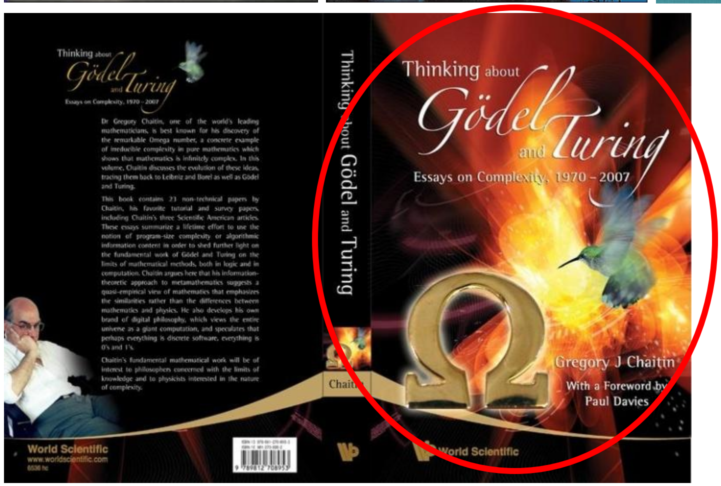

DEGREES OF UNSOLVABILITY
BY
GERALD E. SACKS
SECOND EDITION
ANNALS OF MATHEMATICS STUDIES
PRINCETON UNIVERSITY PRESS


The UNKNOWABLE
by
Gregory J. CHAITIN
Author of
"The Limits of Mathematics"
Springer


Abel's Proof
An Essay on the Sources and Meaning of Mathematical Unsolvability
Peter Pesic


London Mathematical Society
Lecture Note Series 224
Computability, Enumerability, Unsolvability
Directions in recursion theory
Edited by
S. B. Cooper, T. A. Slaman & S. S. Wainer
London Mathematical Society
CAMBRIDGE UNIVERSITY PRESS


Perspectives in Mathematical Logic
Robert I. Soare
Recursively Enumerable Sets and Degrees
A Study of Computable Functions and Computably Generated Sets
Springer


Thinking about Gödel and Turing
Essays on Complexity, 1970 – 2007
Gregory J Chaitin
With a Foreword by Paul Davies
World Scientific


CLAUDE PRESBYTERIAN CHURCH
THERE ARE SOME QUESTIONS THAT CAN'T BE ANSWERED BY GOOGLE
Citytv

Ideas on complexity and randomness originally suggested by Gottfried W. Leibniz in 1686, combined with modern information theory, imply that there can never be a "theory of everything" for all of mathematics

By Gregory Chaitin

The Limits of Reason

In 1956 *Scientific American* published an article by Ernest Nagel and James R. Newman entitled "Gödel's Proof." Two years later the writers published a book with the same title—a wonderful work that is still in print. I was a child, not even a teenager, and I was obsessed by this little book. I remember the thrill of discovering it in the New York Public Library. I used to carry it around with me and try to explain it to other children.

It fascinated me because Kurt Gödel used mathematics to show that mathematics itself has limitations. Gödel refuted the position of David Hilbert, who about a century ago declared that there was a theory of everything for math, a finite set of principles from which one could mindlessly deduce all mathematical truths by tediously following the rules of symbolic logic. But Gödel demonstrated that mathematics contains true statements that cannot be proved that way. His result is based on two self-referential paradoxes: "This statement is false" and "This statement is unprovable." (For more on Gödel's incompleteness theorem, see www.sciam.com/ontheweb)

My attempt to understand Gödel's proof took over my life, and now half a century later I have published a little book of my own. In some respects, it is my own version of Nagel and Newman's book, but it does not focus on Gödel's proof. The only things the two books have in common are their small size and their goal of critiquing mathematical methods.

Unlike Gödel's approach, mine is based on measuring information and showing that some mathematical facts cannot be compressed into a theory because they are too complicated. This new approach suggests that what Gödel

EXISTENCE OF OMEGA (Ω)—a specific, well-defined number that cannot be calculated by any computer program—smashes hopes for a complete, all-encompassing mathematics in which every true fact is true for a reason.

discovered was just the tip of the iceberg: an infinite number of true mathematical theorems exist that cannot be proved from any finite system of axioms.

Complexity and Scientific Laws

MY STORY BEGINS in 1686 with Gottfried W. Leibniz's philosophical essay *Discours de métaphysique* (*Discourse on Metaphysics*), in which he discusses how one can distinguish between facts that can be described by some law and those that are lawless, irregular facts. Leibniz's very simple and profound idea appears in section VI of the *Discours,* in which he essentially states that a theory has to be simpler than the data it explains, otherwise it does not explain anything. The concept of a law becomes vacuous if arbitrarily high mathematical complexity is permitted, because then one can always construct a law no matter how random and patternless the data really are. Conversely, if the only law that describes some data is an extremely complicated one, then the data are actually lawless.

Today the notions of complexity and simplicity are put in precise quantitative terms by a modern branch of mathematics called algorithmic information theory. Ordinary information theory quantifies information by asking how many bits are needed to encode the information. For example, it takes one bit to encode a single yes/no answer. Algorithmic information, in contrast, is defined by asking what size computer program is necessary to generate the data. The minimum number of bits—what size string of zeros and ones—needed to store the program is called the algorithmic information content of the data. Thus, the infinite sequence of numbers 1, 2, 3, … has very little algorithmic information; a very short computer program can generate all those numbers. It does not matter how long the program must take to do the computation or how much memory it must use—just the length of the program in bits counts. (I gloss over the question of what programming language is used to write the program—for a rigorous definition, the language would have to be specified precisely. Different programming languages would result in somewhat different values of algorithmic information content.)

To take another example, the number pi, 3.14159…, also has only a little algorithmic information content, because a relatively short algorithm can be programmed into a computer to compute digit after digit. In contrast, a random number with a mere million digits, say 1.341285…64, has a much larger amount of algorithmic information. Because the number lacks a defining pattern, the shortest program for outputting it will be about as long as the number itself:

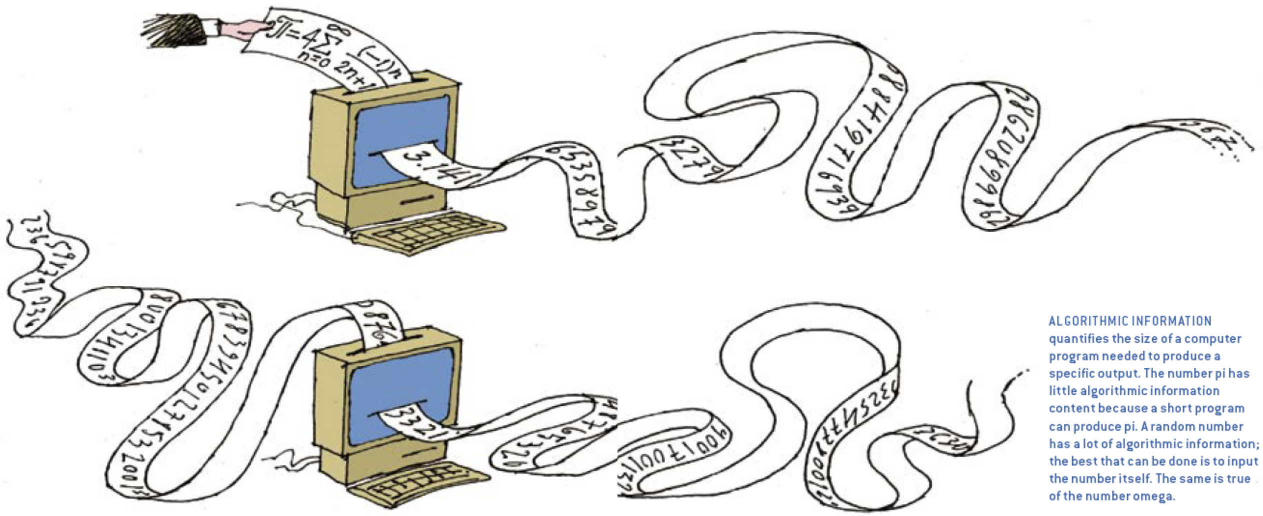
```
Begin
Print "1.341285…64"
End
```

(All the digits represented by the ellipsis are included in the program.) No smaller program can calculate that sequence of digits. In other words, such digit streams are incompressible, they have no redundancy; the best that one can do is transmit them directly. They are called irreducible or algorithmically random.

How do such ideas relate to scientific laws and facts? The basic insight is a software view of science: a scientific theory is like a computer program that predicts our observations, the experimental data. Two fundamental principles inform this viewpoint. First, as William of Occam noted, given two theories that explain the data, the simpler theory is to be preferred (Occam's razor). That is, the smallest program that calculates the observations is the best theory. Second is Leibniz's insight, cast in modern terms—if a theory is the same size in bits as the data it explains, then it is worthless, because even the most random of data has a theory of that size. A useful theory is a compression of the data; comprehension is compression. You compress things into computer programs, into concise algorithmic descriptions. The simpler the theory, the better you understand something.

Overview/*Irreducible Complexity*

- Kurt Gödel demonstrated that mathematics is necessarily incomplete, containing true statements that cannot be formally proved. A remarkable number known as omega reveals even greater incompleteness by providing an infinite number of theorems that cannot be proved by any finite system of axioms. A "theory of everything" for mathematics is therefore impossible.
- Omega is perfectly well defined [*see box on opposite page*] and has a definite value, yet it cannot be computed by any finite computer program.
- Omega's properties suggest that mathematicians should be more willing to postulate new axioms, similar to the way that physicists must evaluate experimental results and assert basic laws that cannot be proved logically.
- The results related to omega are grounded in the concept of algorithmic information. Gottfried W. Leibniz anticipated many of the features of algorithmic information theory more than 300 years ago.



ALGORITHMIC INFORMATION quantifies the size of a computer program needed to produce a specific output. The number pi has little algorithmic information content because a short program can produce pi. A random number has a lot of algorithmic information; the best that can be done is to input the number itself. The same is true of the number omega.

KENN BROWN; CONCEPT BY DUSAN PETRICIC (*preceding pages*); DUSAN PETRICIC (*above*)

Sufficient Reason

DESPITE LIVING 250 years before the invention of the computer program, Leibniz came very close to the modern idea of algorithmic information. He had all the key elements. He just never connected them. He knew that everything can be represented with binary information, he built one of the first calculating machines, he appreciated the power of computation, and he discussed complexity and randomness.

If Leibniz had put all this together, he might have questioned one of the key pillars of his philosophy, namely, the principle of sufficient reason—that everything happens for a reason. Furthermore, if something is true, it must be true for a reason. That may be hard to believe sometimes, in the confusion and chaos of daily life, in the contingent ebb and flow of human history. But even if we cannot always see a reason (perhaps because the chain of reasoning is long and subtle), Leibniz asserted, God can see the reason. It is there! In that, he agreed with the ancient Greeks, who originated the idea.

Mathematicians certainly believe in reason and in Leibniz's principle of sufficient reason, because they always try to prove everything. No matter how much evidence there is for a theorem, such as millions of demonstrated examples, mathematicians demand a proof of the general case. Nothing less will satisfy them.

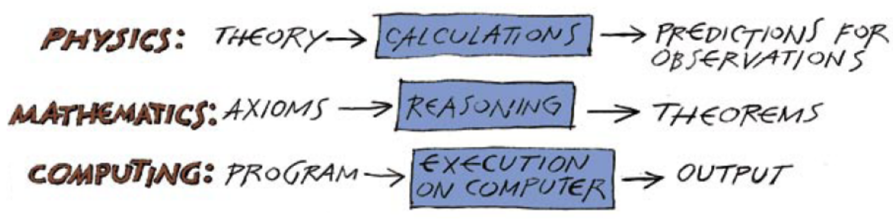
And here is where the concept of algorithmic information can make its surprising contribution to the philosophical discussion of the origins and limits of knowledge. It reveals that certain mathematical facts are true for no rea-

How Omega Is Defined

To see how the value of the number omega is defined, look at a simplified example. Suppose that the computer we are dealing with has only three programs that halt, and they are the bit strings 110, 11100 and 11110. These programs are, respectively, 3, 5 and 5 bits in size. If we are choosing programs at random by flipping a coin for each bit, the probability of getting each of them by chance is precisely $\frac{1}{2}^3$, $\frac{1}{2}^5$ and $\frac{1}{2}^5$, because each particular bit has probability ½. So the value of omega (the halting probability) for this particular computer is given by the equation:

$$\text{omega} = \tfrac{1}{2}^3 + \tfrac{1}{2}^5 + \tfrac{1}{2}^5 = .001 + .00001 + .00001 = .00110$$

This binary number is the probability of getting one of the three halting programs by chance. Thus, it is the probability that our computer will halt. Note that because program 110 halts we do not consider any programs that start with 110 and are larger than three bits—for example, we do not consider 1100 or 1101. That is, we do not add terms of .0001 to the sum for each of those programs. We regard all the longer programs, 1100 and so on, as being included in the halting of 110. Another way of saying this is that the programs are self-delimiting; when they halt, they stop asking for more bits. *—G.C.*



PHYSICS AND MATHEMATICS are in many ways similar to the execution of a program on a computer.

son, a discovery that flies in the face of the principle of sufficient reason.

Indeed, as I will show later, it turns out that an infinite number of mathematical facts are irreducible, which means no theory explains why they are true. These facts are not just computationally irreducible, they are logically irreducible. The only way to “prove” such facts is to assume them directly as new axioms, without using reasoning at all.

The concept of an “axiom” is closely related to the idea of logical irreducibility. Axioms are mathematical facts that we take as self-evident and do not try to prove from simpler principles. All formal mathematical theories start with axioms and then deduce the consequences of these axioms, which are called theorems. That is how Euclid did things in Alexandria two millennia ago, and his treatise on geometry is the classical model for mathematical exposition.

In ancient Greece, if you wanted to convince your fellow citizens to vote with you on some issue, you had to reason with them—which I guess is how the Greeks came up with the idea that in mathematics you have to prove things rather than just discover them experimentally. In contrast, previous cultures in Mesopotamia and Egypt apparently relied on experiment. Using reason has certainly been an extremely fruitful approach, leading to modern mathematics and mathematical physics and all that goes with them, including the technology for building that highly logical and mathematical machine, the computer.

So am I saying that this approach that science and mathematics has been following for more than two millennia crashes and burns? Yes, in a sense I am. My counterexample illustrating the limited power of logic and reason, my source of an infinite stream of unprovable mathematical facts, is the number that I call omega.

The Number Omega

THE FIRST STEP on the road to omega came in a famous paper published precisely 250 years after Leibniz's essay. In a 1936 issue of the *Proceedings of the London Mathematical Society*, Alan M. Turing began the computer age by presenting a mathematical model of a simple, general-purpose, programmable digital computer. He then asked, Can we determine whether or not a computer program will ever halt? This is Turing's famous halting problem.

Of course, by running a program you can eventually discover that it halts, if it halts. The problem, and it is an extremely fundamental one, is to decide when to give up on a program that does not halt. A great many special cases can be solved, but Turing showed that a general solution is impossible. No algorithm, no mathematical theory, can ever tell us which programs will halt and which will not. (For a modern proof of Turing's thesis, see www.sciam.com/ontheweb) By the way, when I say “program,” in modern terms I mean the concatenation of the computer program and the data to be read in by the program.

The next step on the path to the number omega is to consider the ensemble of all possible programs. Does a program chosen at random ever halt? The probability of having that happen is my omega number. First, I must specify how to pick a program at random. A program is simply a series of bits, so flip a coin to determine the value of each bit. How many bits long should the program be? Keep flipping the coin so long as the computer is asking for another bit of input. Omega is just the probability that the machine will eventually come to a halt when supplied with a stream of random bits in this fashion. (The precise numerical value of omega depends on the choice of computer programming language, but omega's surprising properties are not affected by this choice. And once you have chosen a language, omega has a definite value, just like pi or the number 3.)

Being a probability, omega has to be greater than 0 and less than 1, because some programs halt and some do not. Imagine writing omega out in binary. You would get something like 0.1110100.… These bits after the decimal point form an irreducible stream of bits. They are our irreducible mathematical facts (each fact being whether the bit is a 0 or a 1).

Omega can be defined as an infinite sum, and each N-bit program that halts contributes precisely $1/2^N$ to the sum [*see box on preceding page*]. In other words, each N-bit program that halts adds a 1 to the Nth bit in the binary expansion of omega. Add up all the bits for all programs that halt, and you would get the precise value of omega. This description may make it sound like you can calculate omega accurately, just as if it were the square root of 2 or the number pi. Not so—omega is perfectly well defined and it is a specific number, but it is impossible to compute in its entirety.

We can be sure that omega cannot be computed because knowing omega would let us solve Turing's halting problem, but we know that this problem is unsolvable. More specifically, knowing the first N bits of omega would enable you to decide whether or not each program up to N bits in size ever halts [*see box on page 80*]. From this it follows that you need at least an N-bit program to calculate N bits of omega.

Note that I am not saying that it is impossible to compute some digits of omega. For example, if we knew that computer programs 0, 10 and 110 all halt, then we would know that the first digits of omega were 0.111. The point is that the first N digits of omega cannot be computed using a program significantly shorter than N bits long.

Most important, omega supplies us with an infinite number of these irreducible bits. Given any finite program, no matter how many billions of bits long, we have an infinite number of bits that the program cannot compute. Given any finite set of axioms, we have an infinite number of truths that are unprovable in that system.

Because omega is irreducible, we can immediately conclude that a theory of everything for all of mathematics cannot exist. An infinite number of bits of omega constitute mathematical facts (whether each bit is a 0 or a 1) that cannot be derived from any principles simpler than the string of bits itself. Mathematics therefore has infinite complexity, whereas any individual theory of everything would have only finite complexity and could not capture all the richness of the full world of mathematical truth.

This conclusion does not mean that proofs are no good, and I am certainly not against reason. Just because some things are irreducible does not mean we should give up using reasoning. Irreducible principles—axioms—have always been a part of mathematics. Omega just shows that a lot more of them are out there than people suspected.

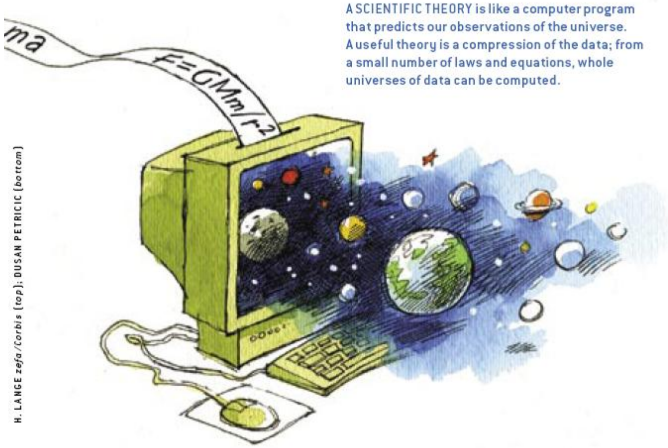
So perhaps mathematicians should not try to prove everything. Sometimes they should just add new axioms. That is what you have got to do if you are faced with irreducible facts. The problem is realizing that they are irreducible! In a way, saying something is irreducible is giving up, saying that it cannot ever be proved. Mathematicians would rather die than do that, in sharp contrast with their physicist colleagues, who are happy to be pragmatic and to use plausible reasoning instead of rigorous proof. Physicists are willing to add new principles, new scientific laws, to understand new domains of experience. This raises what I think is an extremely interesting question: Is mathematics like physics?

Mathematics and Physics

THE TRADITIONAL VIEW is that mathematics and physics are quite different. Physics describes the universe and depends on experiment and observation. The particular laws that govern our universe—whether Newton's laws of motion or the Standard Model of particle physics—must be determined empirically and then asserted like axioms that cannot be logically proved, merely verified.

Mathematics, in contrast, is somehow independent of the universe. Results and theorems, such as the properties of the integers and real numbers, do not depend in any way on the particular nature of reality in which we find ourselves. Mathematical truths would be true in any universe.

GOTTFRIED W. LEIBNIZ, commemorated by a statue in Leipzig, Germany, anticipated many of the features of modern algorithmic information theory more than 300 years ago.



A SCIENTIFIC THEORY is like a computer program that predicts our observations of the universe. A useful theory is a compression of the data; from a small number of laws and equations, whole universes of data can be computed.

H. LANGE zefa/Corbis (top); DUSAN PETRICIC (bottom)

THE AUTHOR

GREGORY CHAITIN is a researcher at the IBM Thomas J. Watson Research Center. He is also honorary professor at the University of Buenos Aires and visiting professor at the University of Auckland. He is co-founder, with Andrei N. Kolmogorov, of the field of algorithmic information theory. His nine books include the nontechnical works *Conversations with a Mathematician* (2002) and *Meta Math!* (2005). When he is not thinking about the foundations of mathematics, he enjoys hiking and snowshoeing in the mountains.

Yet both fields are similar. In physics, and indeed in science generally, scientists compress their experimental observations into scientific laws. They then show how their observations can be deduced from these laws. In mathematics, too, something like this happens—mathematicians compress their computational experiments into mathematical axioms, and they then show how to deduce theorems from these axioms.

If Hilbert had been right, mathematics would be a closed system, without room for new ideas. There would be a static, closed theory of everything for all of mathematics, and this would be like a dictatorship. In fact, for mathematics to progress you actually need new ideas and plenty of room for creativity. It does not suffice to grind away, mechanically deducing all the possible consequences of a fixed number of basic principles. I much prefer an open system. I do not like rigid, authoritarian ways of thinking.

Another person who thought mathematics is like physics was Imre Lakatos, who left Hungary in 1956 and later worked on philosophy of science in England. There Lakatos came up with a great word, "quasi-empirical," which means that even though there are no true experiments that can be carried out in mathematics, something similar does take place. For example, the Goldbach conjecture states that any even number greater than 2 can be expressed as the sum of two prime numbers. This conjecture was arrived at experimentally, by noting empirically that it was true for every even number that anyone cared to examine. The conjecture has not yet been proved, but it has been verified up to 10^{14}.

I think that mathematics is quasi-empirical. In other words, I feel that mathematics is different from physics (which is truly empirical) but perhaps not as different as most people think.

I have lived in the worlds of both mathematics and physics, and I never thought there was such a big difference between these two fields. It is a matter of degree, of emphasis, not an absolute difference. After all, mathematics and physics coevolved. Mathematicians should not isolate themselves. They should not cut themselves off from rich sources of new ideas.

New Mathematical Axioms

THE IDEA OF CHOOSING to add more axioms is not an alien one to mathematics. A well-known example is the parallel postulate in Euclidean geometry: given a line and a point not on the line, there is exactly one line that can be drawn through the point that never intersects the original line. For centuries geometers wondered whether that result could be proved using the rest of Euclid's axioms. It could not. Finally, mathematicians realized that they could substitute different axioms in place of the Euclidean version, thereby producing the non-Euclidean geometries of curved spaces, such as the surface of a sphere or of a saddle.

Why Is Omega Incompressible?

I wish to demonstrate that omega is incompressible—that one cannot use a program substantially shorter than *N* bits long to compute the first *N* bits of omega. The demonstration will involve a careful combination of facts about omega and the Turing halting problem that it is so intimately related to. Specifically, I will use the fact that the halting problem for programs up to length *N* bits cannot be solved by a program that is itself shorter than *N* bits (see www.sciam.com/ontheweb).

My strategy for demonstrating that omega is incompressible is to show that having the first *N* bits of omega would tell me how to solve the Turing halting problem for programs up to length *N* bits. It follows from that conclusion that no program shorter than *N* bits can compute the first *N* bits of omega. (If such a program existed, I could use it to compute the first *N* bits of omega and then use those bits to solve Turing's problem up to *N* bits—a task that is impossible for such a short program.)

Now let us see how knowing *N* bits of omega would enable me to solve the halting problem—to determine which programs halt—for all programs up to *N* bits in size. Do this by performing a computation in stages. Use the integer *K* to label which stage we are at: $K = 1, 2, 3, \ldots$

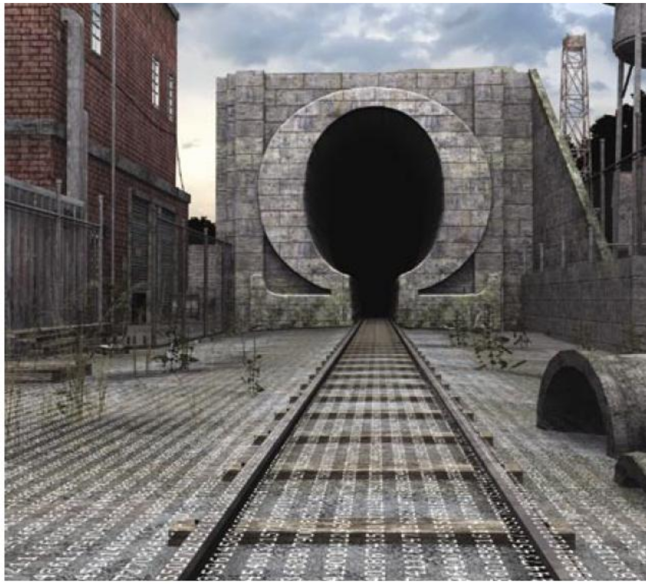
At stage *K*, run every program up to *K* bits in size for *K* seconds. Then compute a halting probability, which we will call omega_K, based on all the programs that halt by stage *K*. Omega_K will be less than omega because it is based on only a subset of all the programs that halt eventually, whereas omega is based on *all* such programs.

