ALAN TURING

HIS WORK AND IMPACT

S. Barry Cooper • Jan van Leeuwen

Alan Turing

HIS WORK AND IMPACT

Edited by

S. BARRY COOPER
University of Leeds, UK
and
JAN VAN LEEUWEN
Utrecht University, The Netherlands

ELSEVIER
AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Elsevier
225 Wyman Street, Waltham, MA 02451, USA
The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, UK
Radarweg 29, PO Box 211, 1000 AE Amsterdam, The Netherlands

Library of Congress Cataloging-in-Publication Data
Cooper, S. B. (S. Barry)
The selected works of A.M. Turing: his work and impact / S. Barry Cooper and J. van Leeuwen.
p. cm.
Includes bibliographical references.
ISBN 978-0-12-386980-7
1. Turing, Alan Mathison, 1912–1954. 2. Mathematicians–Great Britain–Biography. 3. Computer science–Mathematics. 4. Enigma cipher system. 5. Logic, Symbolic and mathematical.
I. Leeuwen, J. van (Jan) II. Title.
QA29.T8C65 2012
510.92–dc23
[B]

2012006868

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

For information on all Elsevier publications
visit our web site at *elsevierdirect.com*

Printed and bound in USA
12 13 14 15 10 9 8 7 6 5 4 3 2 1

ISBN: 978-0-12-386980-7

Contents

This page intentionally left blank

Part I

How Do We Compute? What Can We Prove?

This page intentionally left blank

Alan Mathison Turing by Max Newman

(Bibliographic Memoirs of the Fellows of the Royal Society,
vol. 1 (Nov. 1955), pp. 253–263)

Andrew Hodges Contributes

A Comment on Newman's Biographical Memoir

Newman had to comply with official secrecy and said virtually nothing regarding Turing's work from 1939 to 1945. Although the words 'Foreign Office' would have conveyed 'codes and ciphers' to all but the most naive readers, nothing went beyond this to convey scale or significance or scientific content. Indeed Newman's account went further than *suppressio veri* and led into a *suggestio falsi.* The expression 'mild routine' probably reinforced the prevalent impression of Bletchley Park as the resort of leisured time-wasters. Turing's work had been far from routine, involving real-time day and night work on the U-boat messages, and hair-raising missions to France, the United States, and Germany. It also required great intellectual originality. Newman could probably have given a clue to its content by making a reference to I. J. Good's 1950 book *Probability and the weighing of evidence*. But there was no such hint, and the 1955 reader could never have guessed that Newman had headed the section that used the most advanced electronic technology and Turing's statistical theory to break Hitler's messages.

A more surprising feature of Newman's account is the claim that 'the designers' of 'the new automatic computing machines' had worked in ignorance of Turing's universal machine. This is an odd expression since Turing himself was one such designer, as Newman's reference to 'the first plan of the ACE' makes clear, and obviously he knew of his own theory. Moreover, this plan was a very early one submitted to the NPL in March 1946. Newman can therefore only have meant that von Neumann's report of June 1945 was written in ignorance of Turing's work. The origin of the digital computer is a major point of interest in the history of science, and it seems strange that Newman lent his authority to such an oblique and vague comment on it, with an implicit assertion about von Neumann that is at variance with other evidence. Newman's statement is also misleading in its implication that Turing only turned his attention to computers in the summer of 1945 after learning of von Neumann's design. As it happens, Newman had actually written to von Neumann on 8 February 1946 with a sharply worded statement about British developments, asserting their early start and intellectual independence.[1] Already he was applying to the Royal Society for a large grant to fund what became the Manchester computer. 'By about 18 months ago', he wrote, 'I had decided to try my hand at starting up a machine unit... This was before I knew anything of the American work... I am of course in close touch with Turing...' The date of '18 months ago' is that of August 1944. In the light of what was revealed over 20 years later, it seems obvious that the success of the electronic Colossus after D-Day prompted discussion between Turing and Newman of how the logic of the universal machine could be implemented in a practical form. All this pre-1945 history was obliterated by Newman's account in 1955. It is of course very possible that the overpowering nature of official secrecy deterred Newman from giving even the faintest hint of his own and Turing's wartime experience at Bletchley Park. Unfortunately this omission contributed to a distortion of the historical record.