As *K* increases, the value of omega_K will get closer and closer to the actual value of omega. As it gets closer to omega's actual value, more and more of omega_K's first bits will be correct—that is, the same as the corresponding bits of omega.

And as soon as the first *N* bits are correct, you know that you have encountered every program up to *N* bits in size that will ever halt. (If there were another such *N*-bit program, at some later-stage *K* that program would halt, which would increase the value of omega_K to be greater than omega, which is impossible.)

So we can use the first *N* bits of omega to solve the halting problem for all programs up to *N* bits in size. Now suppose we could compute the first *N* bits of omega with a program substantially shorter than *N* bits long. We could then combine that program with the one for carrying out the omega_K algorithm, to produce a program shorter than *N* bits that solves the Turing halting problem up to programs of length *N* bits.

But, as stated up front, we know that no such program exists. Consequently, the first *N* bits of omega must require a program that is almost *N* bits long to compute them. That is good enough to call omega incompressible or irreducible. (A compression from *N* bits to almost *N* bits is not significant for large *N*.) *—G.C.*



OMEGA represents a part of mathematics that is in a sense unknowable. A finite computer program can reveal only a finite number of omega's digits; the rest remain shrouded in obscurity.

Other examples are the law of the excluded middle in logic and the axiom of choice in set theory. Most mathematicians are happy to make use of those axioms in their proofs, although others do not, exploring instead so-called intuitionist logic or constructivist mathematics. Mathematics is not a single monolithic structure of absolute truth!

Another very interesting axiom may be the "P not equal to NP" conjecture. P and NP are names for classes of problems. An NP problem is one for which a proposed solution can be verified quickly. For example, for the problem "find the factors of 8,633," one can quickly verify the proposed solution "97 and 89" by multiplying those two numbers. (There is a technical definition of "quickly," but those details are not important here.) A P problem is one that can be solved quickly even without being given the solution. The question is—and no one knows the answer—can every NP problem be solved quickly? (Is there a quick way to find the factors of 8,633?) That is, is the class P the same as the class NP? This problem is one of the Clay Millennium Prize Problems for which a reward of $1 million is on offer.

Computer scientists widely believe that P is not equal to NP, but no proof is known. One could say that a lot of quasi-empirical evidence points to P not being equal to NP. Should P not equal to NP be adopted as an axiom, then? In effect, this is what the computer science community has done. Closely related to this issue is the security of certain cryptographic systems used throughout the world. The systems are believed to be invulnerable to being cracked, but no one can prove it.

Experimental Mathematics

ANOTHER AREA of similarity between mathematics and physics is experimental mathematics: the discovery of new mathematical results by looking at many examples using a computer. Whereas this approach is not as persuasive as a short proof, it can be more convincing than a long and extremely complicated proof, and for some purposes it is quite sufficient.

In the past, this approach was defended with great vigor by both George Pólya and Lakatos, believers in heuristic reasoning and in the quasi-empirical nature of mathematics. This methodology is also practiced and justified in Stephen Wolfram's *A New Kind of Science* (2002).

Extensive computer calculations can be extremely persuasive, but do they render proof unnecessary? Yes and no. In fact, they provide a different kind of evidence. In important situations, I would argue that both kinds of evidence are required, as proofs may be flawed, and conversely computer searches may have the bad luck to stop just before encountering a counterexample that disproves the conjectured result.

All these issues are intriguing but far from resolved. It is now 2006, 50 years after this magazine published its article on Gödel's proof, and we still do not know how serious incompleteness is. We do not know if incompleteness is telling us that mathematics should be done somewhat differently. Maybe 50 years from now we will know the answer. SA

MORE TO EXPLORE

For a chapter on Leibniz, see **Men of Mathematics.** E. T. Bell. Reissue. Touchstone, 1986.

For more on a quasi-empirical view of math, see **New Directions in the Philosophy of Mathematics.** Edited by Thomas Tymoczko. Princeton University Press, 1998.

Gödel's Proof. Revised edition. E. Nagel, J. R. Newman and D. R. Hofstadter. New York University Press, 2002.

Mathematics by Experiment: Plausible Reasoning in the 21st Century. J. Borwein and D. Bailey. A. K. Peters, 2004.

For Gödel as a philosopher and the Gödel-Leibniz connection, see **Incompleteness: The Proof and Paradox of Kurt Gödel.** Rebecca Goldstein. W. W. Norton, 2005.

Meta Math!: The Quest for Omega. Gregory Chaitin. Pantheon Books, 2005.

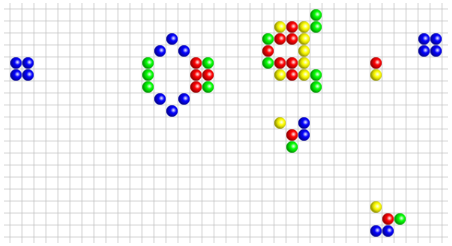
Short biographies of mathematicians can be found at **www-history.mcs.st-andrews.ac.uk/BiogIndex.html**

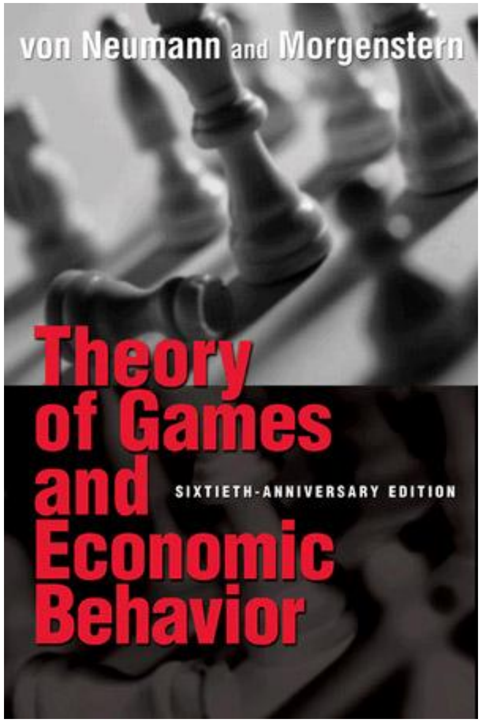
Gregory Chaitin's home page is **www.umcs.maine.edu/~chaitin/**

Historical Perspectives

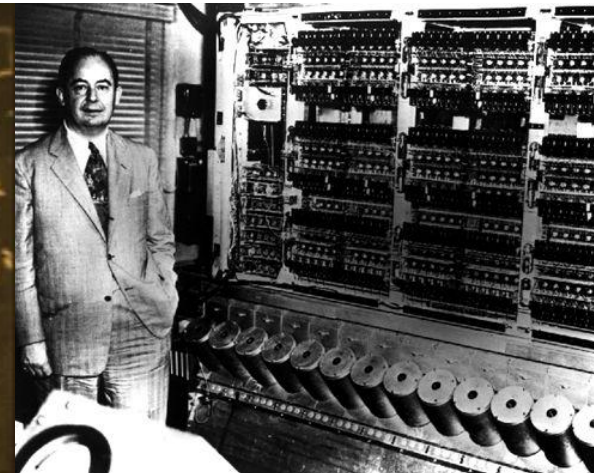
John von Neumann (1903-1957)

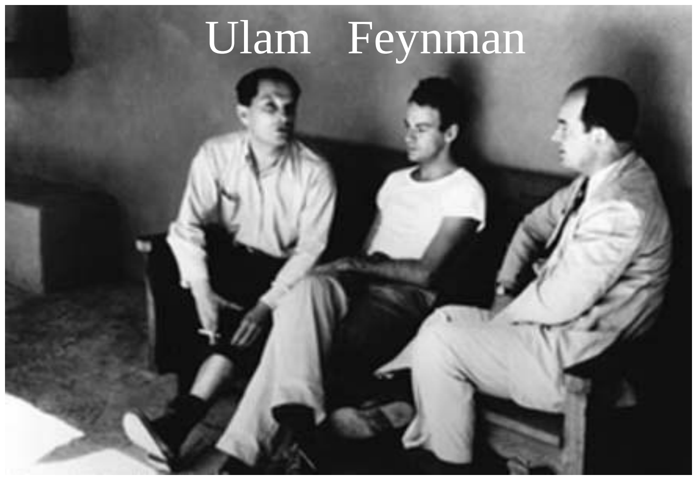
- Contributed to set theory, functional analysis, quantum mechanics, ergodic theory, economics, geometry, hydrodynamics, statistics, analysis, measure theory, ballistics, meteorology, …
- Invented game theory (used in Cold War)
- Re-axiomatized set theory
- Principal member of Manhattan Project
- Helped design the hydrogen / fusion bomb
- Pioneered modern computer science
- Originated the "stored program"
- "von Neumann architecture" and "bottleneck"
- Helped design & build the EDVAC computer
- Created field of cellular automata
- Investigated self-replication
- Invented merge sort

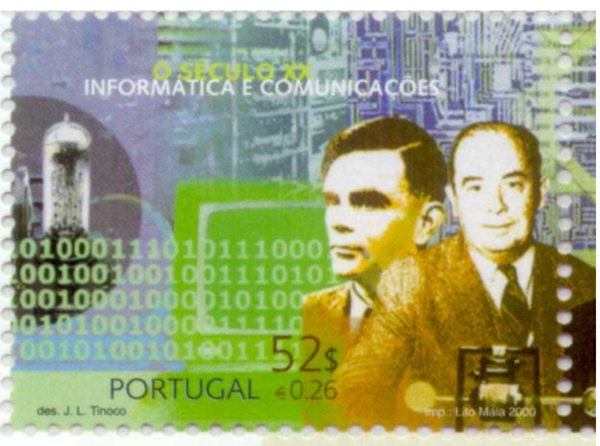




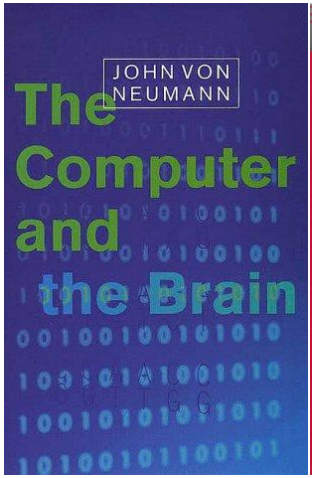


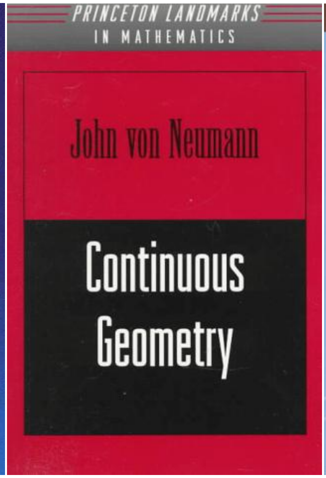


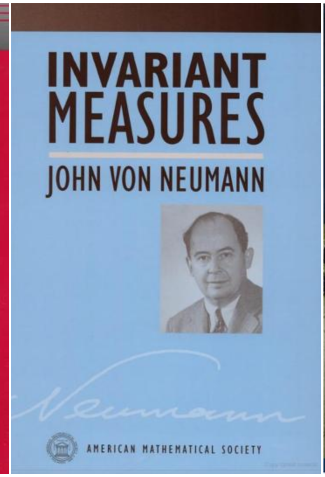



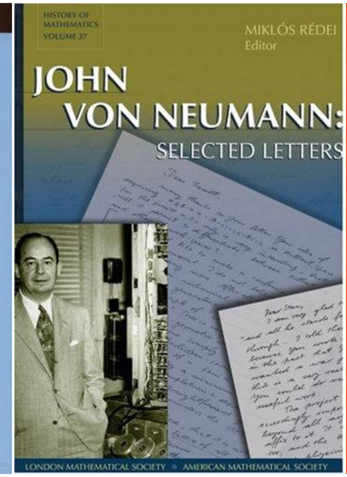


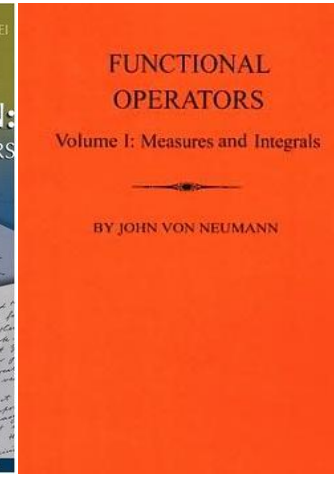
"Most mathematicians prove what they can; von Neumann proves what he wants."

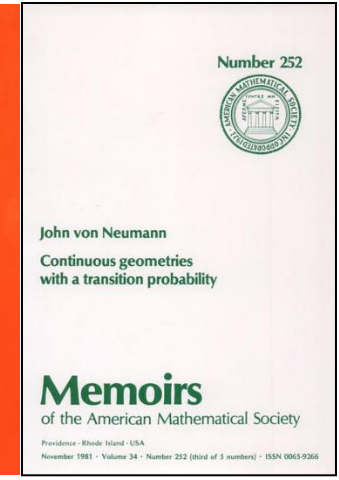

JOHN VON NEUMANN
The Computer and the Brain

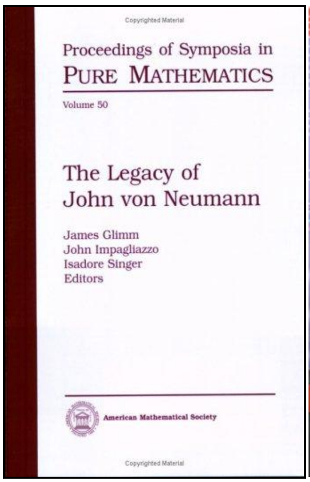

PRINCETON LANDMARKS IN MATHEMATICS
John von Neumann
Continuous Geometry


INVARIANT MEASURES
JOHN VON NEUMANN
AMERICAN MATHEMATICAL SOCIETY

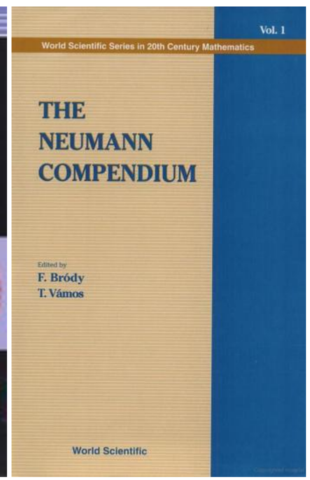

HISTORY OF MATHEMATICS VOLUME 27
MIKLÓS RÉDEI
Editor
JOHN VON NEUMANN:
SELECTED LETTERS
LONDON MATHEMATICAL SOCIETY • AMERICAN MATHEMATICAL SOCIETY

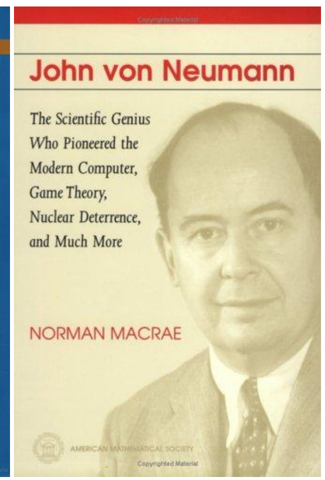

FUNCTIONAL OPERATORS
Volume I: Measures and Integrals
BY JOHN VON NEUMANN

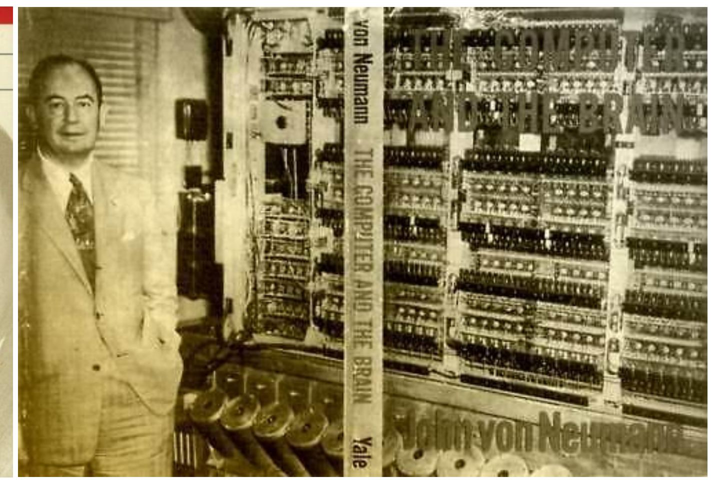

Number 252
John von Neumann
Continuous geometries with a transition probability
Memoirs
of the American Mathematical Society
Providence · Rhode Island · USA
November 1981 · Volume 34 · Number 252 (third of 5 numbers) · ISSN 0065-9266


Copyrighted Material
Proceedings of Symposia in
PURE MATHEMATICS
Volume 50
The Legacy of
John von Neumann
James Glimm
John Impagliazzo
Isadore Singer
Editors
American Mathematical Society
Copyrighted Material

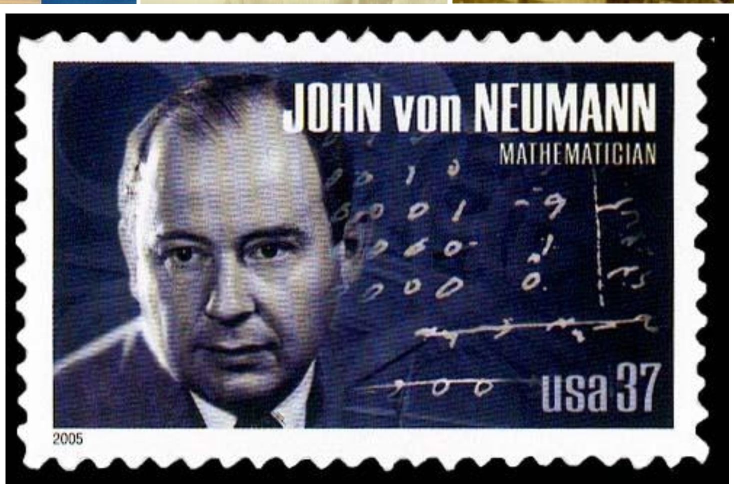

The Computer
from Pascal to von Neumann
Herman H. Goldstine

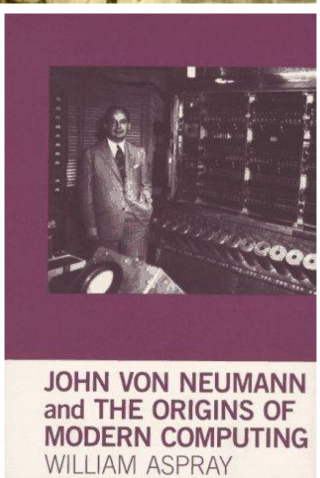

Vol. 1
World Scientific Series in 20th Century Mathematics
THE NEUMANN COMPENDIUM
Edited by
F. Bródy
T. Vámos
World Scientific


John von Neumann
The Scientific Genius
Who Pioneered the
Modern Computer,
Game Theory,
Nuclear Deterrence,
and Much More
NORMAN MACRAE


von Neumann
THE COMPUTER AND THE BRAIN
Yale

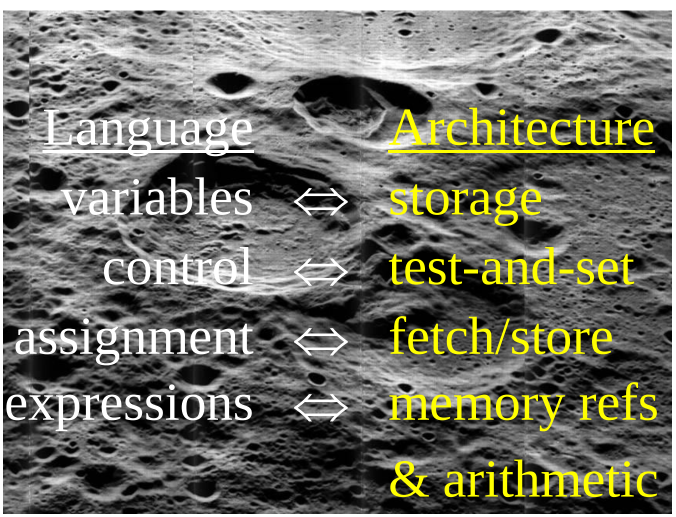

FELADO
LEVELEZŐLAP
MAGYAR POSTA
NEUMANN JÁNOS SZÁMÍTÓGÉP-TUDOMÁNYI TÁRSASÁG
INTERKOMPUTO
BUDAPEST 1982
KOMPUTO-SCIENCA SOCIETO JOHANO NEUMANN

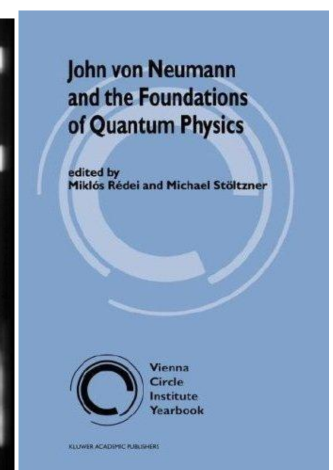

JOHN von NEUMANN
MATHEMATICIAN
usa 37
2005

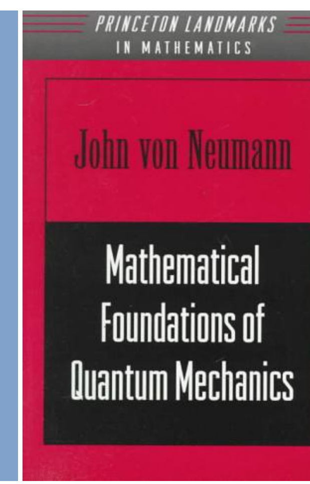

JOHN VON NEUMANN
and THE ORIGINS OF
MODERN COMPUTING
WILLIAM ASPRAY

von Neumann’s Legacy

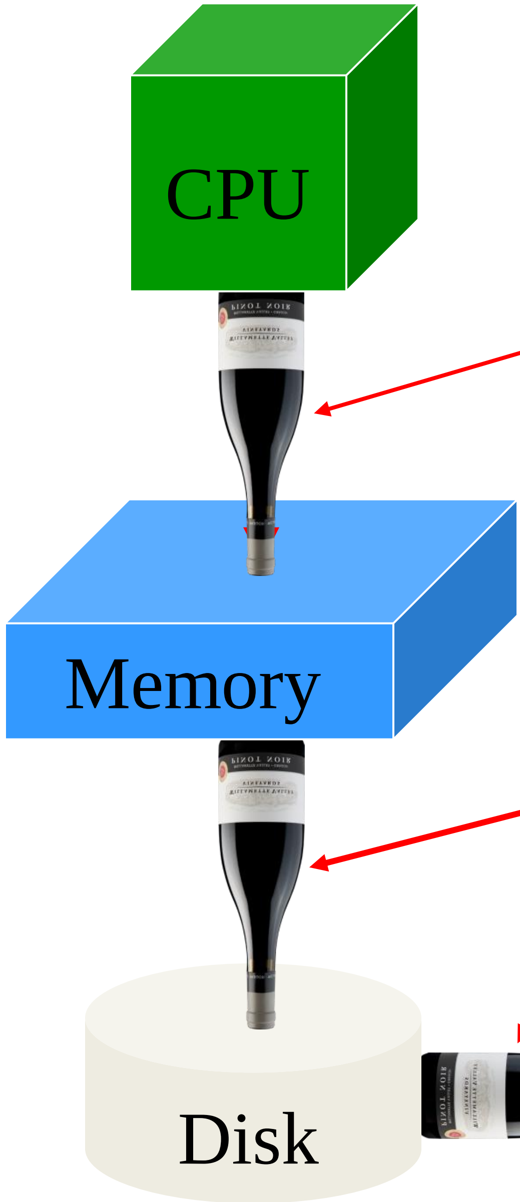
- Re-axiomatized set theory to address Russell’s paradox
- Independently proved Godel’s second incompleteness theorem: aximomatic systems are unable to prove their own consistency.
- Addressed Hilbert’s 6th problem: axiomatized quantum mechanics using Hilbert spaces.
- Developed the game-theory based Mutually-Assured Destruction (MAD) strategic equilibrium policy – still in effect today!
- von Neumann regular rings, von Neumann bicommutant theorem, von Neumann entropy, von Neumann programming languages







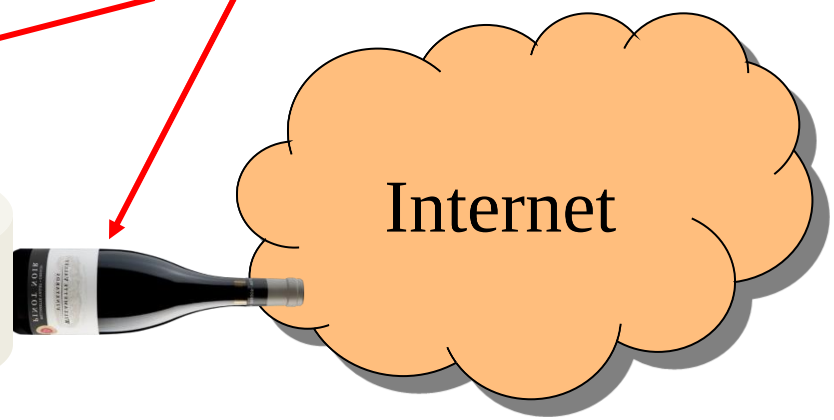
Von Neumann Architecture



“Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck. Not only is this tube a literal bottleneck for the data traffic of a problem, but, more importantly, it is an intellectual bottleneck that has kept us tied to word-at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand. Thus programming is basically planning and detailing the enormous traffic of words through the Von Neumann bottleneck, and much of that traffic concerns not significant data itself, but where to find it.”

- John Backus, 1977 ACM Turing Award lecture

More bottlenecks

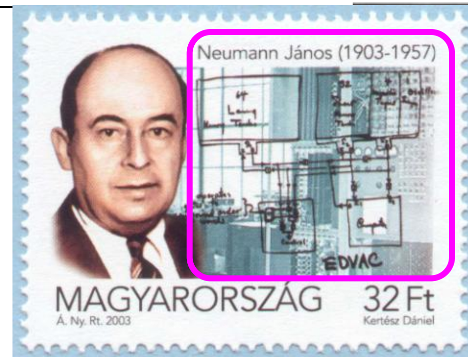


Functional programming

First Draft of a Report on the EDVAC

by

John von Neumann



Contract No. W–670–ORD–4926

Between the

United States Army Ordnance Department

and the

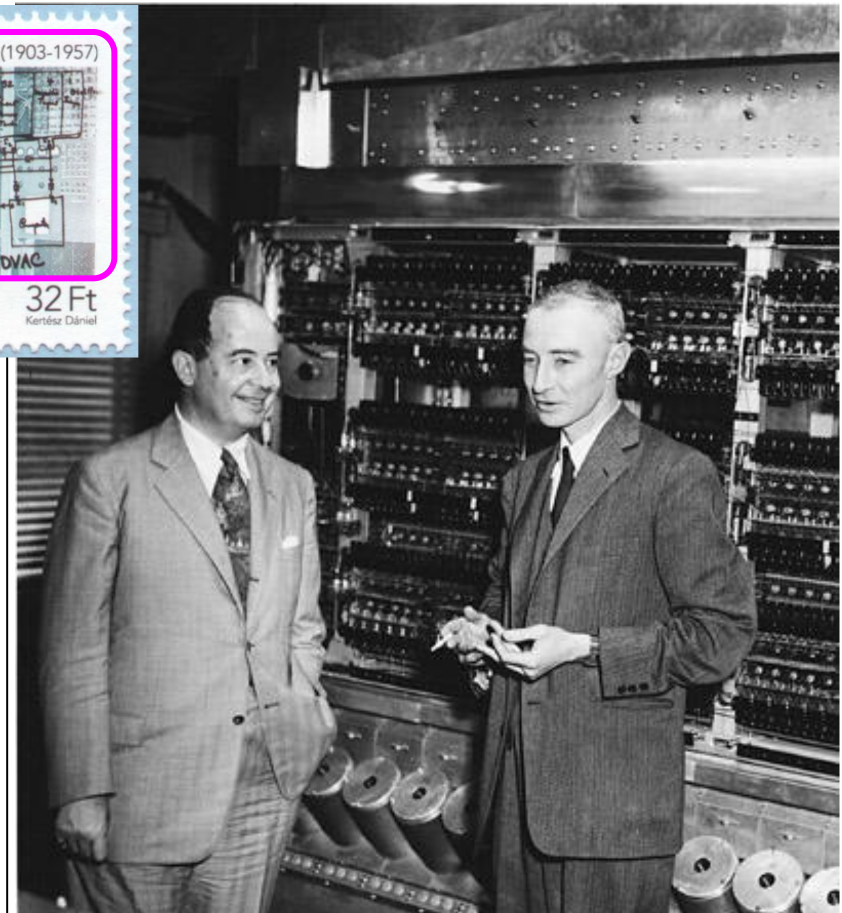
University of Pennsylvania

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945

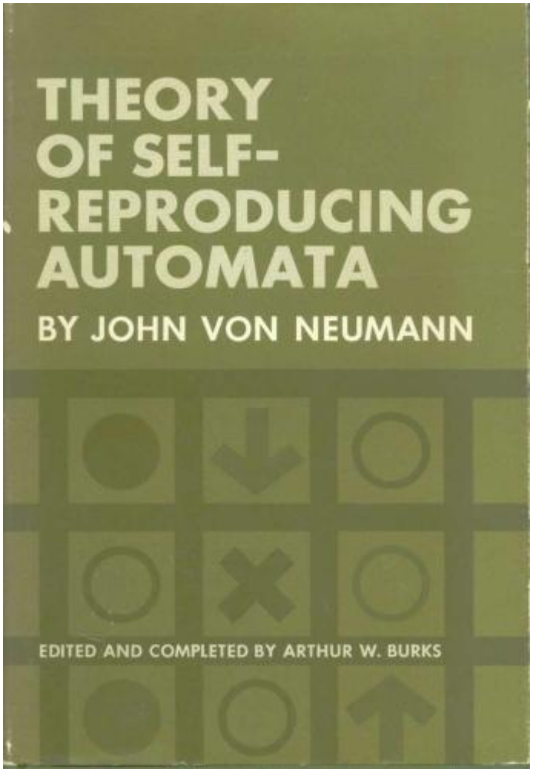
This is an exact copy of the original typescript draft as obtained from the University of Pennsylvania Moore School Library except that a large number of typographical errors have been corrected and the forward references that von Neumann had not filled in are provided where possible. Missing references, mainly to unwritten Sections after 15.0, are indicated by empty {}. All added material, mainly forward references, is enclosed in { }. The text and figures have been reset using TEX in order to improve readability. However, the original manuscript layout has been adhered to very closely. For a more "modern" interpretation of the von Neumann design see M. D. Godfrey and D. F. Hendry, "The Computer as von Neumann Planned It," *IEEE Annals of the History of Computing,* vol. 15 no. 1, 1993.

Michael D. Godfrey, Information Systems Laboratory, Electrical Engineering Department
Stanford University, Stanford, California, November 1992



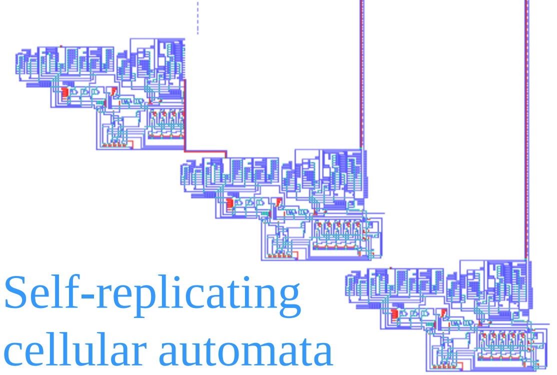
EDVAC (1945):

- 1024 words (44-bits) – 5.5KB
- 864 microsec / add (1157 / sec)
- 2900 microsec / multiply (345/sec)
- Magnetic tape (no disk), oscilloscope
- 6,000 vacuum tubes
- 56,000 Watts of power
- 17,300 lbs (7.9 tons), 490 sqft
- 30 people to operate

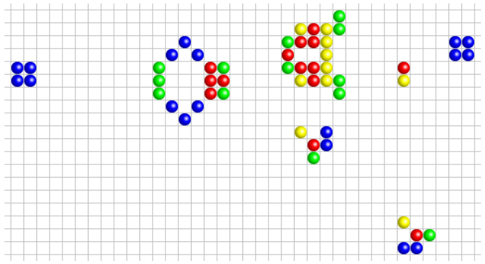


Self-Replication

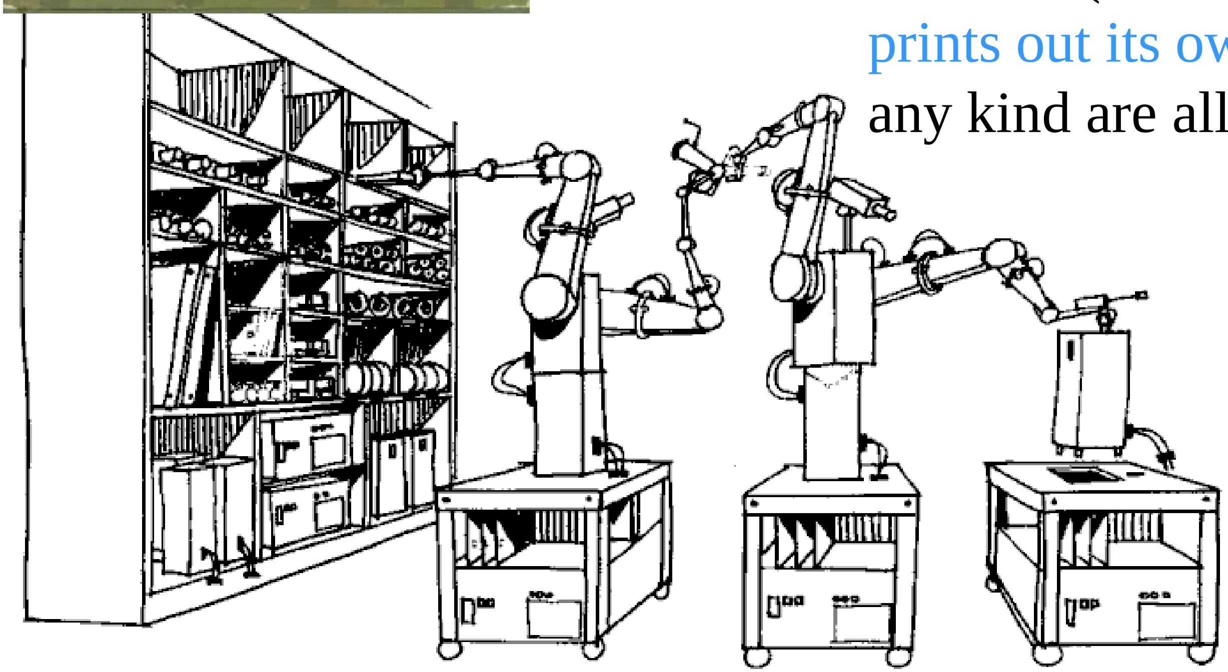
- Biology / DNA
- Nanotechnology
- Computer viruses
- Space exploration
- Memetics / memes
- "Gray goo"



Self-replicating cellular automata designed by von Neumann



Problem (extra credit): write a program that prints out its own source code (no inputs of any kind are allowed).



US005764518A

United States Patent [19]

Collins

[11] **Patent Number: 5,764,518**

[45] **Date of Patent: Jun. 9, 1998**

[54] **SELF REPRODUCING FUNDAMENTAL FABRICATING MACHINE SYSTEM**

[76] Inventor: **Charles M. Collins**, 10800 Oak Wilds Ct., Burke, Va. 22015

[21] Appl. No.: **757,005**

[22] Filed: **Nov. 25, 1996**

Related U.S. Application Data

[63] Continuation-in-part of Ser. No. 364,926, Dec. 28, 1994, Pat. No. 5,659,477.

[51] **Int. Cl.**^{6} **G06F 19/00**

[52] **U.S. Cl.** **364/468.01**; 364/468.24

[58] **Field of Search** 364/468.23, 468.22, 364/468.19, 468.01, 468.2, 468.21, 468.24, 474.21, 478.01, 478.03, 478.05, 478.06, 478.13–478.18, 424.028, 424.027, 424.07; 180/168, 8.1–8.7; 104/88.03, 88.04, 88.02; 901/6–8, 1; 318/568.12, 587; 395/80, 82, 901

[56] **References Cited**

U.S. PATENT DOCUMENTS

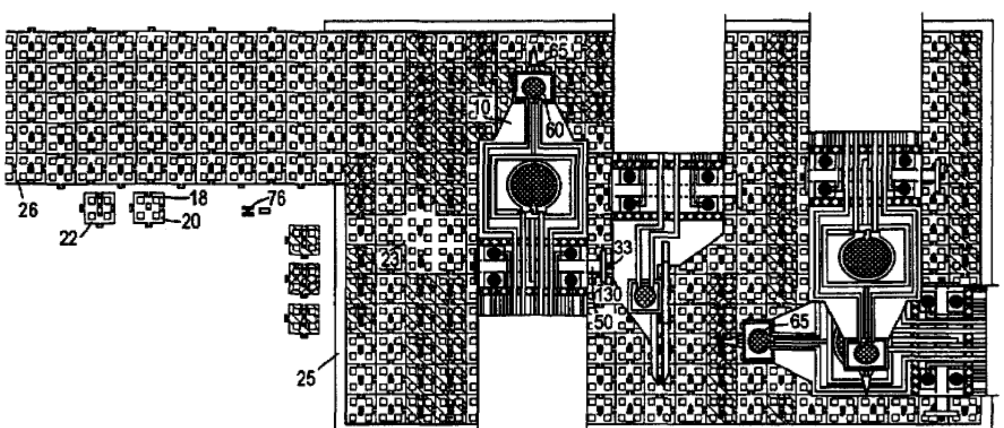
4,621,410	11/1986	Williamson	364/468.22 X
4,669,168	6/1987	Tamura et al.	901/7 X
4,870,592	9/1989	Lampi et al.	364/468.19
4,964,062	10/1990	Ubhayakar et al.	901/39 X
5,084,829	1/1992	Kato	901/7 X
5,145,130	9/1992	Purves	901/1 X
5,150,288	9/1992	Imai et al.	364/468.23 X
5,214,588	5/1993	Kaneko et al.	364/468.2

Primary Examiner—Joseph Ruggiero
Attorney, Agent, or Firm—Henry G. Kohlmann

[57] **ABSTRACT**

A system of units for constructing or replicating a means (**10.10.10**p) including means of diverse materials consisting of a plurality of pieces (**20.22.23**, **156–165**) having at least one indicia (**18**) thereon for detection thereof, at least one adjoining means functioning according to instructions of a computer program of a processor means for adjoining in any predetermined relation with other of the plurality of the pieces (**20**, **22**, **23**, **156–165**), and the processor means (**30**, **120**, **166**, **167**) having the computer program instructions being responsive to detection of the at least one indicia to provide for arranging the other of the plurality of the pieces in the predetermined relation for controlling the fabrication means in assembling a given number of the plurality of the pieces in the predetermined relation to comprise a produced fabrication means (**10**,**10**,**10**p) are selected from a group consisting of a puzzle piece system, a construction system, a hot knife system, a holed piece system.

75 Claims, 30 Drawing Sheets



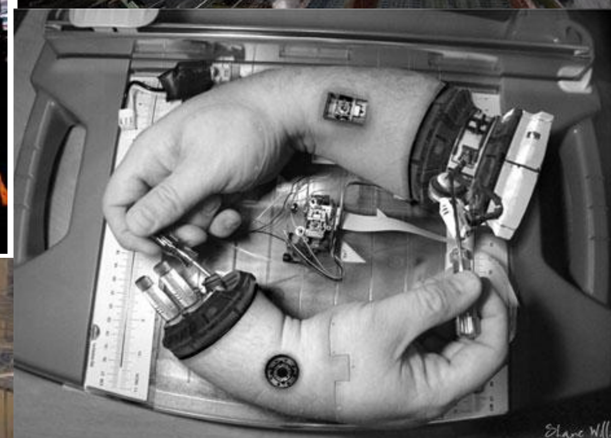
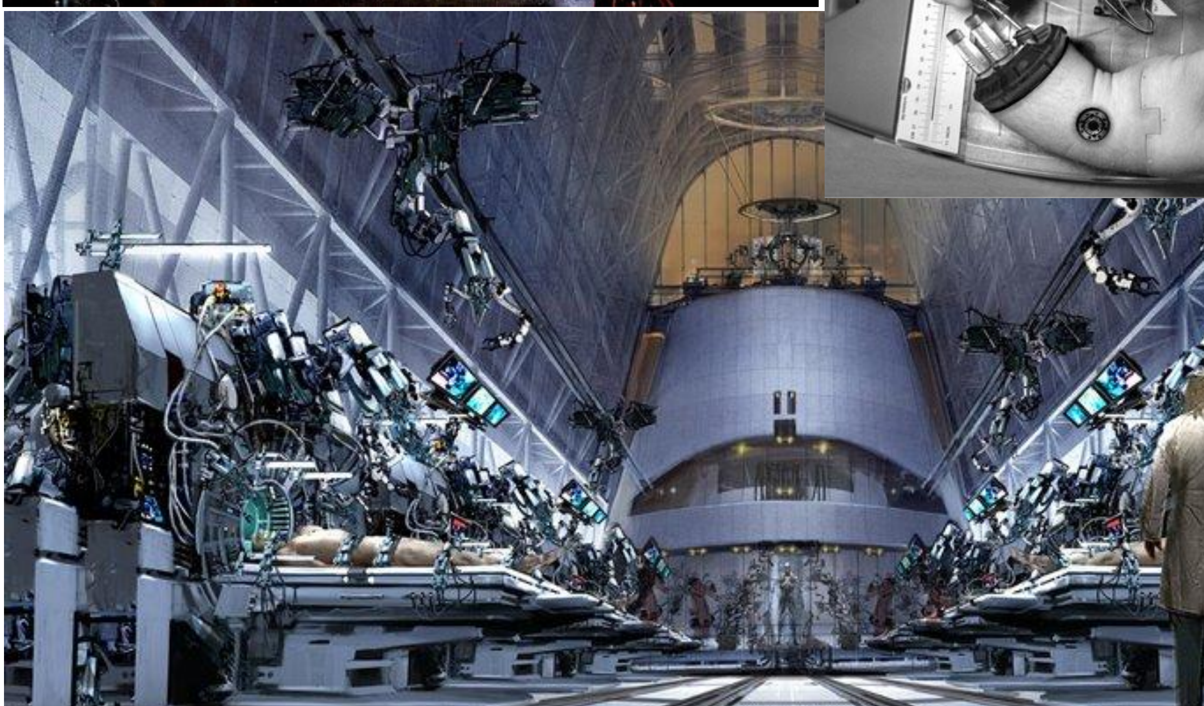
"In mathematics you don't understand things. You just get used to them."
– John von Neumann



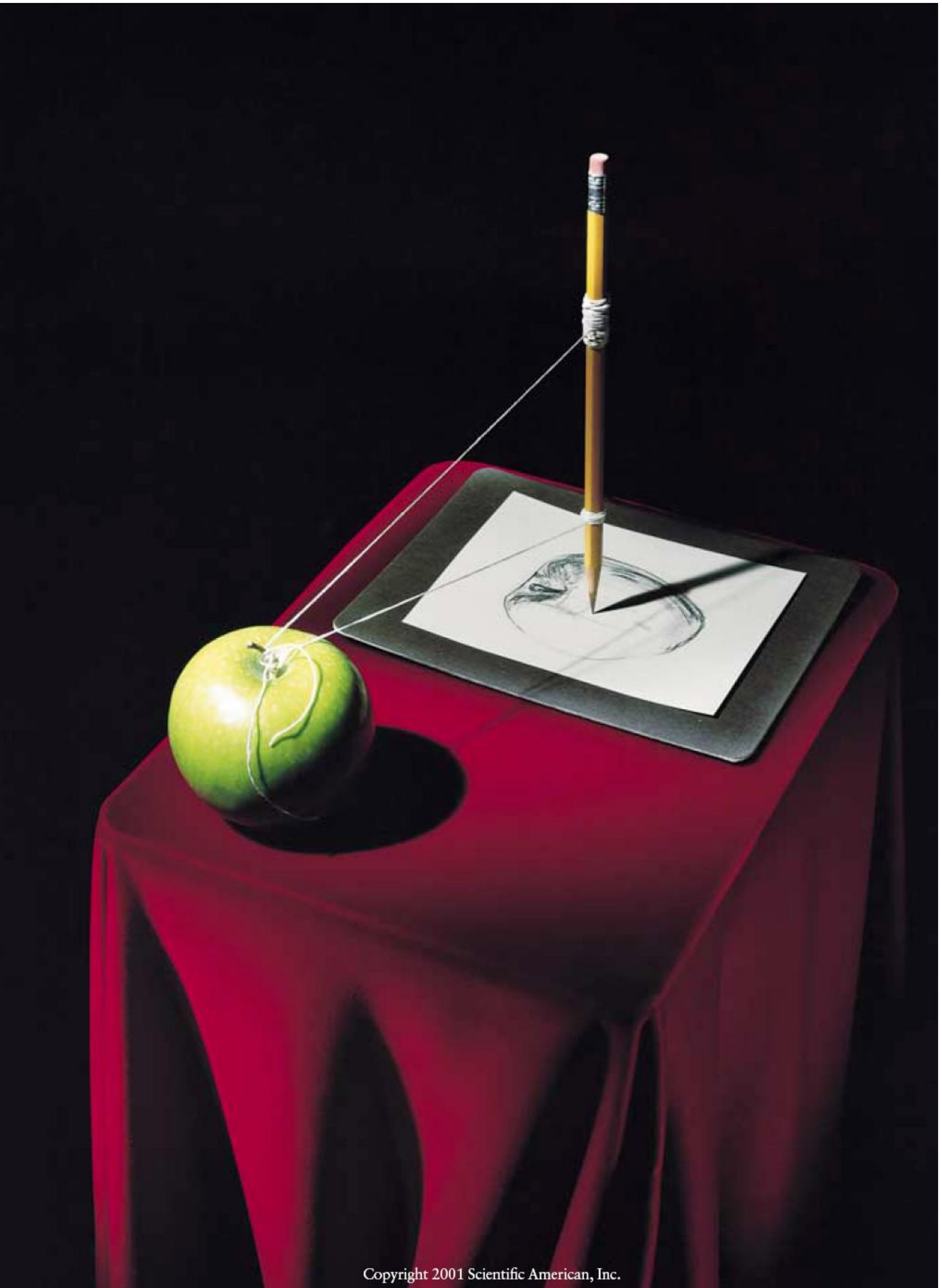






TERMINATOR
THE SARAH CONNOR CHRONICLES
FOX



Go Forth and Replicate

Birds do it, bees do it, but could machines do it? New computer simulations suggest that the answer is yes

Apples beget apples, but can machines beget machines? Today it takes an elaborate manufacturing apparatus to build even a simple machine. Could we endow an artificial device with the ability to multiply on its own? Self-replication has long been considered one of the fundamental properties separating the living from the nonliving. Historically our limited understanding of how biological reproduction works has given it an aura of mystery and made it seem unlikely that it would ever be done by a man-made object. It is reported that when René Descartes averred to Queen Christina of Sweden that animals were just another form of mechanical automata, Her Majesty pointed to a clock and said, "See to it that it produces offspring."

The problem of machine self-replication moved from philosophy into the realm of science and engineering in the late 1940s with the work of eminent mathematician and physicist John von Neumann. Some researchers have actually constructed physical replicators. Forty years ago, for example, geneticist Lionel Penrose and his son, Roger (the famous physicist), built small assemblies of plywood that exhibited a simple form of self-replication [see "Self-Reproducing Machines," by Lionel Penrose; SCIENTIFIC AMERICAN, June 1959]. But self-replication has proved to be so difficult that most researchers study it with the conceptual tool that von Neumann developed: two-dimensional cellular automata.

Implemented on a computer, cellular automata can simulate a huge variety of self-replicators in what amount to austere universes with different laws of physics from our own. Such models free researchers from having to worry about logistical issues such as energy and physical construction so that they can focus on the fundamental questions of information flow. How is a living being able to replicate unaided, whereas mechanical objects must be constructed by humans? How does replication at the level of an organism emerge from the numerous interactions in tissues, cells and molecules? How did Darwinian evolution give rise to self-replicating organisms?

The emerging answers have inspired the development of self-repairing silicon chips [*see box on page 40*] and autocatalyzing molecules [see "Synthetic Self-Replicating Molecules," by Julius Rebek, Jr.; SCIENTIFIC AMERICAN, July 1994]. And this may be just the beginning. Researchers in the field of nanotechnology have long proposed that self-replication will be crucial to manu-

By Moshe Sipper and James A. Reggia

Photoillustrations by David Emmite

Her Majesty pointed to a clock
and said, "See to it that it produces offspring."

facturing molecular-scale machines, and proponents of space exploration see a macroscopic version of the process as a way to colonize planets using in situ materials. Recent advances have given credence to these futuristic-sounding ideas. As with other scientific disciplines, including genetics, nuclear energy and chemistry, those of us who study self-replication face the twofold challenge of creating replicating machines and avoiding dystopian predictions of devices running amok. The knowledge we gain will help us separate good technologies from destructive ones.

Playing Life

SCIENCE-FICTION STORIES often depict cybernetic self-replication as a natural development of current technology, but they gloss over the profound problem it poses: how to avoid an infinite regress. A system might try to build a clone using a blueprint—that is, a self-description. Yet the self-description is part of the machine, is it not? If so, what describes the description? And what describes the description of the description? Self-replication in this case would be like asking an architect to make a perfect blueprint of his or her own studio. The blueprint would have to contain a miniature version of the blueprint, which would contain a miniature version of the blueprint and so on. Without this information, a construction crew would be unable to re-create the studio fully; there would be a blank space where the blueprint had been.