[1] Letter in the von Neumann archive, Library of Congress, Washington D.C. Quoted by A. Hodges *Alan Turing: the enigma,* p. 341.

Alan Turing: His Work and Impact. DOI: 10.1016/B978-0-12-386980-7.00001-1

By courtesy of the National Portrait Gallery, London

ALAN MATHISON TURING

1912–1954

The sudden death of Alan Turing on 7 June 1954 deprived mathematics and science of a great original mind at the height of its power. After some years of scientific indecision, since the end of the war, Turing had found, in his chemical theory of growth and form, a theme that gave the fullest scope for his rare combination of abilities, as a mathematical analyst with a flair for machine computing, and a natural philosopher full of bold original ideas. The preliminary report of 1952, and the account that will appear posthumously, describe only his first rough sketch of this theory, and the unfulfilled design must remain a painful reminder of the loss that his early death has caused to science.

Alan Mathison Turing was born in London on 23 June 1912, the son of Julius Mathison Turing, of the Indian Civil Service, and of Ethel Sara Turing (*née* Stoney). The name 'Turing' is of Scottish, perhaps ultimately of Norman origin, the final *g* being an addition made by Sir William Turing, of Aberdeenshire, in the reign James VI and I. The Stoneys, an English-Irish family of Yorkshire origin, produced some distinguished physicists and engineers in the nineteenth century, three of whom became Fellows of the Society; and Edith A. Stoney was one of the early women equal-to-wranglers at Cambridge (bracketed with 17th Wrangler, 1893).

Alan Turing's interest in science began early and never wavered. Both at his preparatory schools and later at Sherborne, which he entered in 1926, the contrast between his absorbed interest in science and mathematics, and his indifference to Latin and 'English subjects' perplexed and distressed his teachers, bent on giving him a well-balanced education. Many of the characteristics that were strongly marked in his later life can already be clearly seen in remembered incidents of this time: his particular delight in problems, large or small, that enabled him to combine theory with the kind of experiments he could carry out with his own hands, with the help of whatever apparatus was at hand; his strong preference for working everything out from first principles instead of borrowing from others—a habit which gave freshness and independence to his work, but also undoubtedly slowed him down, and later on made him a difficult author to read. At school homemade experiments in his study did not fit well into the routine of the house: a letter from his housemaster mentions 'Heaven knows what witches' brew blazing on a naked wooden window sill'. But before he left school his abilities, and his obvious seriousness of purpose, had won him respect and affection, and even tolerance for his own peculiar methods.

In 1931 he entered King's College, Cambridge, as a mathematical scholar. A second class in Part I of the Tripos showed him still determined not to spend time on subjects that did not interest him. In Part II he was a Wrangler, with 'b^*', and he won a Smith's Prize in 1936. He was elected a Fellow of King's in 1935, for a dissertation on the Central Limit Theorem of probability (which he discovered anew, in ignorance of recent previous work).

It was in 1935 that he first began to work in mathematical logic, and almost immediately started on the investigation that was to lead to his best known results, on computable numbers and the

---

Royal Society Memoir (Max Newman)
Reproduced from the Bibliographic Memoirs of the Fellows of the Royal Society, vol. 1 (November 1955) pp. 253–263 by kind permission of the Royal Society and of Edward and William Newman.

'Turing machine'. The paper attracted attention as soon as it appeared and the resulting correspondence led to his spending the next two years (1936–8) in Princeton, working with Professor Alonzo Church, the second of them as Proctor Fellow.

In 1938 Turing returned to Cambridge; in 1939 the war broke out. For the next six years he was fully occupied with his duties for the Foreign Office. These years were happy enough, perhaps the happiest of his life, with full scope for his inventiveness, a mild routine to shape the day, and a congenial set of fellow-workers. But the loss to his scientific work of the years between the ages of 27 and 33 was a cruel one. Three remarkable papers written just before the war, on three diverse mathematical subjects, show the quality of the work that might have been produced if he had settled down to work on some big problem at that critical time. For his work for the Foreign Office he was awarded the O.B.E.