Von Neumann's great insight was an explanation of how to break out of the infinite regress. He realized that the self-description could be used in two distinct ways: first, as the instructions whose interpretation leads to the construction of an identical copy of the device; next, as data to be copied, uninterpreted, and attached to the newly created child so that it too possesses the ability to self-replicate. With this two-step process, the self-description need not contain a description of itself. In the architectural analogy, the blueprint would include a plan for building a photocopy machine. Once the new studio and the photocopier were built, the construction crew would simply run off a copy of the blueprint and put it into the new studio.

Living cells use their self-description, which biologists call the genotype, in exactly these two ways: transcription (DNA is copied mostly uninterpreted to form mRNA) and translation (mRNA is interpreted to build proteins). Von Neumann made this transcription-translation distinction several years before molecular biologists did, and his work has been crucial in understanding self-replication in nature.

To prove these ideas, von Neumann and mathematician Stanislaw M. Ulam came up with the idea of cellular automata. A cellular-automata simulation involves a chessboardlike grid of squares, or cells, each of which is either empty or occupied by one of several possible components. At discrete intervals of time, each cell looks at itself and its neighbors and decides whether to metamorphose into a different component. In making this decision, the cell follows relatively simple rules, which are the same for all cells. These rules constitute the basic physics of the cellular-automata world. All decisions and actions take place locally; cells do not know directly what is happening outside their immediate neighborhood.

The apparent simplicity of cellular automata is deceptive; it does not imply ease of design or poverty of behavior. The most famous automata, John Horton Conway's Game of Life, produces amazingly intricate patterns. Many questions about the dynamic behavior of cellular automata are formally unsolvable. To see how a pattern will unfold, you need to simulate it fully [see Mathematical Games, by Martin Gardner; SCIENTIFIC AMERICAN, October 1970 and February 1971; and "The Ultimate in Anty-Particles," by Ian Stewart, July 1994]. In its own way, a cellular-automata model can be just as complex as the real world.

Copy Machines

WITHIN CELLULAR AUTOMATA, self-replication occurs when a group of components—a "machine"—goes through a sequence of steps to construct a nearby duplicate of itself. Von Neumann's machine was based on a universal constructor, a machine that, given the appropriate instructions, could create any pattern. The constructor consisted of numerous types of components spread over tens of thousands of cells and required a book-length manuscript to be specified. It has still not been simulated in its entirety, let alone actually built, on account of its complexity. A constructor would be even more complicated in the Game of Life because the functions performed by single cells in von Neumann's model—such as transmission of signals and generation of new components—have to be performed by composite structures in Life.

Going to the other extreme, it is easy to find trivial examples of self-replication. For example, suppose a cellular automata has only one type of component, labeled +, and that each cell follows only a single rule: if exactly one of the four neighboring cells contains a +, then the cell becomes a +; otherwise it becomes vacant. With this rule, a single + grows into four more +'s, each of which grows likewise, and so forth.

THE AUTHORS

MOSHE SIPPER and *JAMES A. REGGIA* share a long-standing interest in how complex systems can self-organize. Sipper is a senior lecturer in the department of computer science at Ben-Gurion University in Israel and a visiting researcher at the Logic Systems Laboratory of the Swiss Federal Institute of Technology in Lausanne. He is interested mainly in bio-inspired computational paradigms such as evolutionary computation, self-replicating systems and cellular computing. Reggia is a professor of computer science and neurology, working in the Institute for Advanced Computer Studies at the University of Maryland. In addition to studying self-replication, he conducts research on computational models of the brain and its disorders, such as stroke.

Such weedlike proliferation does not shed much light on the principles of replication, because there is no significant machine. Of course, that invites the question of how you would tell a "significant" machine from a trivially prolific automata. No one has yet devised a satisfactory answer. What is clear, however, is that the replicating structure must in some sense be complex. For example, it must consist of multiple, diverse components whose interactions collectively bring about replication—the proverbial "whole must be greater than the sum of the parts." The existence of multiple distinct components permits a self-description to be stored within the replicating structure.

In the years since von Neumann's seminal work, many researchers have probed the domain between the complex and the trivial, developing replicators that require fewer components, less space or simpler rules. A major step forward was taken in 1984 when Christopher G. Langton, then at the University of Michigan, observed that looplike storage devices—which had formed modules of earlier self-replicating machines—could be programmed to replicate on their own. These devices typically consist of two pieces: the loop itself, which is a string of components that circulate around a rectangle, and a construction arm, which protrudes from a corner of the rectangle into the surrounding space. The circulating components constitute a recipe for the loop—for example, "go three squares ahead, then turn left." When this recipe reaches the construction arm, the automata rules make a copy of it. One copy continues around the loop; the other goes down the arm, where it is interpreted as instructions.

By giving up the requirement of universal construction, which was central to von Neumann's approach, Langton showed that a replicator could be constructed from just seven unique components occupying only 86 cells. Even smaller and simpler self-replicating loops have been devised by one of us (Reggia) and our colleagues [*see box on next page*]. Because they have multiple interacting components and include a self-description, they are not trivial. Intriguingly, asymmetry plays an unexpected role: the rules governing replication are often simpler when the components are not rotationally symmetric than when they are.

Emergent Replication

ALL THESE SELF-REPLICATING structures have been designed through ingenuity and much trial and error. This process is arduous and often frustrating; a small change to one of the rules results in an entirely different global behavior, most likely the disintegration of the structure in question. But recent work has gone beyond the direct-design approach. Instead of tailoring the rules to suit a particular type of structure, researchers have experimented with various sets of rules, filled the cellular-automata grid with a "primordial soup" of randomly selected components and checked whether self-replicators emerged spontaneously.

In 1997 Hui-Hsien Chou, now at Iowa State University, and Reggia noticed that as long as the initial density of the free-floating components was above a certain threshold, small self-replicating loops reliably appeared. Loops that collided underwent annihilation, so there was an ongoing process of death as well as birth. Over time, loops proliferated, grew in size and evolved through mutations triggered by debris from past collisions. Although the automata rules were deterministic, these mutations were effectively random, because the system was complex and the components started in random locations.

Such loops are intended as abstract machines and not as simulacra of anything biological, but it is interesting to compare them with biomolecular structures. A loop loosely resembles circular DNA in bacteria, and the construction arm acts as the enzyme that catalyzes DNA replication. More important, replicating loops illustrate how complex global behaviors can arise from simple local interactions. For example, components move around a loop even though the rules say nothing about movement; what is actually happening is that individual cells are coming alive, dying or metamorphosing in such a way that a pattern is eliminated from one position and reconstructed elsewhere—a process that we perceive as motion. In short, cellular automata act locally but appear to think globally. Much the same is true of molecular biology.

In a recent computational experiment, Jason Lohn, now at the NASA Ames Research Center, and Reggia experimented not with different structures but with different sets of rules. Starting with an arbitrary block of four components, they found they could determine a set of rules that made the block self-replicate. They discovered these rules via a genetic algorithm, an automated process that simulates Darwinian evolution.

The most challenging aspect of this work was the definition of the so-called fitness function—the criteria by which sets of rules were judged, thus separating good solutions from bad ones and driving the evolutionary process toward rule sets that facilitated replication. You cannot simply assign high fitness to those sets of rules that cause a structure to replicate, because none of the initial rule sets is likely to allow for replication. The solution was to devise a fitness function composed of a weighted sum of three measures: a growth measure (the extent to which each component type generates an increasing supply of that component), a relative position measure (the extent to which neighboring components stay together) and a replicant measure (a function of the number of actual replicators present). With the right fitness function, evolution can turn rule sets that are sterile into ones that are fecund; the process usually takes 150 or so generations.

Self-replicating structures discovered in this fashion work in a fundamentally different way than self-replicating loops do. For example, they move and deposit copies along the way—unlike replicating loops, which are essentially static. And although these newly discovered replicators consist of multiple, locally interacting components, they do not have an identifiable self-description—there is no obvious genome. The ability to replicate without a self-description may be relevant to questions about how the earliest biological

Continued on page 43

BUILD YOUR OWN REPLICATOR

SIMULATING A SMALL self-replicating loop using an ordinary chess set is a good way to get an intuitive sense of how these systems work. This particular cellular-automata model has four different types of components: pawns, knights, bishops and rooks. The machine initially comprises four pawns, a knight and a bishop. It has two parts: the loop itself, which consists of a two-by-two square, and a construction arm, which sticks out to the right.

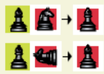
The knight and bishop represent the self-description: the knight, whose orientation is significant, determines which direction to grow, while the bishop tags along and determines how long the side of the loop should be. The pawns are fillers that define the rest of the shape of the loop, and the rook is a transient signal to guide the growth of a new construction arm.

As time progresses, the knight and bishop circulate counterclockwise around the loop. Whenever they encounter the arm, one copy goes out the arm while the original continues around the loop.

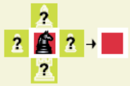
HOW TO PLAY: You will need two chessboards: one to represent the current configuration, the other to show the next configuration. For each round, look at each square of the current configuration, consult the rules and place the appropriate piece in the corresponding square on the other board. Each piece metamorphoses depending on its identity and that of the four squares immediately to the left, to the right, above and below. When you have reviewed each square and set up the next configuration, the round is over. Clear the first board and repeat. Because the rules are complicated, it takes a bit of patience at first. You can also view the simulation at lslwww.epfl.ch/chess

The direction in which a knight faces is significant. In the drawings here, we use standard chess conventions to indicate the orientation of the knight: the horse's muzzle points forward. If no rule explicitly applies, the contents of the square stay the same. Squares on the edge should be treated as if they have adjacent empty squares off the board. —*M.S. and J.A.R.*

KNIGHT



IF THERE is a bishop just behind or to the left of the knight, replace the knight with another bishop.

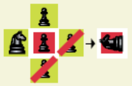


OTHERWISE, if at least one of the neighboring squares is occupied, remove the knight and leave the square empty.

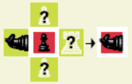
PAWN

IF THERE is a neighboring knight, replace the pawn with a knight with a certain orientation, as follows:

IF A NEIGHBORING knight is facing away from the pawn, the new knight faces the opposite way.

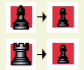


OTHERWISE, if there is exactly one neighboring pawn, the new knight faces that pawn.



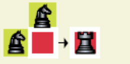
OTHERWISE the new knight faces in the same direction as the neighboring knight.

BISHOP OR ROOK



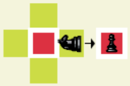
REPLACE IT with a pawn.

EMPTY SQUARE

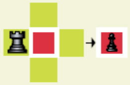


IF THERE are two neighboring knights and either faces the empty square, fill the square with a rook.

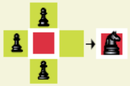
IF THERE is only one neighboring knight and it faces the square, fill the square with a knight rotated 90 degrees counterclockwise.



IF THERE is a neighboring knight and its left side faces the square, and the other neighbors are empty, fill the square with a pawn.

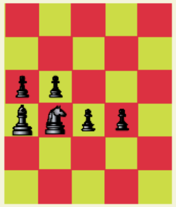


IF THERE is a neighboring rook, and the other neighbors are empty, fill the square with a pawn.

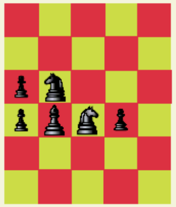


IF THERE are three neighboring pawns, fill the square with a knight facing the fourth, empty neighbor.

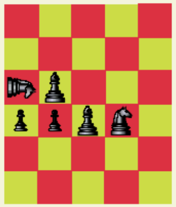
STAGES OF REPLICATION



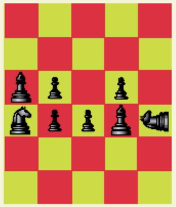
INITIALLY, the self-description, or "genome"—a knight followed by a bishop—is poised at the start of the construction arm.



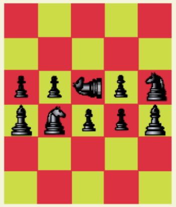
1 The knight and bishop move counterclockwise around the loop. A clone of the knight heads out the arm.



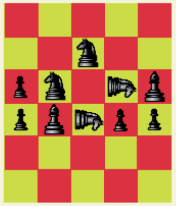
2 The original knight-bishop pair continues to circulate. The bishop is cloned and follows the new knight out the arm.



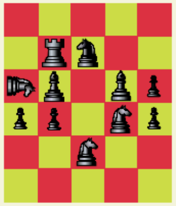
3 The knight triggers the formation of two corners of the child loop. The bishop tags along, completing the gene transfer.



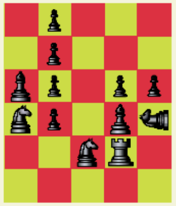
4 The knight forges the remaining corner of the child loop. The loops are connected by the construction arm and a knight-errant.



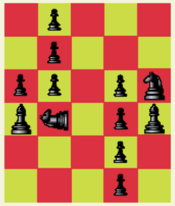
5 The knight-errant moves up to endow the parent with a new arm. A similar process, one step delayed, begins for the child loop.



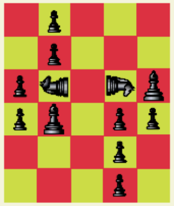
6 The knight-errant, together with the original knight-bishop pair, conjures up a rook. Meanwhile the old arm is erased.



7 The rook kills the knight and generates the new, upward arm. Another rook prepares to do the same for the child.



8 At last the two loops are separate and whole. The self-descriptions continue to circulate, but otherwise all is calm.



9 The parent prepares to give birth again. In the following step, the child too will begin to replicate.

ROBOT, HEAL THYSELF

Computers that fix themselves are the first application of artificial self-replication

LAUSANNE, SWITZERLAND—Not many researchers encourage the wanton destruction of equipment in their labs. Daniel Mange, however, likes it when visitors walk up to one of his inventions and press the button marked KILL. The lights on the panel go out; a small box full of circuitry is toast. Early in May his team unveiled its latest contraption at a science festival here—a wall-size digital clock whose components you can zap at will—and told the public: Give it your best shot. See if you can crash the system.

The goal of Mange and his team is to instill electronic circuits with the ability to take a lickin' and keep on tickin'—just like living things. Flesh-and-blood creatures might not be so good at calculating π to the millionth digit, but they can get through the day without someone pressing Ctrl-Alt-Del. Combining the precision of digital hardware with the resilience of biological wetware is a leading challenge for modern electronics.

Electronics engineers have been working on fault-tolerant circuits ever since there were electronics engineers [see "Redundancy in Computers," by William H. Pierce; SCIENTIFIC AMERICAN, February 1964]. Computer modems would still be dribbling data at 1200 baud if it weren't for error detection and correction. In many applications, simple quality-control checks, such as extra data bits, suffice. More complex systems provide entire backup computers. The space shuttle, for example, has five processors. Four of them perform the same calculations; the fifth checks whether they agree and pulls the plug on any dissenter.

The problem with these systems, though, is that they rely on centralized control. What if that control unit goes bad?

Nature has solved that problem through radical decentralization. Cells in the body are all basically identical; each takes on a specialized task, performs it autonomously and, in the event of infection or failure, commits hara-kiri so that its tasks can be taken up by new cells. These are the attributes that Mange, a professor at the Swiss Federal Institute of Technology here, and others have sought since 1993 to emulate in circuitry, as part of the "Embryonics" (embryonic electronics) project.

One of their earlier inventions, the MICTREE (microinstruction tree) artificial cell, consisted of a simple processor and four bits of data storage. The cell is contained in a plastic box roughly the size of a pack of Post-its. Electrical contacts run along the sides so that cells can be snapped together like Legos. As in cellular automata, the models used to study the theory of self-replication, the MICTREE cells are connected only to their immediate neighbors. The communication burden on each cell is thus independent of the total number of cells. The system, in other words, is easily scalable—unlike many parallel-computing architectures.

Cells follow the instructions in their "genome," a program written in a subset of the Pascal computer language. Like their biological antecedents, the cells all contain the exact same genome and execute part of it based on their position within the array, which each cell calculates relative to its neighbors. Wasteful though it may seem, this redundancy allows the array to withstand the loss of any cell. Whenever someone presses the KILL button on a cell, that cell shuts down, and its left and right neighbors become directly connected. The right neighbor recalculates its position and starts executing the deceased's program. Its tasks, in turn, are taken up by the next cell to the right, and so on, until a cell designated as a spare is pressed into service.

Writing programs for any parallel processor is tricky, but the MICTREE array requires an especially unconventional approach. Instead of giving explicit instructions, the programmer must devise simple rules out of which the desired function will emerge. Being Swiss, Mange demonstrates by building a superreliable stopwatch. Displaying minutes and seconds requires four cells in a row, one for each digit. The genome allows for two cell types: a counter from zero to nine and a counter from zero to five. An oscillator feeds one pulse per second into the rightmost cell. After 10 pulses, this cell cycles back to zero and sends a pulse to the cell on its left, and so on down the line. The watch takes up part of an array of 12 cells; when you kill one, the clock transplants itself one cell over and carries on. Obviously, though, there is a limit to its resilience: the whole thing will fail after, at most, eight kills.

The prototype MICTREE cells are hardwired, so their processing power cannot be tailored to a specific application. In a finished product, cells would instead be implemented on a field-programmable gate array, a grid of electronic components that can be reconfigured on the fly [see "Configurable Computing," by John Villasenor and William H. Mangione-Smith; SCIENTIFIC AMERICAN, June 1997]. Mange's team is now custom-designing a gate array, known as MUXTREE (multiplexer tree), that is optimized for artificial cells. In the biological metaphor, the components of this array are the "molecules" that constitute a cell. Each consists of a logic gate, a data bit and a string of configuration bits that determines the function of this gate.

Building a cell out of such molecules offers not only flexibility but also extra endurance. Each molecule contains two copies of the gate and three of the storage bit. If the two gates ever give different results, the molecule kills itself for the greater good of the cell. As a last gasp, the molecule sends its data bit (preserved by the triplicate storage) and configuration to its right neighbor, which does the same, and the process continues until the rightmost molecule transfers its data to a spare. This second level of fault tolerance prevents a single error from wiping out an entire cell.

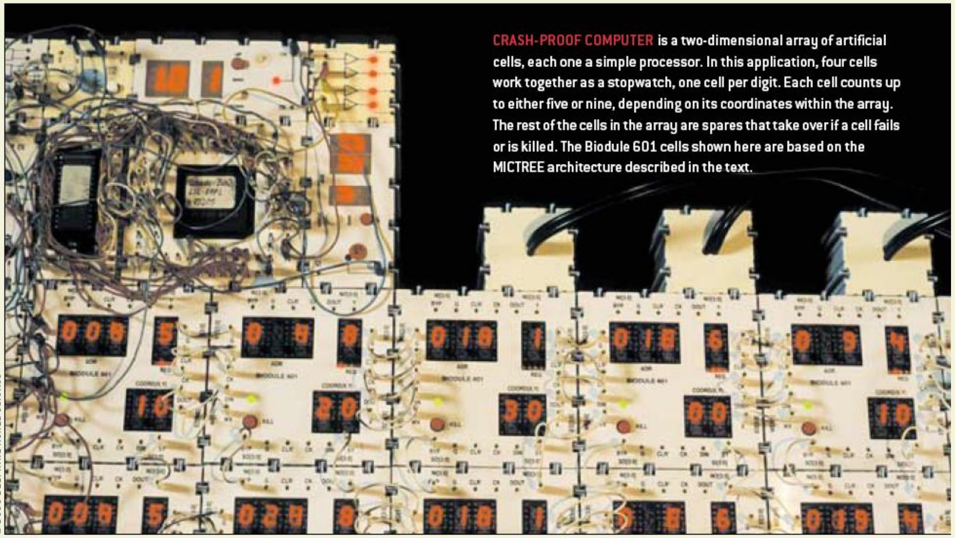
A total of 2,000 molecules, divided into four 20-by-25 cells, make up the BioWall—the giant digital clock that Mange's team has just put on display. Each molecule is enclosed in a small box and includes a KILL button and an LED display. Some molecules are configured to perform computations; others serve as pixels in the clock display. Making liberal use of the KILL buttons, I did my utmost to crash the system, something I'm usually quite good at. But the plucky clock just wouldn't submit. The clock display did start to look funny—numerals bent over as their pixels shifted to the right—but at least it was still legible, unlike most faulty electronic signs.

That said, the system did suffer from display glitches, which Mange attributed mainly to timing problems. Although the processing power is decentralized, the cells still rely on a central oscillator to coordinate their communications; sometimes they fall out of sync. Another Embryonics team, led by Andy Tyrrell of the University of York in England, has been studying making the cells asynchronous, like their biological counterparts. Cells would generate handshaking signals to orchestrate data transfers. The present system is also unable to catch certain types of error, including damaged configuration strings. Tyrrell's team has proposed adding watchdog molecules—an immune system—that would monitor the configurations (and one another) for defects.

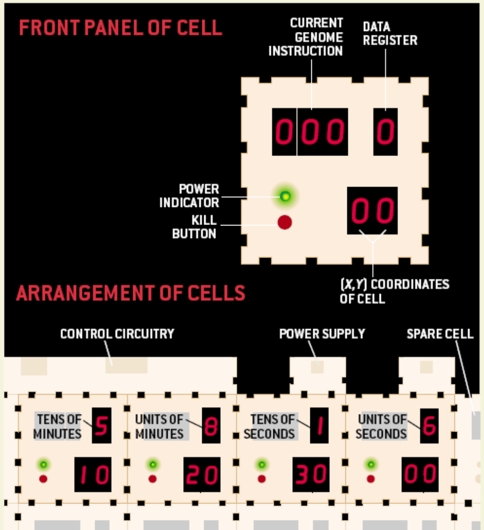
Although these systems demand an awful lot of overhead, so do other fault-tolerance technologies. "While Embryonics appears to be heavy on redundancy, it actually is not that bad when compared to other systems," Tyrrell argues. Moreover, MUXTREE should be easier to scale down to the nano level; the "molecules" are simple enough to really be molecules. Says Mange, "We are preparing for the situation where electronics will be at the same scale as biology."

On a philosophical level, Embryonics comes very close to the dream of building a self-replicating machine. It may not be quite as dramatic as a robot that can go down to Radio Shack, pull parts off the racks, and take them home to resolder a connection or build a loving mate. But the effect is much the same. Letting machines determine their own destiny—whether reconfiguring themselves on a silicon chip or reprogramming themselves using a neural network or genetic algorithm—sounds scary, but perhaps we should be gratified that machines are becoming more like us: imperfect, fallible but stubbornly resourceful.

—George Musser, imperfect but resourceful staff editor and writer



CRASH-PROOF COMPUTER is a two-dimensional array of artificial cells, each one a simple processor. In this application, four cells work together as a stopwatch, one cell per digit. Each cell counts up to either five or nine, depending on its coordinates within the array. The rest of the cells in the array are spares that take over if a cell fails or is killed. The Biodule 601 cells shown here are based on the MICTREE architecture described in the text.



Continued from page 39

replicators originated. In a sense, researchers are seeing a continuum between nonliving and living structures.

Many researchers have tried other computational models besides the traditional cellular automata. In asynchronous cellular automata, cells are not updated in concert; in nonuniform cellular automata, the rules can vary from cell to cell. Another approach altogether is Core War [see Computer Recreations, by A. K. Dewdney; SCIENTIFIC AMERICAN, May 1984] and its successors, such as ecologist Thomas S. Ray's Tierra system. In these simulations the "organisms" are computer programs that vie for processor time and memory. Ray has observed the emergence of "parasites" that co-opt the self-replication code of other organisms.

Getting Real

SO WHAT GOOD are these machines? Von Neumann's universal constructor can compute in addition to replicating, but it is an impractical beast. A major advance has been the development of simple yet useful replicators. In 1995 Gianluca Tempesti of the Swiss Federal Institute of Technology in Lausanne simplified the loop self-description so it could be interlaced with a small program—in this case, one that would spell the acronym of his lab, "LSL." His insight was to create automata rules that allow loops to replicate in two stages. First the loop, like Langton's loop, makes a copy of itself. Once finished, the daughter loop sends a signal back to its parent, at which point the parent sends the instructions for writing out the letters.

Drawing letters was just a demonstration. The following year Jean-Yves Perrier, Jacques Zahnd and one of us (Sipper) designed a self-replicating loop with universal computational capabilities—that is, with the computational power of a universal Turing machine, a highly simplified but fully capable computer. This loop has two "tapes," or long strings of components, one for the program and the other for data. The loops can execute an arbitrary program in addition to self-replicating. In a sense, they are as complex as the computer that simulates them. Their main limitation is that the program is copied unchanged from parent to child, so that all loops carry out the same set of instructions.

In 1998 Chou and Reggia swept away this limitation. They showed how self-replicating loops carrying distinct information, rather than a cloned program, can be used to solve a problem known as satisfiability. The loops can be used to determine whether the variables in a logical expression can be assigned values such that the entire expression evaluates to "true." This problem is NP-complete—in other words, it belongs to the family of nasty puzzles, including the famous traveling-salesman problem, for which there is no known efficient solution. In Chou and Reggia's cellular-automata universe, each replicator received a different partial solution. During replication, the solutions mutated, and replicators with promising solutions were allowed to proliferate while those with failed solutions died out.

Although various teams have created cellular automata in electronic hardware, such systems are probably too wasteful for practical applications; automata were never really intended to be implemented directly. Their purpose is to illuminate the underlying principles of replication and, by doing so, inspire more concrete efforts. The loops provide a new paradigm for designing a parallel computer from either transistors or chemicals [see "Computing with DNA," by Leonard M. Adleman; SCIENTIFIC AMERICAN, August 1998].