At the end of the war many circumstances combined to turn his attention to the new automatic computing machines. They were in principle realizations of the 'universal machine' which he had described in the 1937 paper for the purpose of a logical argument, though their designers did not yet know of Turing's work. Besides this theoretical link, there was a strong attraction in the many-sided nature of the work, ranging from electric circuit design to the entirely new field or organizing mathematical problems for a machine. He decided to decline an offer of a Cambridge University Lectureship, and join the group that was being formed at the National Physical Laboratory for the design, construction and use of a large automatic computing machine. In the three years (1945–8) that this association lasted he made the first plan of the ACE, the N.P.L's automatic computer, and did a great deal of pioneering work in the design of sub-routines.

In 1948 he was appointed to a Readership in the University of Manchester, where work was beginning on the construction of a computing machine by F. C. Williams and T. Kilburn. The expectation was that Turing would lead the mathematical side of the work, and for a few years he continued to work, first on the design of the sub-routines out of which the larger programmes for such a machine are built, and then, as this kind work became standardized, on more general problems of numerical analysis. From 1950 onward he turned back for a while to mathematics and finally to his biological theory. But he remained in close contact with the Computing Machine Laboratory, whose members found him ready to tackle the mathematical problems that arose in their work, and what is more, to find the answers, by that combination of powerful mathematical analysis and intuitive short cuts that showed him at heart more of an applied than a pure mathematician.

He was elected to the Fellowship of the Society in 1951.

For recreation he turned mostly to those 'home-made' projects and experiments, self-contained both in theory and practice, that have already been mentioned: they remained a ruling passion up to the last hours of his life. The rule of the game was that everything was to be done with the materials at hand, and worked out from data to be found in the house, or in his own head. This sort of self-sufficiency stood him in good stead in starting on his theory of 'morphogenesis', where the preliminary reading would have drowned out a more orthodox approach. In everyday life it led to a certain fondness for the gimcrack, for example the famous Bletchley bicycle, the chain of which would stay on if the rider counted his pedal-strokes and executed a certain manoeuvre after every seventeen strokes.

After the war, feeling in need of violent exercise, he took to long distance running, and found that he was very successful at it. He won the 3 miles and 10 miles championships of his club (the Walton Athletic Club), both in record time, and was placed fifth in the A.A.A. Marathon race in 1947. He thought it quite natural to put this accomplishment to practical use from time to time, for example by running some nine miles from Teddington to a technical conference at the Post Office Research Station in North London, when the public transport proved tedious.

In conversation he had a gift for comical but brilliantly apt analogies, which found its full scope in the discussions on 'brains *v.* machines' of the late 1940's. He delighted in confounding those who, as he thought, too easily assumed that the two things are separated by an impassable gulf, by challenging them to produce an examination paper that could be passed by a man, but not by a

machine. The unexpected element in human behaviour he proposed, half seriously, to imitate by a random element, or roulette-wheel, in the machine. This, he said, would enable proud owners to say 'My machine' (instead of 'My little boy') 'said such a funny thing this morning'.

Those who knew Turing will remember most vividly the enthusiasm and excitement with which he would pursue any idea that caught his interest, from a conversational hare to a difficult scientific problem. Nor was it only the pleasure of the chase that inspired him. He would take the greatest pains over services, large or small, to his friends. His colleagues in the computing machine laboratory found him still as ready as ever with his help for their problems when his own interests were fully engaged with his bio-chemical theory; and, as another instance, he gave an immense amount of thought and care to the selection of the presents which he gave to his friends and their children at Christmas.

His death, at a time when he was fully absorbed in his scientific work, was a great and sad loss to his friends, as well as to the wider world of science.

Scientific work

The varied titles of Turing's published work disguise its unity of purpose. The central problem with which he started, and to which he constantly returned, is the extent and the limitations of mechanistic explanations of nature. All his work, except for three papers in pure mathematics (1935*b*, 1938*a* and *b*) grew naturally out of the technical problems encountered in these inquiries. His way of tackling the problem was not by philosophical discussion of general principles, but by mathematical proof of certain limited results: in the first instance the impossibility of the too sanguine programme for the complete mechanization of mathematics, and in his final work, the possibility of, at any rate, a partial explanation of the phenomena of organic growth by the 'blind' operation of chemical laws.

1. *Mathematical logic*

The Hilbert decision-programme of the 1920's and 30's had for its objective the discovery of a general process, applicable to any mathematical theorem expressed in fully symbolical form, for deciding the truth or falsehood of the theorem. A first blow was dealt at the prospects of finding this new philosopher's stone by Gödel's incompleteness theorem (1931), which made it clear that truth or falsehood of *A* could not be equated to provability of *A* or not-*A* in any finitely based logic, chosen once for all; but there still remained in principle the possibility of finding a mechanical process for deciding whether *A*, or not-*A*, or neither, was formally provable in a given system. Many were convinced that no such process was possible, but Turing set out to demonstrate the impossibility rigorously. The first step was evidently to give a definition of 'decision process' sufficiently exact to form the basis of a mathematical proof of impossibility. To the question 'What is a "mechanical" process?' Turing returned the characteristic answer 'Something that can be done by a machine, and he embarked on the highly congenial task of analyzing the general notion of a computing machine. It is difficult to-day to realize how bold an innovation it was to introduce talk about paper tapes and patterns punched in them, into discussions of the foundations of mathematics. It is worth while quoting from his paper (1937*a*) the paragraph in which the computing machine is first introduced, both for the sake of its content and to give the flavour of Turing's writings.

'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, ..., 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 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 succeeding paragraphs he gave arguments for believing that a machine of this kind could be made to do any piece of work which could be done by a human computer obeying explicit instructions given to him before the work starts. A machine of the kind he had described could be made for computing the successive digits of π, another for computing the successive prime numbers, and so forth. Such a machine is completely specified by a table, which states how it moves from each of the finite sets of possible 'configurations' to another. In the computations mentioned above, of π and of the successive primes, the machine may be supposed to be designed for its special purpose. It is supplied with a blank tape and started off. But we may also imagine a machine supplied with a tape already bearing a pattern which will influence its subsequent behaviour, and this pattern might be the table, suitably encoded, of a particular computing machine, X. It could be arranged that this tape would cause the machine, M, into which it was inserted to behave like machine X. Turing proved the fundamental result that there is a 'universal' machine, U (of which he gave the table), which can be made to do the work of any assigned special-purpose machine, that is to say to carry out any piece of computing, if a tape bearing suitable 'instructions' is inserted into it. The machine U is so constructed that, presented with a tape bearing any arbitrary pattern it will move through a determinate, in general endless, succession of configurations; and it may or may not print at least one digit, 0 or 1. If it does, the pattern is 'circle-free'. It is therefore a problem, for which a decision process might be sought, to determine from inspection of a tape, whether or not it is circle-free. By means of a Cantor diagonal argument, Turing showed that no instruction-tape will cause the machine U to solve this problem, i.e. no pattern P is such that U, when presented with P followed by an arbitrary pattern Υ, will print 0 if Υ is 'circle-free', and 1 if it is not. If Turing's thesis is accepted, that the existence of a *method* for solving such a problem means the existence of a machine (or an instruction-tape for the universal machine U) that will solve it, it follows that the discovery of a process for discriminating between circle-free and other tapes is an insoluble problem, in an absolute and inescapable sense. From this basic insoluble problem it

was not difficult to infer that the Hilbert programme of finding a decision method for the axiomatic system, Z, of elementary number-theory, is also impossible.

In the application he had principally in mind, namely, the breaking down of the Hilbert programme, Turing was unluckily anticipated by a few months by Church, who proved the same result by means of his 'λ-calculus'. An offprint arrived in Cambridge just as Turing was ready to send off his manuscript. But it was soon realised that Turing's 'machine' had a significance going far beyond this particular application. It was shown by Turing (1937*b*) and others that the definitions of 'general recursive' (by Gödel in 1931 and Kleene in 1935), 'λ-definable' (by Church in 1936) and 'computable' (Turing, 1937*a*) have exactly the same scope, a fact which greatly strengthened the belief that they describe a fundamentally important body of functions. Turing's treatment has the merit of making a particularly convincing case for the acceptance of these and no other processes, as genuinely constructive; and it turned out to be well adapted for use in finding other insoluble problems, e.g., in the theory of groups and semi-groups.