In 1980 a NASA team led by Robert Freitas, Jr., proposed planting a factory on the moon that would replicate itself, using local lunar materials, to populate a large area exponentially. Indeed, a similar probe could colonize the entire galaxy, as physicist Frank J. Tipler of Tulane University has argued. In the nearer term, computer scientists and engineers have experimented with the automated design of robots [see "Dawn of a New Species?" by George Musser; SCIENTIFIC AMERICAN, November 2000]. Although these systems are not truly self-replicating—the offspring are much simpler than the parent—they are a first step toward fulfilling the queen of Sweden's request.

In a sense, researchers are seeing a continuum between nonliving and living structures.

Should physical self-replicating machines become practical, they and related technologies will raise difficult issues, including the *Terminator* film scenario in which artificial creatures outcompete natural ones. We prefer the more optimistic, and more probable, scenario that replicators will be harnessed to the benefit of humanity [see "Will Robots Inherit the Earth?" by Marvin Minsky; SCIENTIFIC AMERICAN, October 1994]. The key will be taking the advice of 14th-century English philosopher William of Ockham: *entia non sunt multiplicanda praeter necessitatem*—entities are not to be multiplied beyond necessity. SA

MORE TO EXPLORE

Simple Systems That Exhibit Self-Directed Replication. J. Reggia, S. Armentrout, H. Chou and Y. Peng in *Science*, Vol. 259, No. 5099, pages 1282–1287; February 26, 1993.

Emergence of Self-Replicating Structures in a Cellular Automata Space. H. Chou and J. Reggia in *Physica D*, Vol. 110, Nos. 3–4, pages 252–272; December 15, 1997.

Special Issue: Von Neumann's Legacy: On Self-Replication. Edited by M. Sipper, G. Tempesti, D. Mange and E. Sanchez in *Artificial Life*, Vol. 4, No. 3; Summer 1998.

Towards Robust Integrated Circuits: The Embryonics Approach. D. Mange, M. Sipper, A. Stauffer and G. Tempesti in *Proceedings of the IEEE*, Vol. 88, No. 4, pages 516–541; April 2000.

Moshe Sipper's Web page on artificial self-replication is at **lslwww.epfl.ch/~moshes/selfrep/**

Animations of self-replicating loops can be found at **necsi.org/postdocs/sayama/sdsr/java/**

For John von Neumann's universal constructor, see **alife.santafe.edu/alife/topics/jvn/jvn.html**

John von Neumann Institut für Computing

The NIC
Structure
Addresses
Sites
John von Neumann
NIC Brochure
Supercomputers
Support
Documentation
Computing Time
Research Groups
Publications
Projects
Internals
News & Events
Contact
Imprint
Search

John von Neumann Institute for Computing (NIC)

The John von Neumann Institute for Computing (NIC) is a joint foundation of Forschungszentrum Jülich and Deutsches Elektronen-Synchrotron DESY to support supercomputer-aided scientific research and development. Since April 2006, the GSI Helmholtzzentrum für Schwerionenforschung joined NIC as a contract partner. NIC takes over the functions and tasks of the High Performance Computer Centre (HLRZ) established in 1987 and continues this centre's successful work in the field of supercomputing and its applications.

- **Provision of supercomputer capacity** for projects in science, research and industry in the fields of modelling and computer simulation including their methods.
  Research proposals can be submitted by German scientists and by partners in the EU projects DEISA and I3HP.
  There is also an Offer to the New Member States and candidate countries of the European Union.

  The supercomputers with the required information technology infrastructure (software, data storage, networks) are operated by the Jülich Supercomputing Centre (JSC) in Jülich and by the Centre for Parallel Computing at DESY in Zeuthen.
- **Supercomputer-oriented research and development** in selected fields of physics and other sciences, especially in elementary-particle physics, by research groups for supercomputing applications.
- **Education and training in the fields of scientific computing** by symposia, workshops, summer schools, seminars, courses, and guest programs for scientists and students.

S.Hoefler-Thierfeldt@fz-juelich.de, 01-Jul-2008
URL: <http://www.fz-juelich.de/nic/Allgemeines/Allgemeines-e.html>

HOME DEUTSCH

YOU ARE HERE: AWARDS > - INFORMS Prizes and Awards > John von Neumann Theory Prize

John von Neumann Theory Prize

A prize is awarded annually to a scholar (or scholars in the case of joint work) who has made fundamental, sustained contributions to theory in operations research and the management sciences. The award is given each year at the National Meeting if there is a suitable recipient. Although the prize is normally given to a single individual, in the case of accumulated joint work, the recipients can be multiple individuals. The award is $5,000, a medallion and a citation.

- **Who Was John von Neumann?**
  Read about the life and legacy of John von Neumann
- **Application Process**
  View information about eligibility, procedures and deadlines
- **Past Winners**
  View information about all past winners of this prize

Most Recent Winner

2008: **Frank P. Kelly**

Return to the INFORMS Awards Page

ANNOUNCEMENTS

INFORMS Prizes and Awards: 2008 Winners

Subdivision Prizes and Awards: 2008 Winners

INFORMS Fellows: Class of 2008

Recognition of INFORMS members: View the news

What's New: View new information about prizes and awards

E-mail:
titkarsag@njszt.hu

Neumann János Számítógép-tudományi Társaság

H-1054 Budapest, Báthori u. 16. Tel.: (+36-1) 472-2730
ECDL Iroda: 472-2710 F.: 472-2728

NJSZT in English -> Introduction

Általános információk
Szakmai közösségek
Területi szervezetek
Szakkönyvtár
Informatikai írástudás: ECDL
Versenyek
Kapcsolataink
Rendezvények
Szakértői tevékenység
Archívum
NJSZT in English
Introduction
RCEAS 2007-2008
ETAPS 2008
eVITA 2008
John von Neumann
Mi Újság
Faliújság
Fórum
KERESÉS
OK
Google
Google Search
Üdvözlőlap

Introducing the John von Neumann Computer Society

Ability, pride and creativity of our compatriots are for Hungary the fundament of progress and the only spring-board into the future.
(Count István Széchenyi 1842)

Fields of activity:

As a significant professional body and learned society in the Hungarian IT community, the John von Neumann Computer Society (NJSZT) is dedicated to preserving values that can be included in today's knowledge-based society as well as to setting new directions that meet the requirements of the age and to actively forming the IS world of the future. The primal activities of our Society are IT support, ECDL (European Computer Driving Licence) Hungary, Hungarian Smart Card Forum, Organization of International and National Conferences.

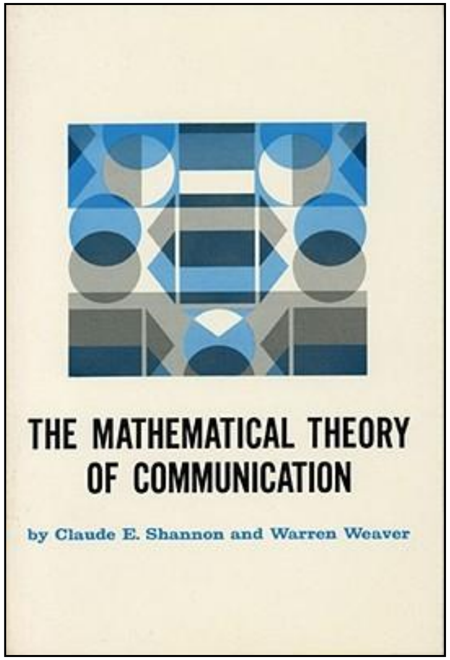
John von Neumann Computer Society
Neumann János Számítógép-tudományi Társaság

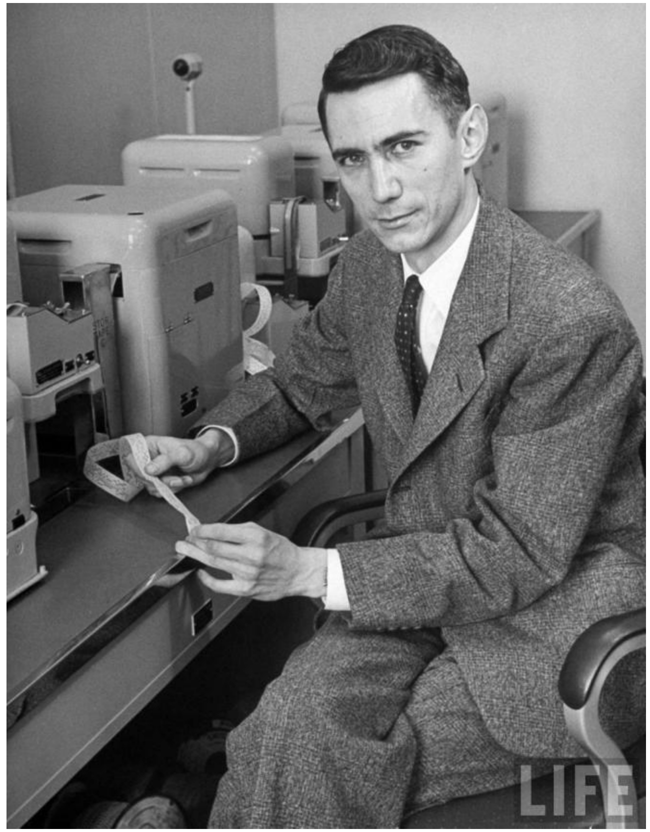
Address :	**Báthori u. 16. 1054 Budapest Hungary**
Mailing address :	**P.O.B. 201 1364 Budapest Hungary**
Telephone :	**(36-1) 472-2730**
Fax. :	**(36-1) 472-2739**
E-mail :	secretariat@njszt.hu
WWW :	www.njszt.hu
President :	**Prof. Gábor PÉCELI**
Telephone :	**(36-1) 463-2057**
Fax.:	**(36-1) 463-3580**
E-mail:	peceli@mit.bme.hu

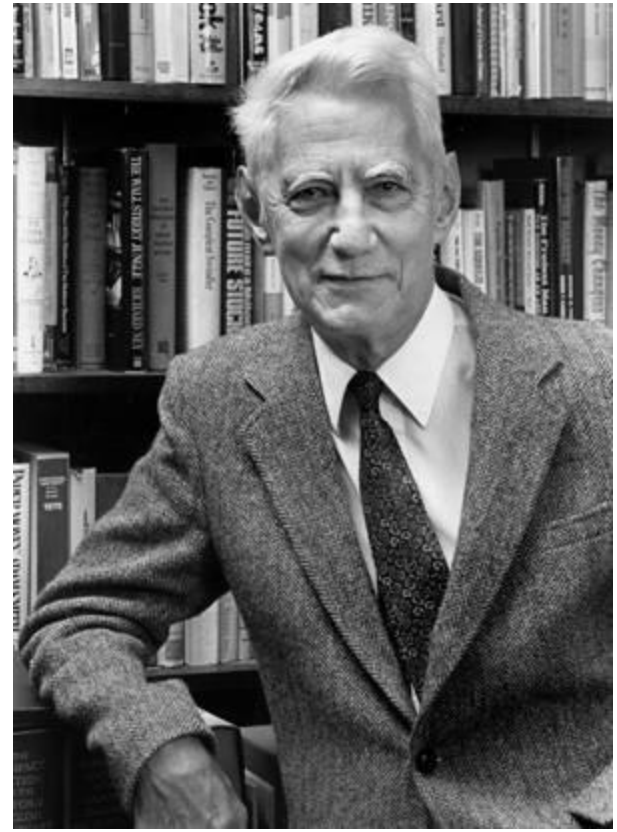
Historical Perspectives

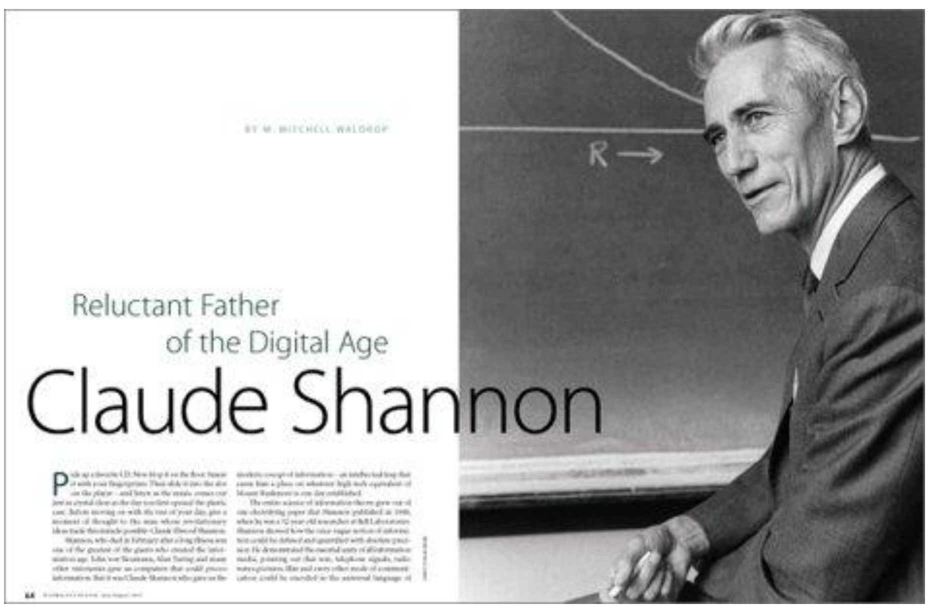
Claude Shannon (1916-2001)

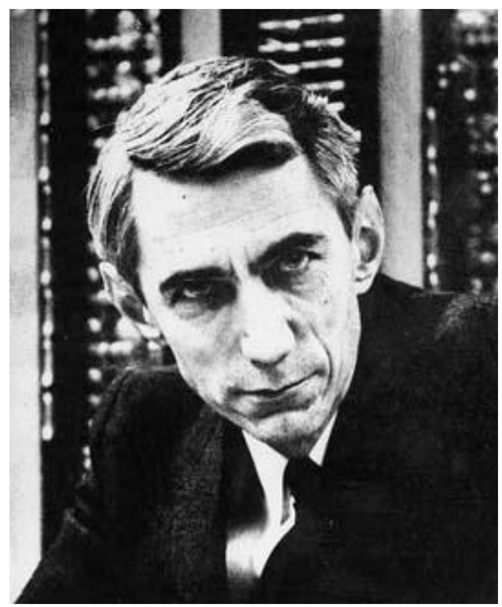
- Invented electrical digital circuits (1937)
- Founded information theory (1948)
- Introduced sampling theory, coined term “bit”
- Contributed to genetics, cryptography
- Joined Institute for Advanced Study (1940)
  Influenced by Turing, von Neumann, Einstein
- Originated information entropy, Nyquist–Shannon, sampling theorem, Shannon-Hartley theorem, Shannon switching game, Shannon-Fano coding, Shannon’s source coding theorem, Shannon limit, Shannon decomposition / expansion, Shannon #
- Other hobbies & inventions: juggling, unicycling, computer chess, rockets, motorized pogo stick, flame-throwers, Rubik's cube solver, wearable computer, mathematical gambling, stock markets
- “AT&T Shannon Labs” named after him

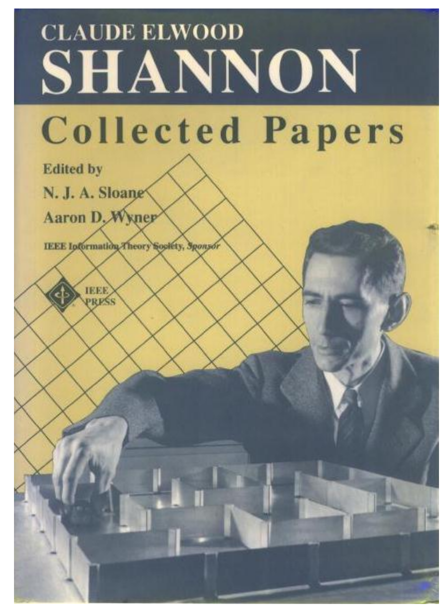



LIFE




BY M. MITCHELL WALDROP
Reluctant Father
of the Digital Age
Claude Shannon




CLAUDE ELWOOD
SHANNON
Collected Papers
Edited by
N. J. A. Sloane
Aaron D. Wyner
IEEE Information Theory Society, Sponsor
IEEE PRESS

MASS. INST. TECH.
DEC 20 1940
LIBRARY

A SYMBOLIC ANALYSIS

OF

RELAY AND SWITCHING CIRCUITS

by

Claude Elwood Shannon

B.S., University of Michigan

1936

Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

from the

Massachusetts Institute of Technology

1940

Signature of Author______________________________

Department of Electrical Engineering, August 10, 1937

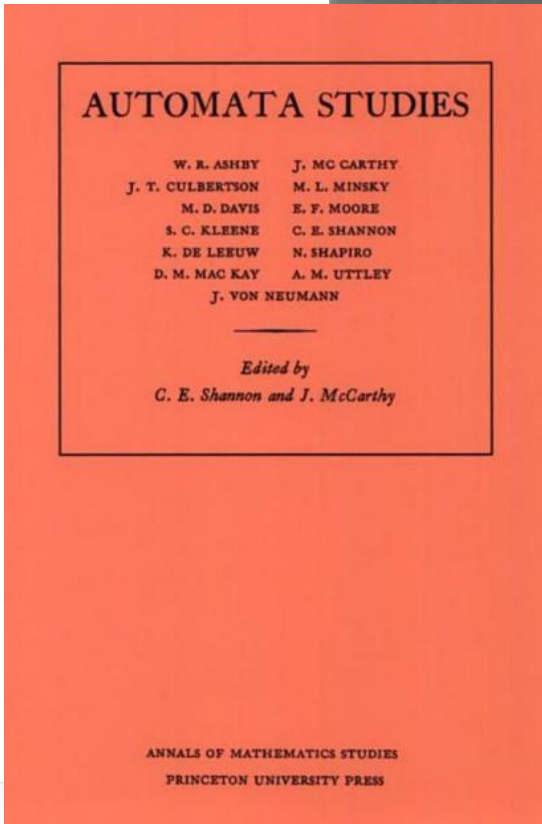
Signature of Professor
in Charge of Research______________________________

Signature of Chairman of Department
Committee on Graduate Students_______________________

38

TABLE OF CONTENTS

Theseus: Shannon’s electro-mechanical mouse (1950): first “learning machine” and AI experiment

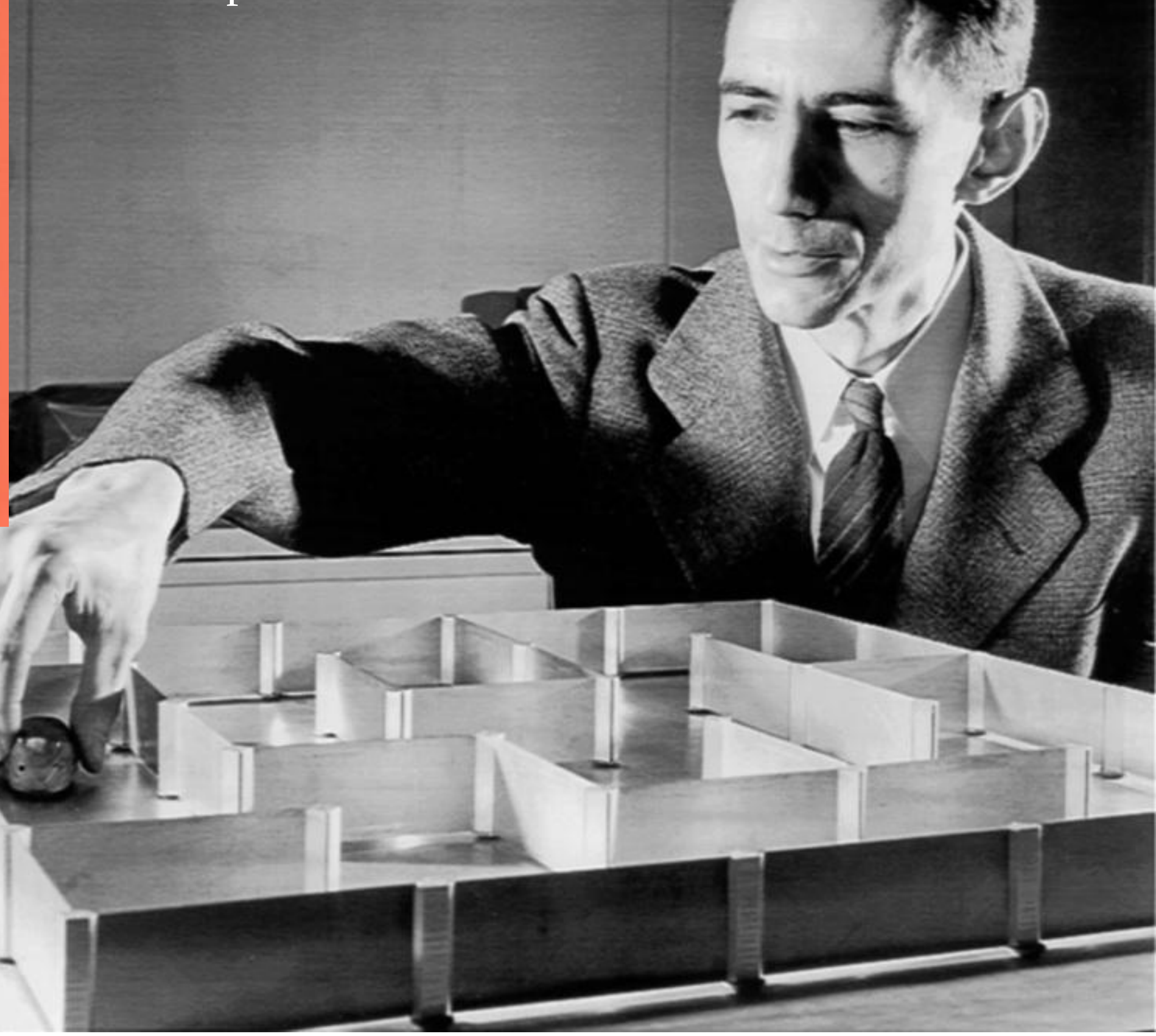


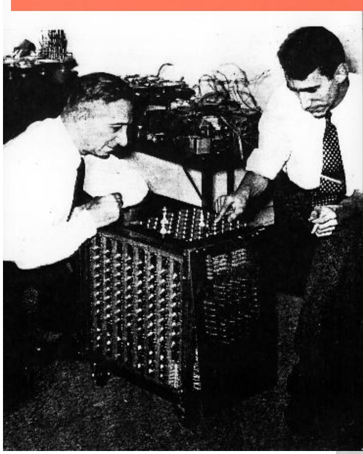
AUTOMATA STUDIES

W. R. ASHBY
J. T. CULBERTSON
M. D. DAVIS
S. C. KLEENE
K. DE LEEUW
D. M. MAC KAY
J. MC CARTHY
M. L. MINSKY
E. F. MOORE
C. E. SHANNON
N. SHAPIRO
A. M. UTTLEY
J. VON NEUMANN

Edited by
C. E. Shannon and J. McCarthy

ANNALS OF MATHEMATICS STUDIES
PRINCETON UNIVERSITY PRESS

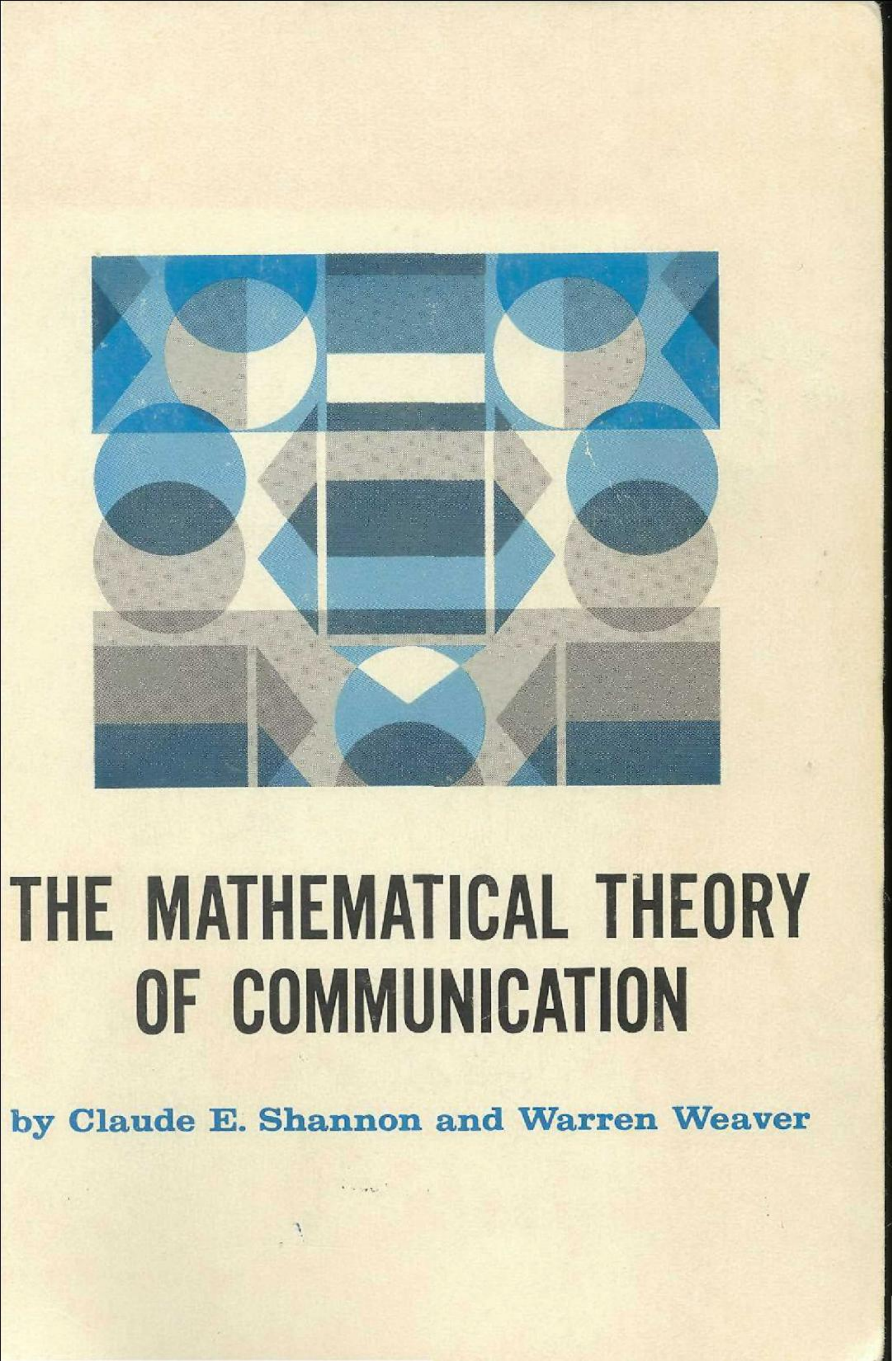


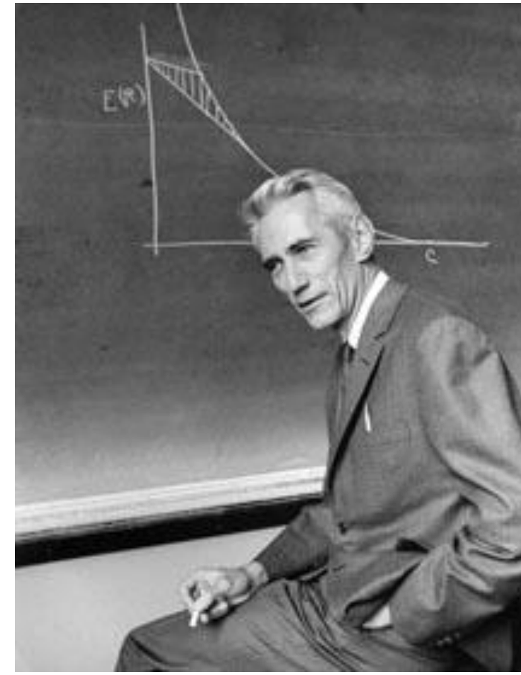


Chess champion Ed Lasker looking at Shannon’s chess-playing machine

Shannon’s home study room

Shannon’s On/Off machine





Eighth paperback printing, 1980

Originally published in a clothbound edition, 1949.