Turing's other major contribution to this part of mathematical logic, the paper (1939) on systems logic based on ordinals, has received less attention than (1937*a*), perhaps owing to its difficulty. The method of Gödel for constructing an undecidable sentence in any finitely based logic, L, i.e. a sentence expressible, but neither provable nor disprovable, in L, has led to the consideration of infinite families of 'logics', L_α, one for each ordinal α, where $L_{\alpha+1}$ is formed from L_α by the adjunction as an axiom of a sentence undecidable for L_α, if such exist, and L_α for limit ordinals α has as 'provable formulae', the union of the sets $P\beta(\beta < \alpha)$, where $P\beta$ is the set of provable formulae in $\mathrm{L}\beta$. The process must terminate for some $\gamma < \omega_1$, since the total set of formulae (which does not change) is countable. This procedure opens up the possibility of finding a logic that is complete, without violating Gödel's principle, since L_α may not be finitely based if α is a limit ordinal. Rosser investigated this possibility in 1937, using the 'classical' non-constructive theory of ordinals. Turing took up the proposal, but with the proviso that, although some non-constructive steps must be made if a complete logic is to be attained, a strict watch should be kept on them. He first introduced a new theory of constructive ordinals, or rather of formulae (of Church's λ-calculus) representing ordinals; and he showed that the problem of deciding whether a formula represents an ordinal (in a plausible sense) is insoluble, in the sense of his earlier paper. A formula $\mathbf{L}$ of the λ-calculus is a *logic* if it gives a means of establishing the truth of number-theoretic theorems; formally, if $\mathbf{L}(\mathbf{A})$ conv. $\mathbf{2}$ implies that $\mathbf{A(n)}$ conv. $\mathbf{2}$ for each $\mathbf{n}$ representing a natural number. The *extent* of $\mathbf{L}$ is the set of $\mathbf{A}$'s such that $\mathbf{L}(\mathbf{A})$ conv. $\mathbf{2}$, i.e., roughly speaking, the set of $\mathbf{A}$'s for which $\mathbf{L}$ proves $\mathbf{A}(\mathbf{n})$ is true for all $\mathbf{n}$. An *ordinal logic* is now defined to be a formula $\mathbf{\Lambda}$, such that $\mathbf{\Lambda(\Omega)}$ is a logic whenever $\mathbf{\Omega}$ represents an ordinal; and $\mathbf{\Lambda}$ is *complete* if every number-theoretic theorem that is true, is probable in $\mathbf{\Lambda}(\mathbf{\Omega})$ for some $\mathbf{\Omega}$, i.e. if given $\mathbf{A}$ such that $\mathbf{A(n)}$ conv. $\mathbf{2}$ for each $\mathbf{n}$ representing a natural number, $\mathbf{\Lambda}(\mathbf{\Omega},\mathbf{A})$ conv. $\mathbf{2}$ for some $\mathbf{\Omega}$ (depending on $\mathbf{A}$). It is next shown, by an example, that formulae $\mathbf{\Omega}_1$, $\mathbf{\Omega}_2$ may represent the same ordinal, but yet make $\mathbf{\Lambda}(\mathbf{\Omega}_1)$ and $\mathbf{\Lambda}(\mathbf{\Omega}_2)$ different logics, in the sense that they have different extents. An ordinal logic for which this cannot happen is *invariant*. It is only in invariant logics that the 'depth' of a theorem can be measured by the size of the ordinal required for its proof. The main theorems of the paper state (1) that complete ordinal logics and invariant ordinal logics exist, (2) that no complete and invariant ordinal logic exists.

This paper is full of interesting suggestions and ideas. In §4 Turing considers, as a system with the minimal departure from constructiveness, one in which number-theoretic problems of some class are arbitrarily assumed to be soluble: as he puts it, 'Let us suppose that we are supplied with some unspecified means of solving number-theoretic problems; a kind of oracle, as it were.' The availability of the oracle is the 'infinite' ingredient necessary to escape the Gödel principle. It also obviously resembles the stages in the construction of a proof by a mathematician where he 'has an idea', as distinct from making mechanical use of a method. The discussion of this and related matters in §11 ('The purpose of ordinal logics') throws much light on Turing's views on the place of intuition in mathematical proof. In the final rather difficult §12 the idea adumbrated by Hilbert in 1922 of recursive definitions of order-types other than ω received its first detailed exposition.