Manufactured in the United States of America.
Library of Congress Catalog Card No. 49-11922.

ISBN 0-252-72548-4

Introduction

The recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist[1] and Hartley[2] on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the statistical structure of the original message and due to the nature of the final destination of the information.

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning;* that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one *selected from a set* of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

[1] Nyquist, H., "Certain Factors Affecting Telegraph Speed," *Bell System Technical Journal,* April 1924, p. 324; "Certain Topics in Telegraph Transmission Theory," *A.I.E.E. Trans.,* v. 47, April 1928, p. 617.
[2] Hartley, R. V. L., "Transmission of Information," *Bell System Technical Journal,* July 1928, p. 535.

If the number of messages in the set is finite then this number or any monotonic function of this number can be regarded as a measure of the information produced when one message is chosen from the set, all choices being equally likely. As was pointed out by Hartley the most natural choice is the logarithmic function. Although this definition must be generalized considerably when we consider the influence of the statistics of the message and when we have a continuous range of messages, we will in all cases use an essentially logarithmic measure.

The logarithmic measure is more convenient for various reasons:

1. It is practically more useful. Parameters of engineering importance such as time, bandwidth, number of relays, etc., tend to vary linearly with the logarithm of the number of possibilities. For example, adding one relay to a group doubles the number of possible states of the relays. It adds 1 to the base 2 logarithm of this number. Doubling the time roughly squares the number of possible messages, or doubles the logarithm, etc.

2. It is nearer to our intuitive feeling as to the proper measure. This is closely related to (1) since we intuitively measure entities by linear comparison with common standards. One feels, for example, that two punched cards should have twice the capacity of one for information storage, and two identical channels twice the capacity of one for transmitting information.

3. It is mathematically more suitable. Many of the limiting operations are simple in terms of the logarithm but would require clumsy restatement in terms of the number of possibilities.

The choice of a logarithmic base corresponds to the choice of a unit for measuring information. If the base 2 is used the resulting units may be called binary digits, or more briefly *bits,* a word suggested by J. W. Tukey. A device with two stable positions, such as a relay or a flip-flop circuit, can store one bit of information. N such devices can store N bits, since the total number of possible states is 2^N and $\log_2 2^N = N$. If the base 10 is used the units may be called decimal digits. Since

$$\begin{aligned}\log_2 M &= \log_{10} M/\log_{10} 2\\ &= 3.32 \log_{10} M,\end{aligned}$$

I

Discrete Noiseless Systems

1. The Discrete Noiseless Channel

Teletype and telegraphy are two simple examples of a discrete channel for transmitting information. Generally, a discrete channel will mean a system whereby a sequence of choices from a finite set of elementary symbols $S_1 \cdots S_n$ can be transmitted from one point to another. Each of the symbols S_i is assumed to have a certain duration in time t_i seconds (not necessarily the same for different S_i, for example the dots and dashes in telegraphy). It is not required that all possible sequences of the S_i be capable of transmission on the system; certain sequences only may be allowed. These will be possible signals for the channel. Thus in telegraphy suppose the symbols are: (1) A dot, consisting of line closure for a unit of time and then line open for a unit of time; (2) A dash, consisting of three time units of closure and one unit open; (3) A letter space consisting of, say, three units of line open; (4) A word space of six units of line open. We might place the restriction on allowable sequences that no spaces follow each other (for if two letter spaces are adjacent, they are identical with a word space). The question we now consider is how one can measure the capacity of such a channel to transmit information.

In the teletype case where all symbols are of the same duration, and any sequence of the 32 symbols is allowed, the answer is easy. Each symbol represents five bits of information. If the system

transmits n symbols per second it is natural to say that the channel has a capacity of $5n$ bits per second. This does not mean that the teletype channel will always be transmitting information at this rate — this is the maximum possible rate and whether or not the actual rate reaches this maximum depends on the source of information which feeds the channel, as will appear later.

In the more general case with different lengths of symbols and constraints on the allowed sequences, we make the following definition: The capacity C of a discrete channel is given by

$$C = \lim_{T\to\infty} \frac{\log N(T)}{T}$$

where $N(T)$ is the number of allowed signals of duration T.

It is easily seen that in the teletype case this reduces to the previous result. It can be shown that the limit in question will exist as a finite number in most cases of interest. Suppose all sequences of the symbols $S_1, \cdots, S_n$ are allowed and these symbols have durations $t_1, \cdots, t_n$. What is the channel capacity? If $N(t)$ represents the number of sequences of duration t we have

$$N(t) = N(t-t_1) + N(t-t_2) + \cdots + N(t-t_n).$$

The total number is equal to the sum of the numbers of sequences ending in $S_1, S_2, \cdots, S_n$ and these are $N(t-t_1), N(t-t_2), \cdots, N(t-t_n)$, respectively. According to a well-known result in finite differences, $N(t)$ is the asymptotic for large t to AX_0^t where A is constant and X_0 is the largest real solution of the characteristic equation:

$$X^{-t_1} + X^{-t_2} + \cdots + X^{-t_n} = 1$$

and therefore

$$C = \lim_{T\to\infty} \frac{\log AX_0^T}{T} = \log X_0.$$

In case there are restrictions on allowed sequences we may still often obtain a difference equation of this type and find C from the characteristic equation. In the telegraphy case mentioned above

$$N(t) = N(t-2) + N(t-4) + N(t-5) + N(t-7) + N(t-8) + N(t-10)$$

a decimal digit is about $3\frac{1}{3}$ bits. A digit wheel on a desk computing machine has ten stable positions and therefore has a storage capacity of one decimal digit. In analytical work where integration and differentiation are involved the base e is sometimes useful. The resulting units of information will be called natural units. Change from the base a to base b merely requires multiplication by $\log_b a$.

By a communication system we will mean a system of the type indicated schematically in Fig. 1. It consists of essentially five parts:

1. An *information source* which produces a message or sequence of messages to be communicated to the receiving terminal. The message may be of various types: (a) A sequence of letters as in a telegraph or teletype system; (b) A single function of time $f(t)$ as in radio or telephony; (c) A function of time and other variables as in black and white television — here the message may be thought of as a function $f(x, y, t)$ of two space coordinates and time, the light intensity at point (x, y) and time t on a pickup tube plate; (d) Two or more functions of time, say $f(t)$, $g(t)$, $h(t)$ — this is the case in "three-dimensional" sound transmission or if the system is intended to service several individual channels in multiplex; (e) Several functions of several variables — in color television the message consists of three functions $f(x, y, t)$, $g(x, y, t)$, $h(x, y, t)$ defined in a three-dimensional continuum — we may also think of these three functions as components of a vector field defined in the region — similarly, several black and white television sources would produce "messages" consisting of a number of functions of three variables; (f) Various combinations also occur, for example in television with an associated audio channel.

2. A *transmitter* which operates on the message in some way to produce a signal suitable for transmission over the channel. In telephony this operation consists merely of changing sound pressure into a proportional electrical current. In telegraphy we have an encoding operation which produces a sequence of dots, dashes and spaces on the channel corresponding to the message. In a multiplex PCM system the different speech functions must be sampled, compressed, quantized and encoded, and finally interleaved properly to construct the signal. Vocoder systems, television and frequency modulation are other examples of complex operations applied to the message to obtain the signal.

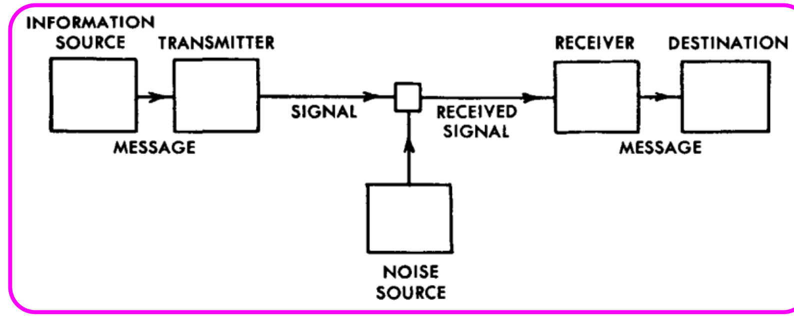


Fig. 1. — Schematic diagram of a general communication system.

3. The *channel* is merely the medium used to transmit the signal from transmitter to receiver. It may be a pair of wires, a coaxial cable, a band of radio frequencies, a beam of light, etc. During transmission, or at one of the terminals, the signal may be perturbed by noise. This is indicated schematically in Fig. 1 by the noise source acting on the transmitted signal to produce the received signal.

4. The *receiver* ordinarily performs the inverse operation of that done by the transmitter, reconstructing the message from the signal.

5. The *destination* is the person (or thing) for whom the message is intended.

We wish to consider certain general problems involving communication systems. To do this it is first necessary to represent the various elements involved as mathematical entities, suitably idealized from their physical counterparts. We may roughly classify communication systems into three main categories: discrete, continuous and mixed. By a discrete system we will mean one in which both the message and the signal are a sequence of discrete symbols. A typical case is telegraphy where the message is a sequence of letters and the signal a sequence of dots, dashes and spaces. A continuous system is one in which the

Suppose we have a set of possible events whose probabilities of occurrence are $p_1, p_2, \cdots, p_n$. These probabilities are known but that is all we know concerning which event will occur. Can we find a measure of how much "choice" is involved in the selection of the event or of how uncertain we are of the outcome?

If there is such a measure, say $H(p_1, p_2, \cdots, p_n)$, it is reasonable to require of it the following properties:

1. H should be continuous in the p_i.
2. If all the p_i are equal, $p_i = \frac{1}{n}$, then H should be a monotonic increasing function of n. With equally likely events there is more choice, or uncertainty, when there are more possible events.
3. If a choice be broken down into two successive choices, the original H should be the weighted sum of the individual values of H. The meaning of this is illustrated in Fig. 6. At the left we have three possibilities $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{3}$, $p_3 = \frac{1}{6}$. On the right we first choose between two possibilities each with probability $\frac{1}{2}$, and if the second occurs make another choice with probabilities $\frac{2}{3}$, $\frac{1}{3}$. The final results have the same probabilities as before. We require, in this special case, that

$$H(\tfrac{1}{2}, \tfrac{1}{3}, \tfrac{1}{6}) = H(\tfrac{1}{2}, \tfrac{1}{2}) + \tfrac{1}{2} H(\tfrac{2}{3}, \tfrac{1}{3}).$$

The coefficient $\frac{1}{2}$ is the weighting factor introduced because this second choice only occurs half the time.

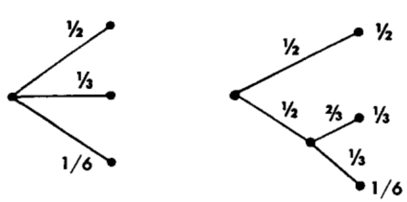


Fig. 6. — Decomposition of a choice from three possibilities.

In Appendix 2, the following result is established:

Theorem 2: The only H satisfying the three above assumptions is of the form:

$$H = -K \sum_{i=1}^{n} p_i \log p_i$$

where K is a positive constant.

This theorem, and the assumptions required for its proof, are in no way necessary for the present theory. It is given chiefly to lend a certain plausibility to some of our later definitions. The real justification of these definitions, however, will reside in their implications.

Quantities of the form $H = -\Sigma\, p_i \log p_i$ (the constant K merely amounts to a choice of a unit of measure) play a central role in information theory as measures of information, choice and uncertainty. The form of H will be recognized as that of entropy

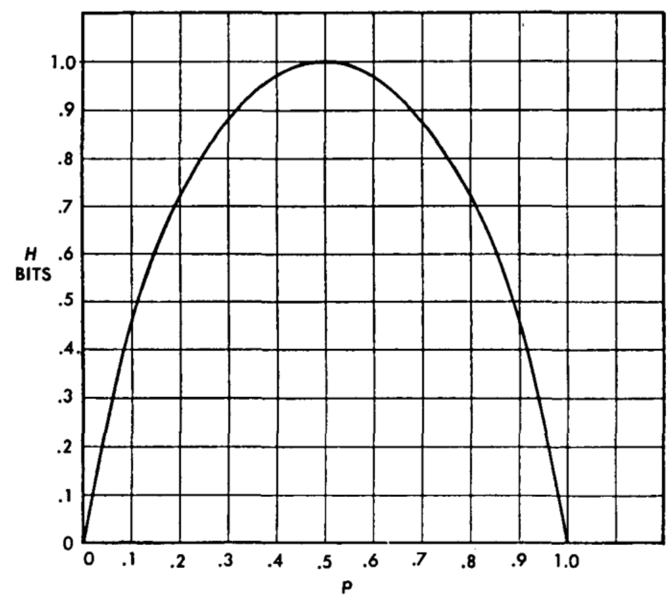


Fig. 7. — Entropy in the case of two possibilities with probabilities p and $(1-p)$.

as defined in certain formulations of statistical mechanics[8] where p_i is the probability of a system being in cell i of its phase space.

[8] See, for example, R. C. Tolman, *Principles of Statistical Mechanics*, Oxford, Clarendon, 1938.

quence of symbols x_i; and let β be the state of the transducer, which produces, in its output, blocks of symbols y_j. The combined system can be represented by the "product state space" of pairs (α, β). Two points in the space (α_1, β_1) and (α_2, β_2), are connected by a line if α_1 can produce an x which changes β_1 to β_2, and this line is given the probability of that x in this case. The line is labeled with the block of y_1 symbols produced by the transducer. The entropy of the output can be calculated as the weighted sum over the states. If we sum first on β each resulting term is less than or equal to the corresponding term for α, hence the entropy is not increased. If the transducer is non-singular let its output be connected to the inverse transducer. If H_1', H_2' and H_3' are the output entropies of the source, the first and second transducers respectively, then $H_1' \geq H_2' \geq H_3' = H_1'$ and therefore $H_1' = H_2'$.

Suppose we have a system of constraints on possible sequences of the type which can be represented by a linear graph as in Fig. 2. If probabilities $p_{ij}^{(s)}$ were assigned to the various lines connecting state i to state j this would become a source. There is one particular assignment which maximizes the resulting entropy (see Appendix 4).

Theorem 8: Let the system of constraints considered as a channel have a capacity $C = \log W$. If we assign

$$p_{ij}^{(s)} = \frac{B_j}{B_i} W^{-l_{ij}^{(s)}}$$

where $l_{ij}^{(s)}$ is the duration of the s^{th} symbol leading from state i to state j and the B_i satisfy

$$B_i = \sum_{s,j} B_j W^{-l_{ij}^{(s)}}$$

then H is maximized and equal to C.

By proper assignment of the transition probabilities the entropy of symbols on a channel can be maximized at the channel capacity.

9. The Fundamental Theorem for a Noiseless Channel

We will now justify our interpretation of H as the rate of generating information by proving that H determines the channel capacity required with most efficient coding.

Theorem 9: Let a source have entropy H (bits per symbol) and a channel have a capacity C (bits per second). Then it is possible to encode the output of the source in such a way as to transmit at the average rate $\frac{C}{H} - \epsilon$ symbols per second over the channel where ϵ is arbitrarily small. It is not possible to transmit at an average rate greater than $\frac{C}{H}$.

The converse part of the theorem, that $\frac{C}{H}$ cannot be exceeded, may be proved by noting that the entropy of the channel input per second is equal to that of the source, since the transmitter must be non-singular, and also this entropy cannot exceed the channel capacity. Hence $H' \leq C$ and the number of symbols per second $= H'/H \leq C/H$.

The first part of the theorem will be proved in two different ways. The first method is to consider the set of all sequences of N symbols produced by the source. For N large we can divide these into two groups, one containing less than $2^{(H+\eta)N}$ members and the second containing less than 2^{RN} members (where R is the logarithm of the number of different symbols) and having a total probability less than μ. As N increases η and μ approach zero. The number of signals of duration T in the channel is greater than $2^{(C-\theta)T}$ with θ small when T is large. If we choose

$$T = \left(\frac{H}{C} + \lambda\right) N$$

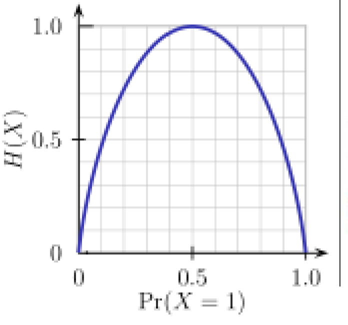
then there will be a sufficient number of sequences of channel symbols for the high probability group when N and T are sufficiently large (however small λ) and also some additional ones. The high probability group is coded in an arbitrary one-to-one way into this set. The remaining sequences are represented by larger sequences, starting and ending with one of the sequences not used for the high probability group. This special sequence acts as a start and stop signal for a different code. In between a sufficient time is allowed to give enough different sequences for all the low probability messages. This will require

Entropy and Randomness

- Entropy measures the expected "uncertainly" (or "surprise") associated with a random variable.

- Entropy quantifies the "information content" and represents a lower bound on the best possible lossless compression.

- Ex: a random fair coin has entropy of 1 bit.
  A biased coin has lower entropy than fair coin.
  A two-headed coin has zero entropy.



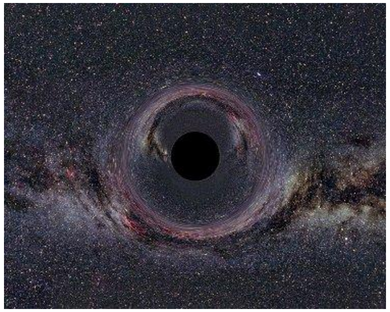
- The string 0000000000000… has zero entropy.
- English text has entropy rate of 0.6 to 1.5 bits per letter.

Q: How do you simulate a fair coin with a biased coin of unknown but fixed bias?

A [von Neumann]: Look at pairs of flips. HT and TH both occur with equal probability of p(1-p), and ignore HH and TT pairs.

Entropy and Randomness

- Information entropy is an analogue of thermodynamic entropy in physics / statistical mechanics, and von Neumann entropy in quantum mechanics.
- Second law of thermodynamics: entropy of an isolated system can not decrease over time.
- Entropy as "disorder" or "chaos".
- Entropy as the "arrow of time".
- "Heat death of the universe" / black holes



- Quantum computing uses a quantum information theory to generalize classical information theory.

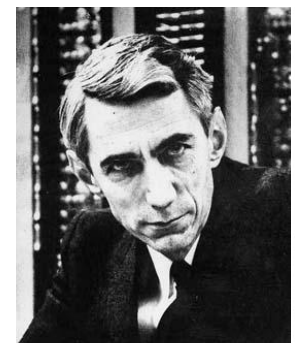
Theorem: String compressibility decreases as entropy increases.

Theorem: Most strings are not (losslessly) compressible.

Corollary: Most strings are random!

"My greatest concern was what to call it. I thought of calling it 'information', but the word was overly used, so I decided to call it 'uncertainty'. When I discussed it with John von Neumann, he had a better idea. Von Neumann told me, 'You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage.' "

- Claude Shannon on his conversation with John von Neumann regarding what name to give to the "measure of uncertainty" or attenuation in phone-line signals (1949)



Claude Shannon - Father of the Information Age - UCSD-TV - University of California Television - Mozilla Firefox

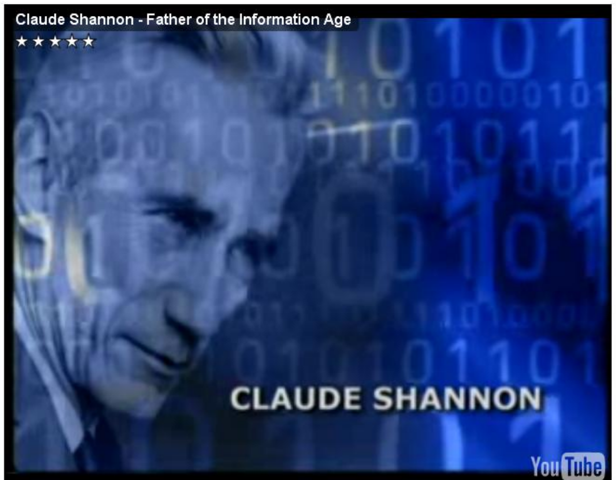
http://www.ucsd.tv/search-details.aspx?showID=6090

SEARCH UCSD-TV: GO

UC SD TV *Unlike Anything Else* University of California Television San Diego

< BACK PROGRAMS SCHEDULE WATCH/LISTEN BEHIND THE SCENES DONATE SHOP ABOUT

Claude Shannon - Father of the Information Age



BUY IT

First Aired: 1/30/2002
29 minutes

Considered the founding father of the electronic communication age, Claude Shannon's work ushered in the Digital Revolution. This fascinating program explores his life and the major influence his work had on today's digital world through interviews with his friends and colleagues. (#6090)

Links / Resources

General

Claude Shannon

Teacher's P.E.T:

K-12 Educational Standards on UCTV

MORE COMPUTER SCIENCE

- On Beyond: Examining Art, Preparing for The Big One
- Technology Management Program UCSB: Innovation
- Technology Management Program UCSB: Software

> MORE PROGRAMS

MORE SCIENCE BIOGRAPHIES AND PROFILES

- "Nukes and Genomes" with Freeman Dyson and Bob Kittle
- Conversations With History: A Microbiologist's Intellectual Odyssey
- Roger Revelle Centennial Symposium: Human Capital and the Promise of Research Universities

> MORE PROGRAMS

Extra credit: watch this video!

Sign up for UCSD-TV's monthly e-newsletter: E-mail Sign Up CONTACT US FEEDBACK WHERE TO WATCH ABOUT UCSD-TV

Transferring data from v14.lscache4.c.youtube.com...

Awards and Honors
- Claude E. Shannon Award
  - Shannon Award Nomination Form
  - 1993 Shannon Lecture (ps)
  - 1993 Shannon Lecture (pdf)
  - 1994 Shannon Lecture (ps)
  - 1994 Shannon Lecture (pdf)
  - 1995 Shannon Lecture (ps)
  - 1995 Shannon Lecture (pdf)
  - 1996 Shannon Lecture (pdf)
  - 1997 Shannon Lecture (pdf)
  - 2007 Shannon Lecture (pdf)
- Aaron D. Wyner Distinguished Service Award
- Information Theory Paper Award
- ComSoc & IT Joint Paper Award
- Chapter of the Year Award
- Golden Jubilee Paper Awards
- Golden Jubilee Awards for Technological Innovation
- IEEE Fellows
- ISIT Student Paper Award

Board of Governors
Committees
Society Chapters

Claude E. Shannon Award

— filed under: awards

The Claude E. Shannon Award of the IT Society has been instituted to honor consistent and profound contributions to the field of information theory. Each Shannon Award winner is expected to present a Shannon Lecture at the following IEEE International Symposium on Information Theory. Transcripts of some of the lectures are available on-line.

Starting for the 2010 Award, the Shannon Award Committee has decided to issue an open call for nominations, preferably using the nomination form. Although anyone may make a nomination, the Committee retains the responsibility of assuring that a suitable slate of candidates is nominated, and may itself generate nominations. Nominations and optional letters of endorsement must be submitted by March 1 to the current President of the IEEE Information Theory Society.

The first Shannon Lecturer was Claude Shannon himself followed by:

- David S. Slepian (1974)
- Robert M. Fano (1976)
- Peter Elias (1977)
- Mark S. Pinsker (1978)
- J. Wolfowitz (1979)
- W. Wesley Peterson (1981)
- Irving S. Reed (1982)
- Robert Gallager (1983)
- Solomon W. Golomb (1985)
- William L. Root (1986)
- James L. Massey (1988)

NEWS
- Postdoctoral position in computational biology
- Call For Papers: Special Issue on Cognitive Wireless Networks
- Research Fellow Position: Satellite Data Communications
- 2010 Claude E. Shannon Award
- 2009 IEEE Fellows

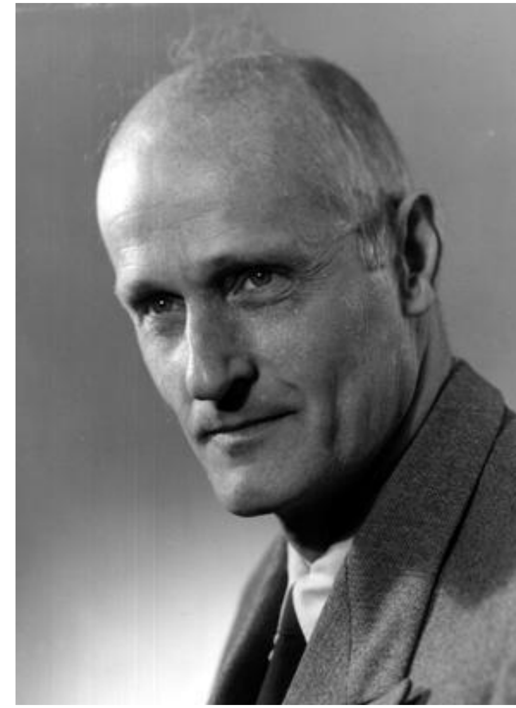
View All

UPCOMING EVENTS
Sun Oct 11 — ITW 2009, Taormina
Tue Oct 13 — BoG Meeting, ITW Taormina 2009
Mon Dec 14 — Twelfth IMA International Conference on Cryptography and Coding (IMACCC)

View All

cs.IT updates on arXiv.org
Robust THP Transceiver Designs

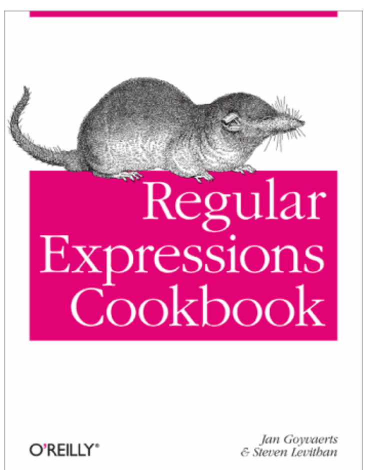
Historical Perspectives

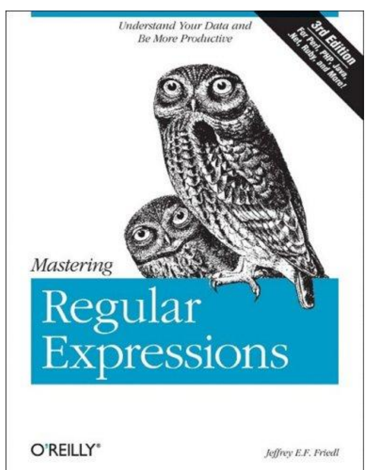


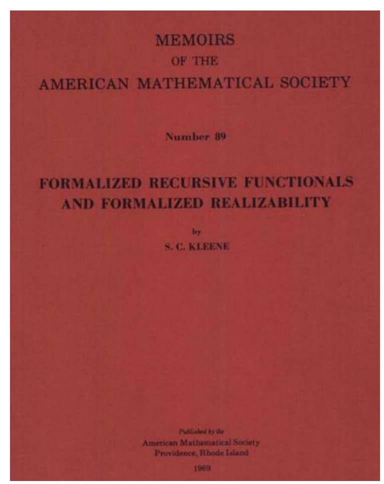
Stephen Kleene (1909-1994)

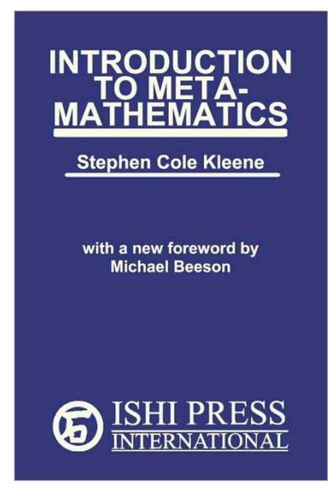
- Founded recursive function theory
- Pioneered theoretical computer science
- Student of Alonzo Church; was at the Institute for Advanced Study (1940)
- Invented regular expressions
- Kleene star / closure, Kleene algebra, Kleene recursion theorem, Kleene fixed point theorem, Kleene-Rosser paradox

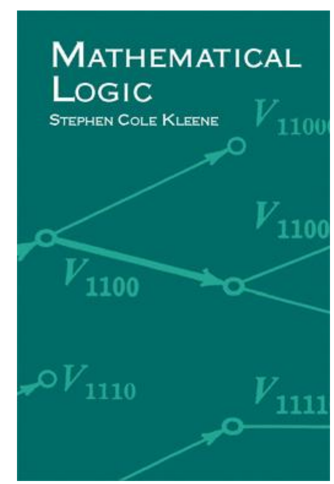
"Kleeneliness is next to Gödeliness"









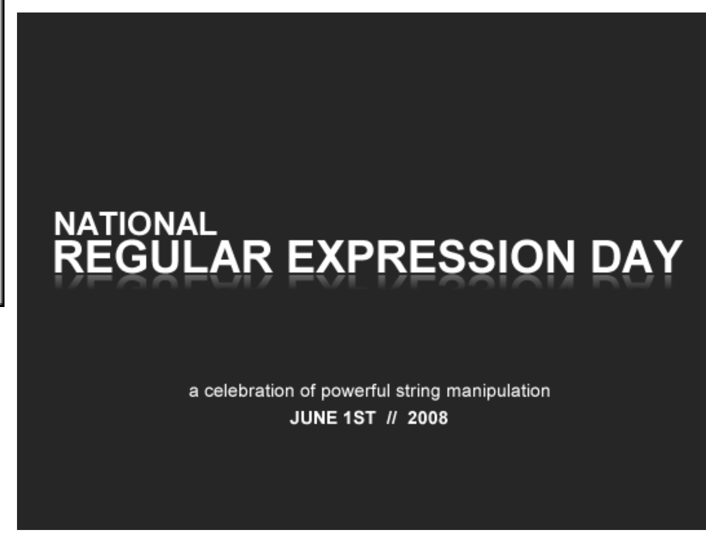



WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP


/Everybody stand back/
I know regular expressions

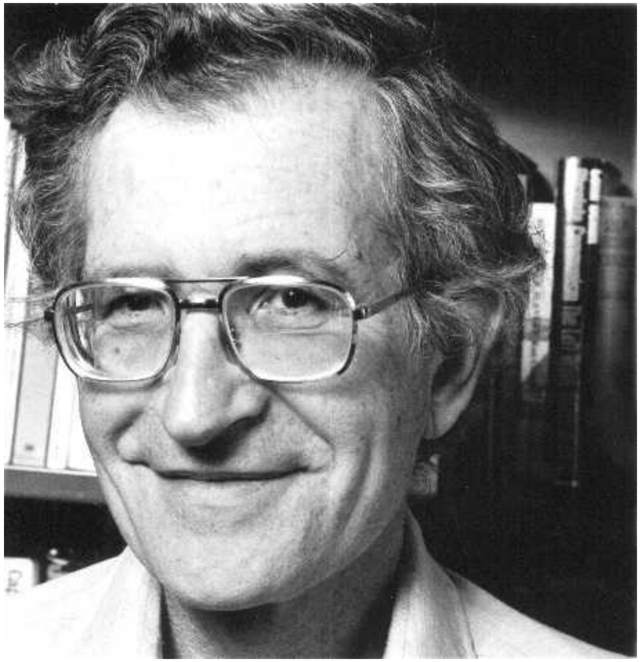

/(bb|[^b]{2})/


RegEx
Regular Expression
/h[a4@](([c<]((k)|(\|<)))|((k)|(\|<))|(x))\s+\
((d)|([t\+]h))[3ea4@]\s+p[l1][a4@]n[3e][t\+]/i
(C)2006 FTS Conventures - www.ftsconventures.com

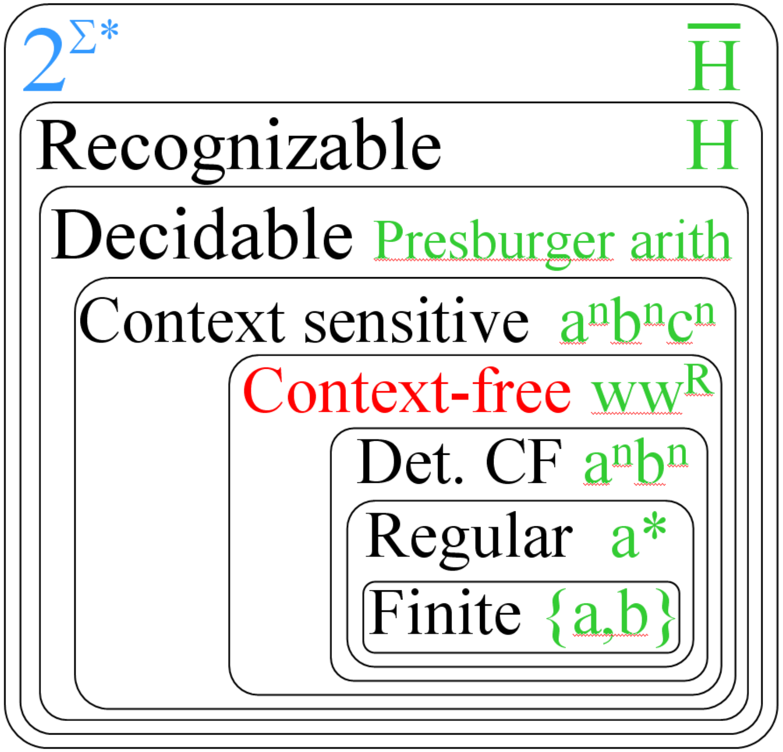

NATIONAL
REGULAR EXPRESSION DAY
a celebration of powerful string manipulation
JUNE 1ST // 2008

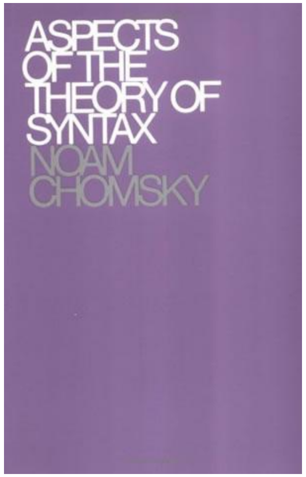
Historical Perspectives

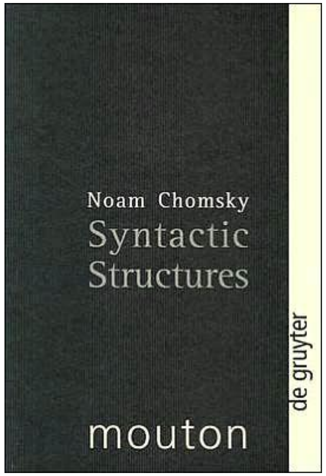
Noam Chomsky (1928-)

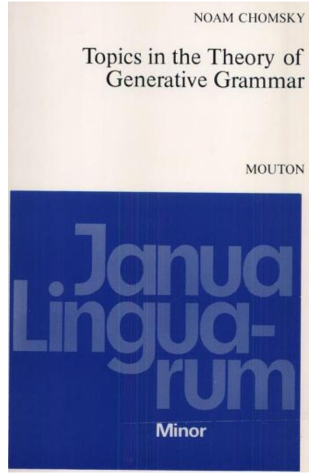


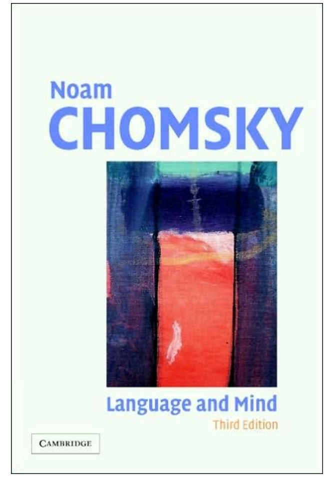
- Linguist, philosopher, cognitive scientist, political activist, dissident, author
- Father of modern linguistics
- Pioneered formal languages
- Developed generative grammars
  Invented context-free grammars
- Defined the Chomsky hierarchy
- Influenced cognitive psychology, philosophy of language and mind
- Chomskyan linguistics, Chomskyan syntax, Chomskyan models
- Critic of U.S. foreign policy
- Most widely cited living scholar
  Eighth most-cited source overall!

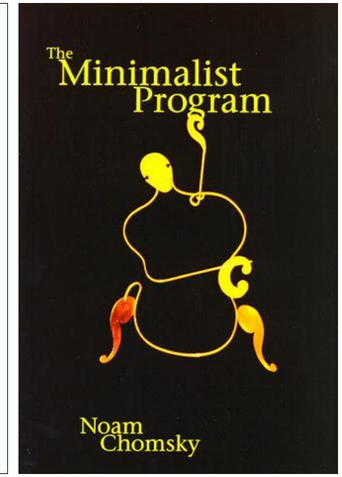


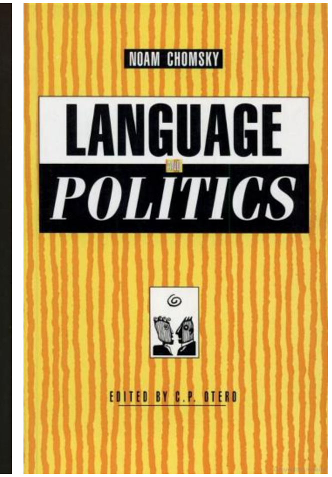

ASPECTS OF THE THEORY OF SYNTAX
NOAM CHOMSKY

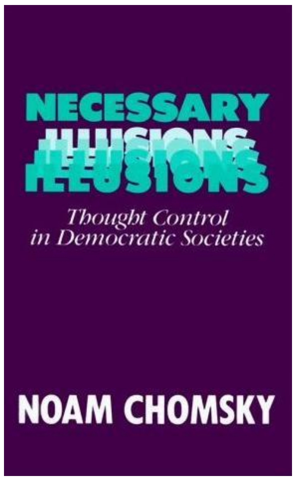

Noam Chomsky
Syntactic Structures
mouton
de gruyter

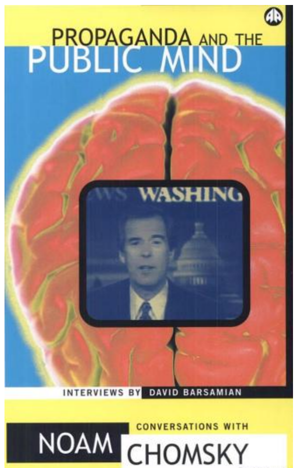

NOAM CHOMSKY
Topics in the Theory of Generative Grammar
MOUTON
Janua Lingua-rum
Minor

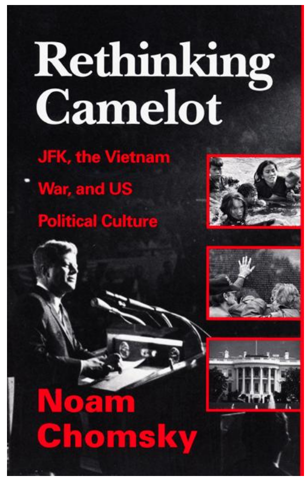

Noam
CHOMSKY
Language and Mind
Third Edition
CAMBRIDGE

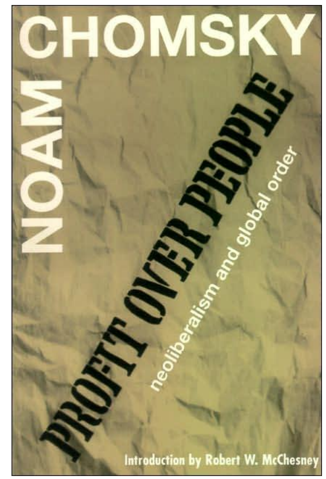

The Minimalist Program
Noam Chomsky

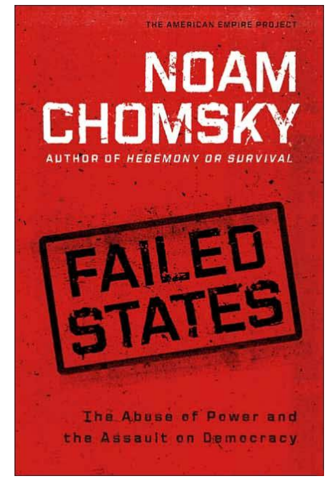

NOAM CHOMSKY
LANGUAGE
POLITICS
EDITED BY C.P. OTERO

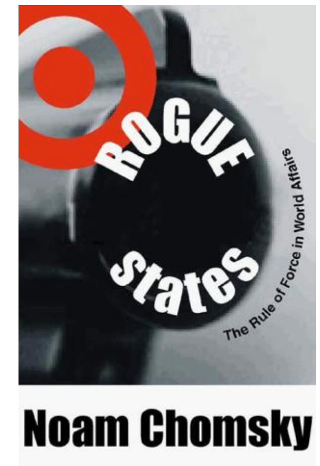

NECESSARY
ILLUSIONS
Thought Control in Democratic Societies
NOAM CHOMSKY

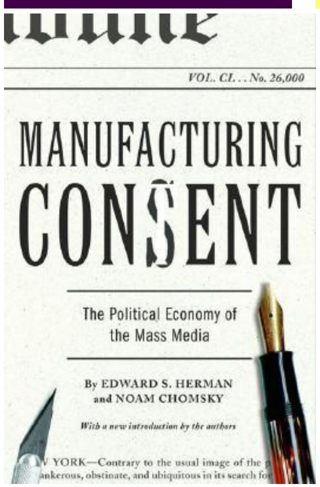

PROPAGANDA AND THE PUBLIC MIND
WASHING
INTERVIEWS BY DAVID BARSAMIAN
CONVERSATIONS WITH
NOAM CHOMSKY

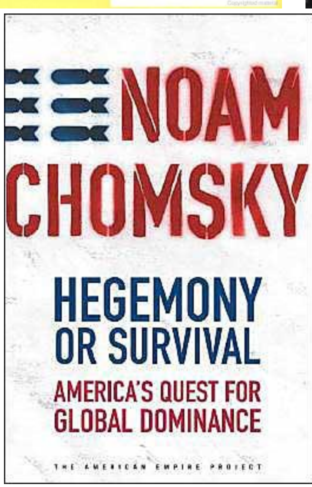

Rethinking Camelot
JFK, the Vietnam War, and US Political Culture
Noam Chomsky

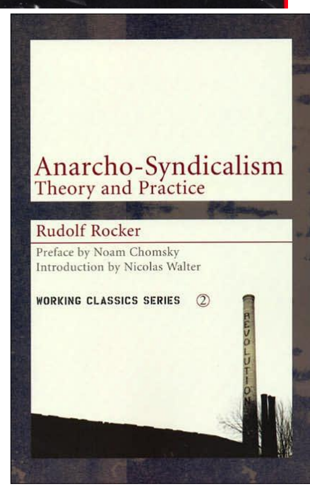

CHOMSKY
NOAM
PROFIT OVER PEOPLE
neoliberalism and global order
Introduction by Robert W. McChesney

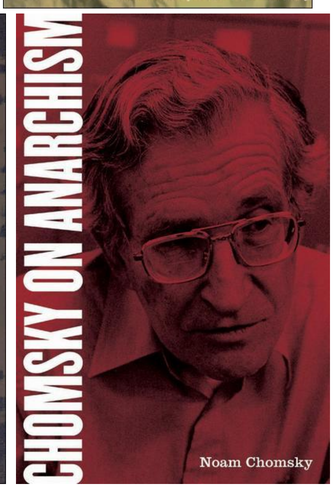

THE AMERICAN EMPIRE PROJECT
NOAM CHOMSKY
AUTHOR OF HEGEMONY OR SURVIVAL
FAILED STATES
The Abuse of Power and the Assault on Democracy

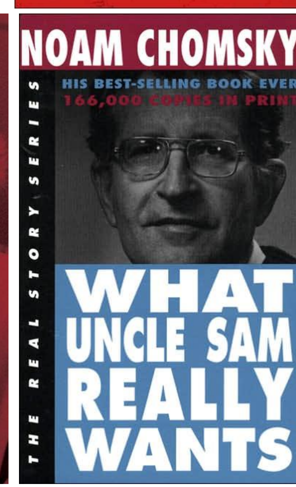

ROGUE States
The Rule of Force in World Affairs
Noam Chomsky

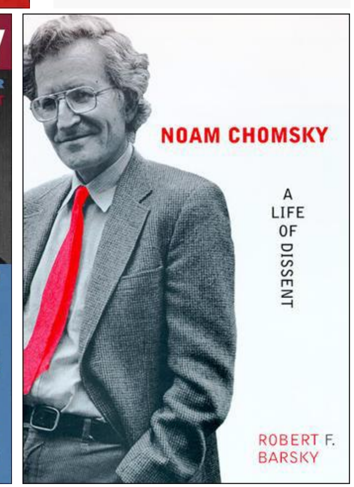

VOL. CL. . . No. 26,000
MANUFACTURING CONSENT
The Political Economy of the Mass Media
By EDWARD S. HERMAN and NOAM CHOMSKY
With a new introduction by the authors

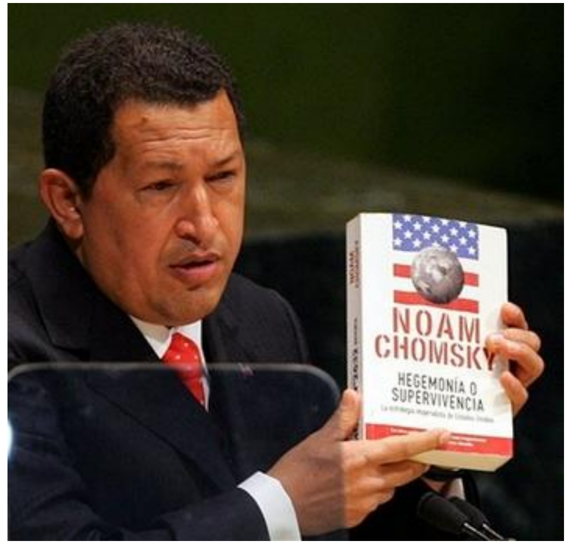

NOAM CHOMSKY
HEGEMONY OR SURVIVAL
AMERICA'S QUEST FOR GLOBAL DOMINANCE
THE AMERICAN EMPIRE PROJECT


Anarcho-Syndicalism
Theory and Practice
Rudolf Rocker
Preface by Noam Chomsky
Introduction by Nicolas Walter
WORKING CLASSICS SERIES 2
REVOLUTION

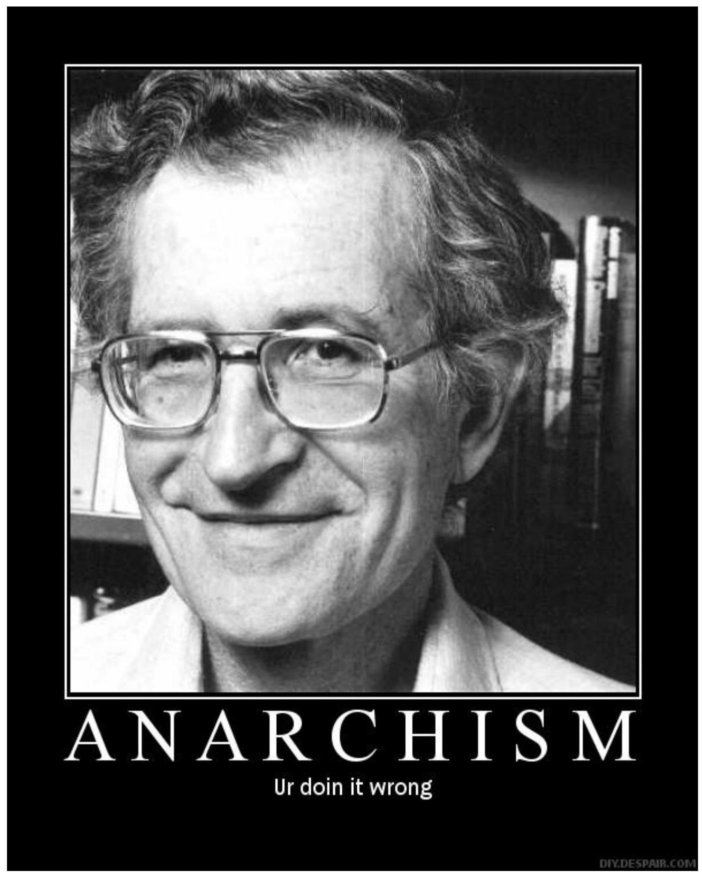

CHOMSKY ON ANARCHISM
Noam Chomsky

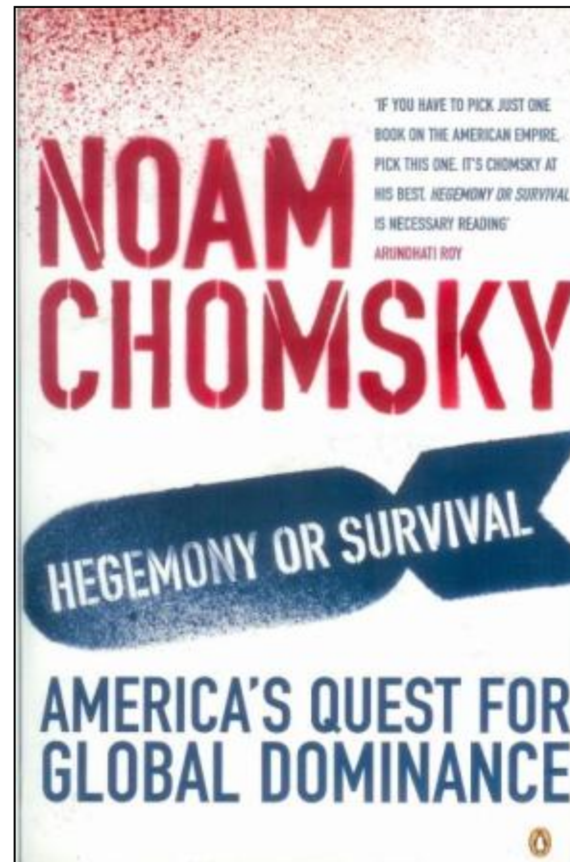

NOAM CHOMSKY
HIS BEST-SELLING BOOK EVER
166,000 COPIES IN PRINT
THE REAL STORY SERIES
WHAT UNCLE SAM REALLY WANTS


NOAM CHOMSKY
A LIFE OF DISSENT
ROBERT F. BARSKY

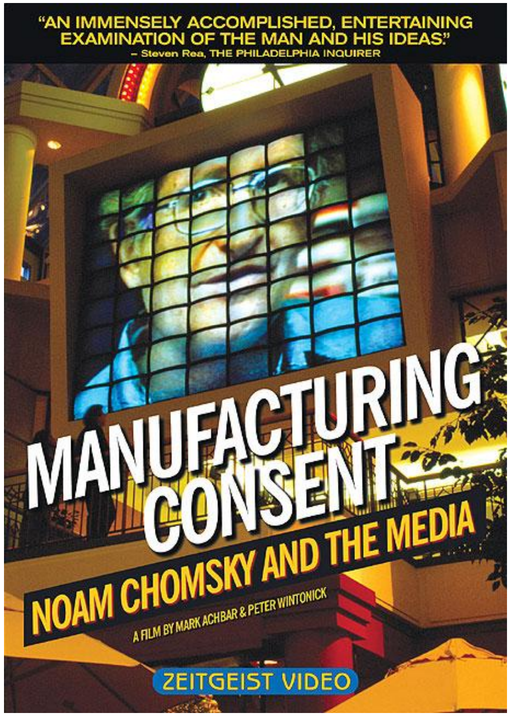

NOAM CHOMSKY
HEGEMONÍA O SUPERVIVENCIA

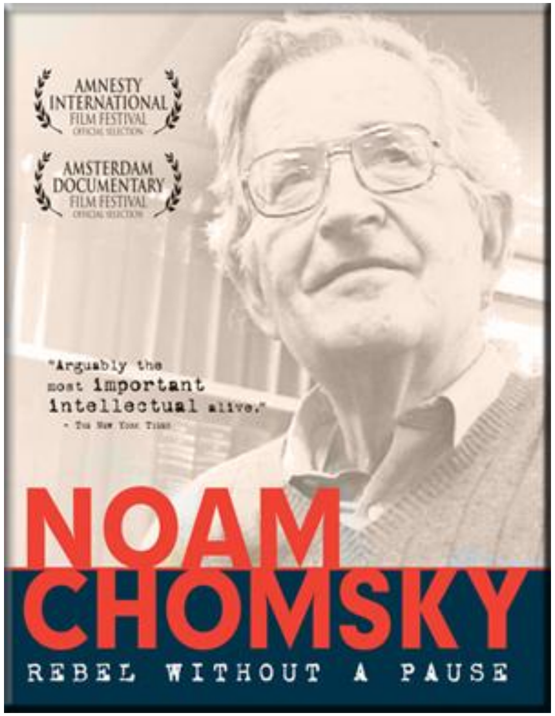

PROF. NOAM CHOMSKY
PROFESSOR OF LINGUISTICS, M.I.T.

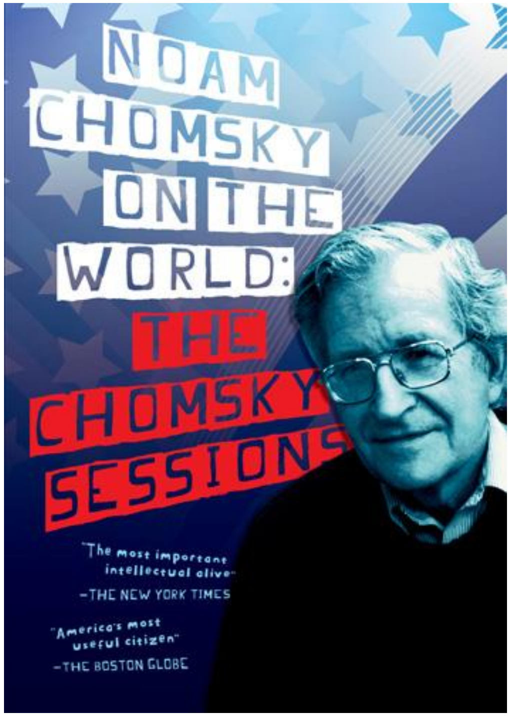

ANARCHISM
Ur doin it wrong
DIY.DESPAIR.COM

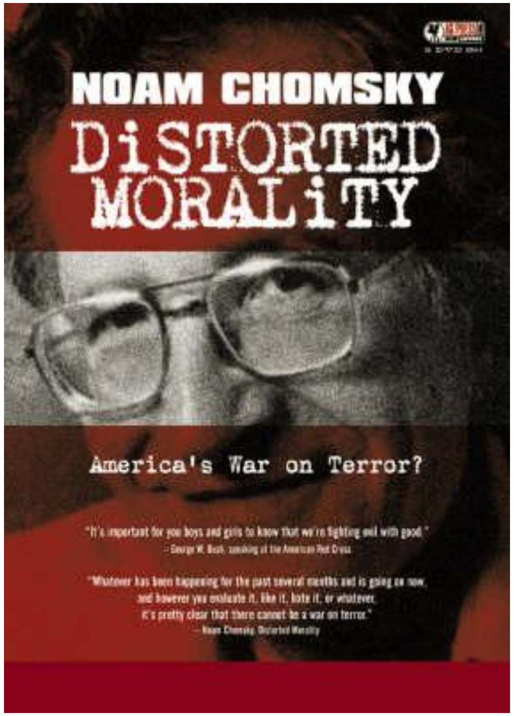

'IF YOU HAVE TO PICK JUST ONE BOOK ON THE AMERICAN EMPIRE, PICK THIS ONE. IT'S CHOMSKY AT HIS BEST. HEGEMONY OR SURVIVAL IS NECESSARY READING'
ARUNDHATI ROY
NOAM CHOMSKY
HEGEMONY OR SURVIVAL
AMERICA'S QUEST FOR GLOBAL DOMINANCE

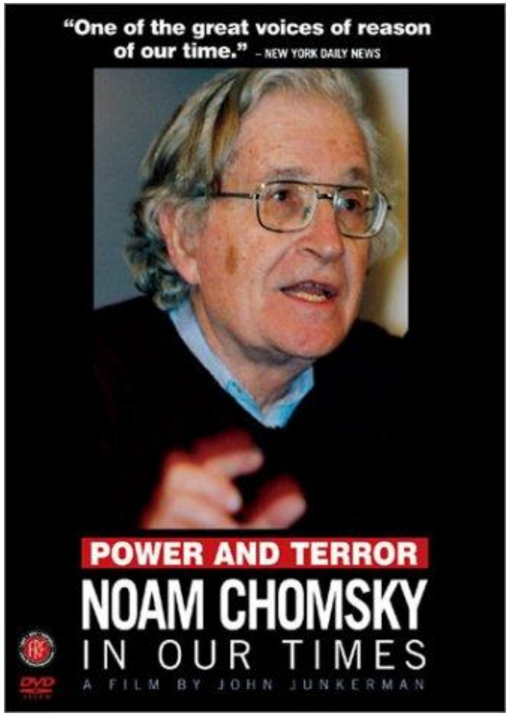
"…I must admit to taking a copy of Noam Chomsky's 'Syntactic Structures' along with me on my honeymoon in 1961 … Here was a marvelous thing: a mathematical theory of language in which I could use as a computer programmer's intuition!"
- Don Knuth on Chomsky's influence

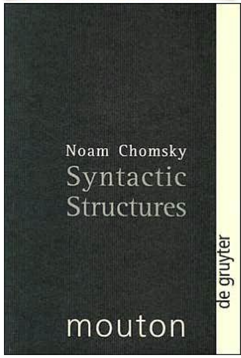




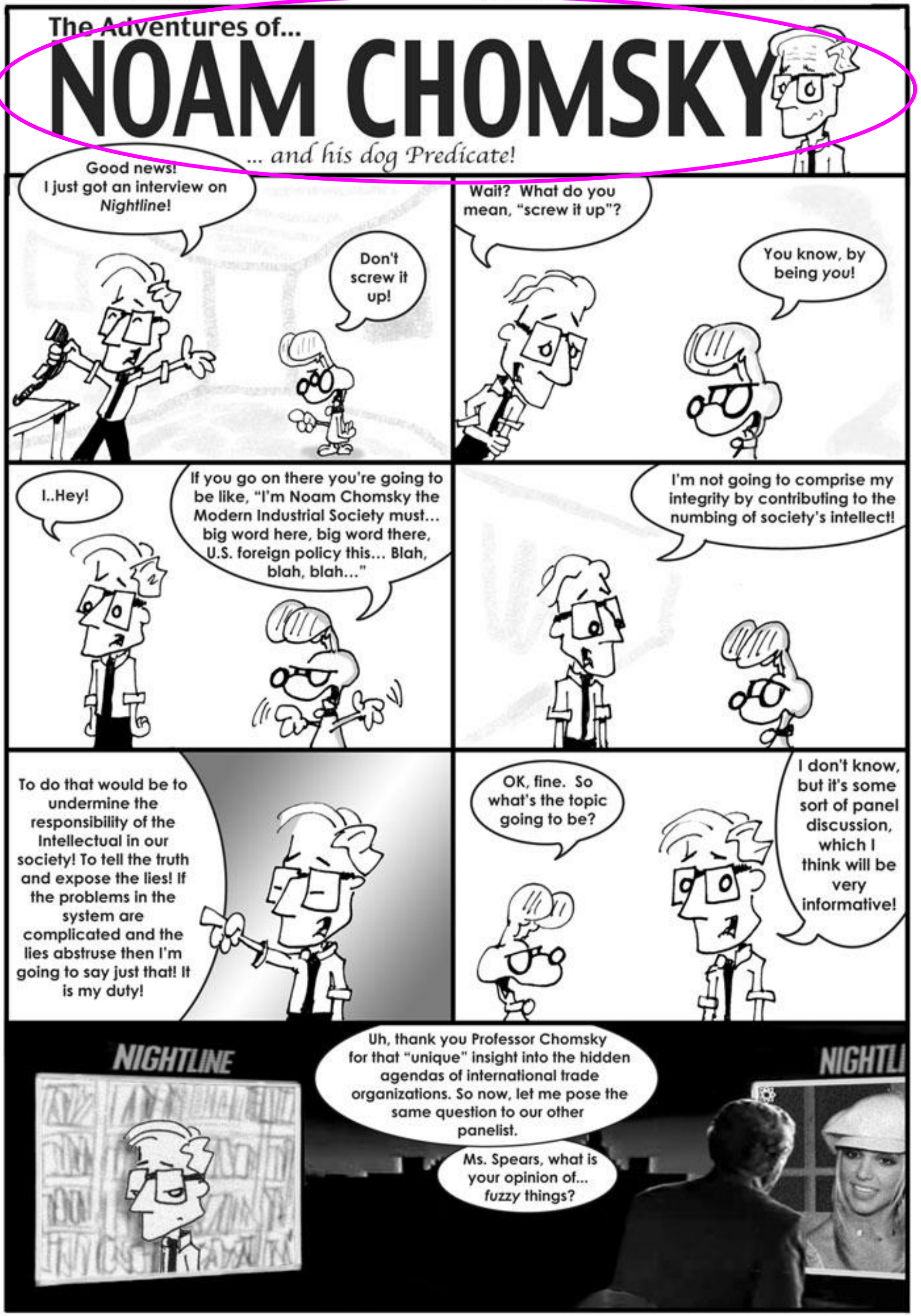




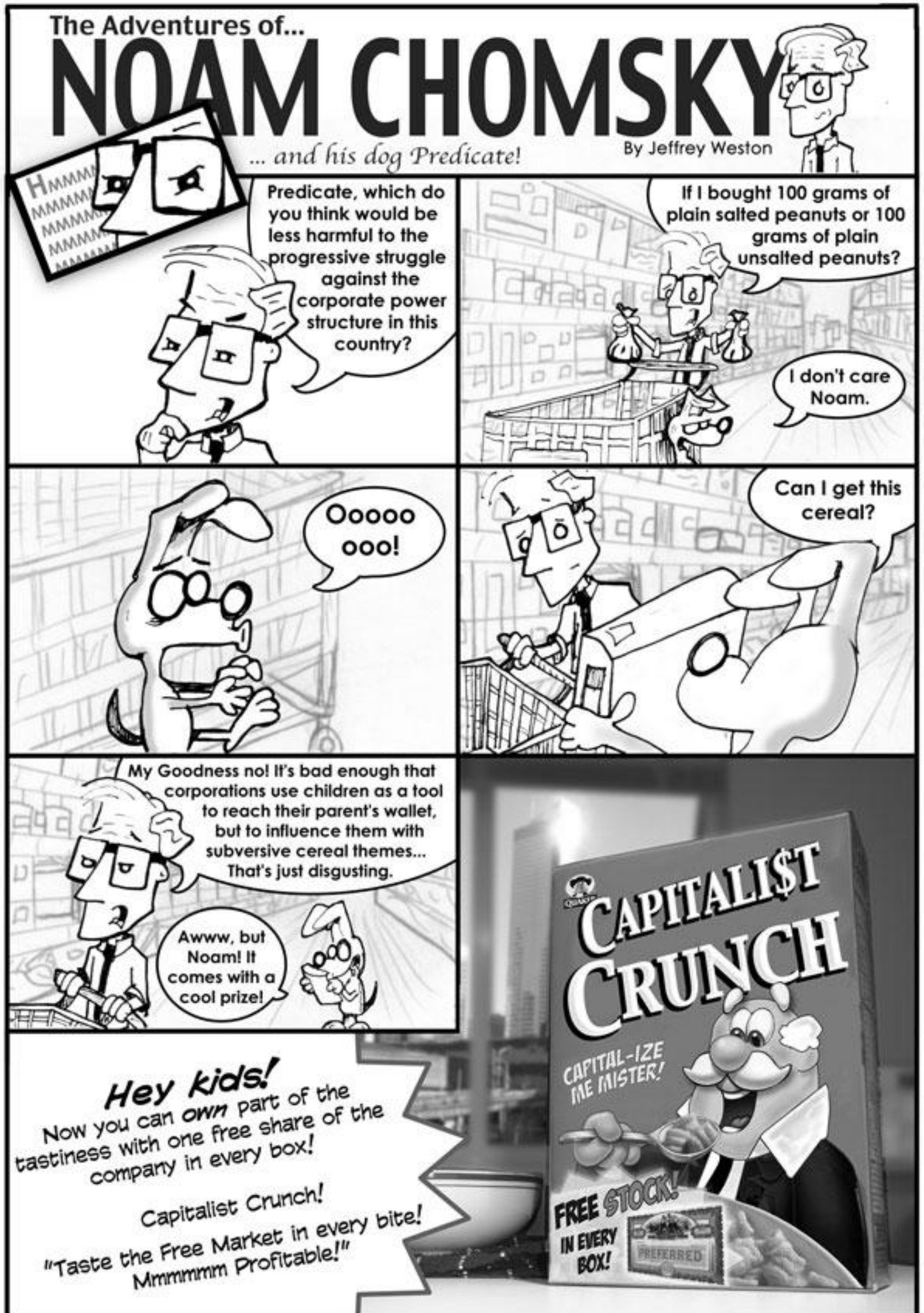




Postmodern Haircut Presents... by Jeffrey Weston, www.postmodernhaircut.com

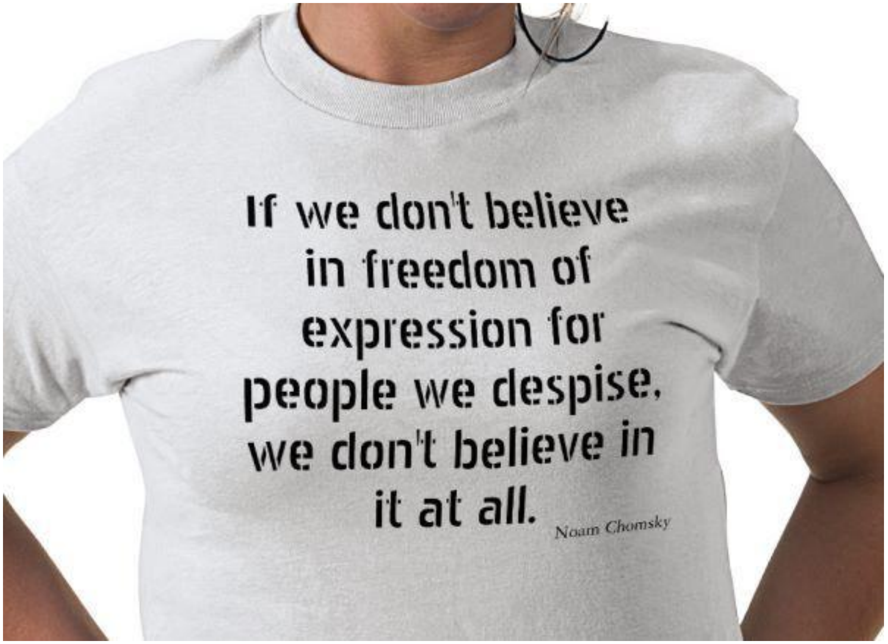
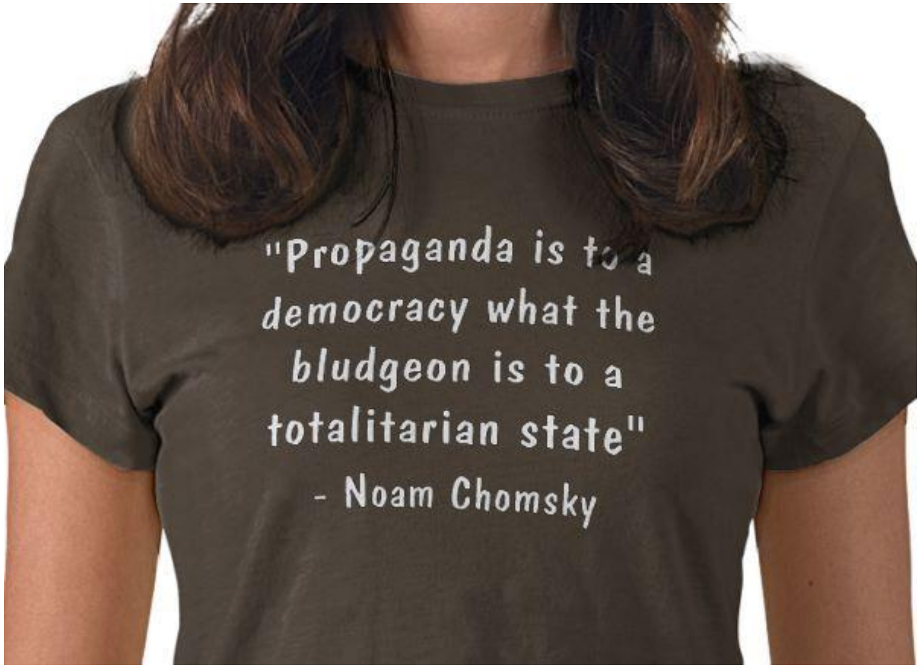


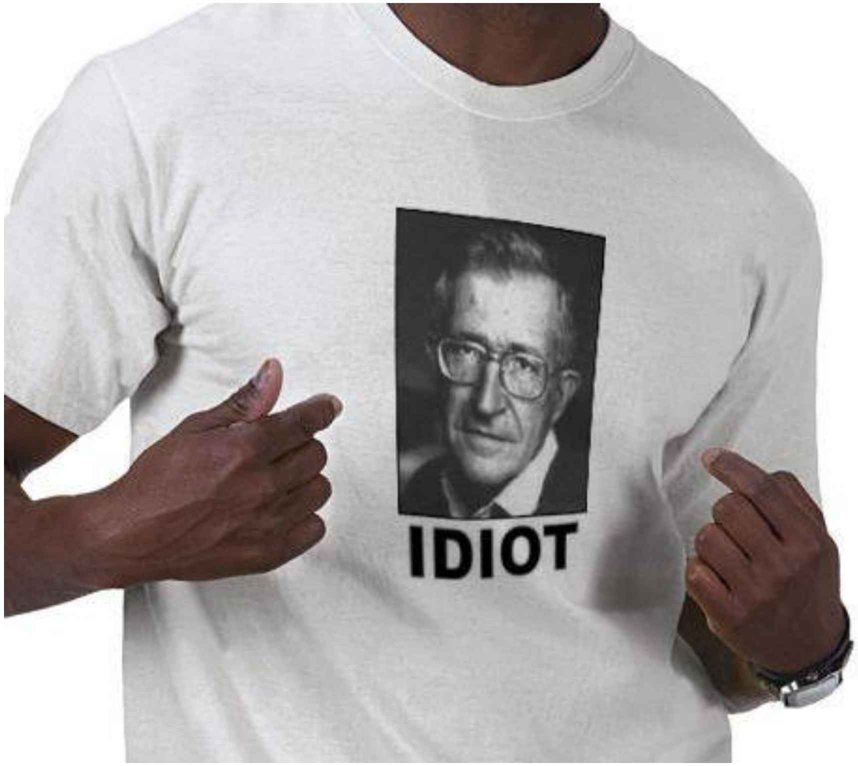
www.PostmodernHaircut.com



www.PostmodernHaircut.com

www.PostmodernHaircut.com


If we don't believe in freedom of expression for people we despise, we don't believe in it at all.
Noam Chomsky

"Propaganda is to a democracy what the bludgeon is to a totalitarian state"
- Noam Chomsky

COULD CHOMSKY BE WRONG?

IDIOT

TURING CENTENARY CONFERENCE

CiE 2012 - How the World Computes

Print this page

Local Arrangements
- Home page
- Contact Info

Scientific Arrangements
- CiE 2012
- CiE Network and Series

External Links
- The Alan Turing Year
- Alan Turing Home Page
- The Turing Digital Archive
- Bletchley Park

University of Cambridge
18 June - 23 June, 2012

CiE 2012 is one of a series of special events, running throughout the Alan Turing Year, celebrating Turing's unique impact on mathematics, computing, computer science, informatics, morphogenesis, philosophy and the wider scientific world. Its central theme is the computability-theoretic concerns underlying the broad spectrum of Turing's interests, and the contemporary research areas founded upon and animated by them. In this sense, **CiE 2012**, held in Cambridge in the week running up to the centenary of Turing's birthday, deals with the essential core of what made Turing's contribution so influential and long-lasting. **CiE 2012** promises to be an event worthy of the remarkable scientific career it commemorates.

Programme Committee: S Barry Cooper (Leeds, **Co-chair**), Anuj Dawar (Cambridge, **Co-chair**)

Organising Committee: Luca Cardelli, S Barry Cooper (Leeds), Ann Copestake, Anuj Dawar (**Chair**), Martin Hyland, Andrew Pitts

News

17.8.09
Andrew Hodges to speak at CiE 2012

22.7.09
Cambridge confirmed for CiE12

31.12.07
Turing Advisory Group founded

Picture of King's College Chapel in Cambridge

Picture of Bletchley Park Bombe rebuild

The Alan Turing Memorial in Sackville Park, Manchester