Besides these two pioneering works, and the papers (1937*b*, *c*), arising directly out of them, Turing published four papers of predominantly logical interest. (A) The paper (1942*a*), with M. H. A. Newman, on a formal question in Church's theory of types. (B) A 'practical form of type-theory' (1948*b*) is intended to give Russell's theory of types a form that could be used in ordinary mathematics. Since the more flexible Zermelo-von Neumann set-theory has been generally preferred to type-theory by mathematicians, this paper has received little attention. It contains a number of interesting ideas, in particular a definition of 'equivalence' between logical systems (p. 89). (C) The use of dots as brackets (1942*b*), an elaborate discussion of punctuation in symbolic logic. Finally, (D) contains the proof (1950*a*) of the insolubility of the word-problem for semi-groups with cancellation. A finitely generated semi-group *without* cancellation is determined by choosing a finite set of pairs of words, $(A_i,\ B_i)(i = 1_1 \ldots k)$ of some alphabet, and declaring two words to be 'equivalent' if they can be proved so by the use of the equations $PA_i\mathrm{Q} = PB_i\mathrm{Q}$, where P and Q can be arbitrary words (possibly empty). The word-problem for such a semi-group is to find a process which will decide whether or not two given words are equivalent. The insolubility of this problem can be brought into relation with the fundamental insoluble machine-tape problem. The table of a computing machine states, for each configuration, what is the configuration that follows it. Since a configuration can be denoted by a 'word', in letters representing the internal configurations and tape-symbols, this table gives a set of pairs words which, when suitably modified, determine a semi-group with insoluble word problem. So much was proved by E. L. Post in 1947. The question becomes much more difficult if the semi-group is required to satisfy the cancellation laws, '$AC = BC$ implies $A = B$' and '$CA = CB$ implies $A = B$' since now a condition is imposed on account of its mathematical interest, and not because it arises naturally from the machine interpretation. This was the step taken by Turing in (1950*a*). (For a helpful discussion and analysis of this difficult paper see the long review by W. W. Boone, *J. Symbolic Logic*, **17** (1952) 74.)

2. *Three mathematical papers*

Shortly before the war Turing made his only contributions to mathematics proper.

The paper 1938*a* contains an interesting theorem on the approximation of Lie groups by finite groups: if a (connected) Lie group, L, can for arbitrary $\varepsilon > 0$ be ε-approximated by a finite group whose multiplication law is an ε-approximation to that of L, in the sense that the two products of any two elements are within ε of each other, then L must be both compact and abelian. The theory of representations of topological groups is used to apply Jordan's theorem on the abelian invariant subgroups of finite groups of linear transformations.

Paper (1938*b*) lies in the domain of classical group theory. Results of R. Baer on the extensions of a group are re-proved by a more unified and simpler method.

Paper (1943)—submitted in 1939, but delayed four years by war-time difficulties—shows that Turing's interest in practical computing goes back at least to this time. A method is given for the calculation of the Riemann zeta-function, suitable for values of t in a range not well covered by the previous work of Siegel and Titchmarsh. The paper is a straightforward but highly skilled piece of classical analysis. (The post-war paper (1953*a*) describes an attempt to apply a modified form of this process, which failed owing to machine trouble.)

3. *Computing machines*

Apart from the practical 'programmer's handbook', only two published papers (1948*a* and 1950*b*) resulted from Turing's work on machines. When binary fractions of fixed length are used (as they must be on a computing machine) for calculations involving a very large number of multiplications and divisions, the necessary rounding-off of exact products introduces cumulative errors, which gradually consume the trustworthy digits as the computation proceeds. The paper (1948*a*) investigates questions of the following type: how many figures of the answer are trustworthy if k figures

are retained in solving n linear equations in n unknowns? The answer depends on the method of solution, and a number of different ones are considered. In particular it is shown that the ordinary method of successive elimination of the variables does not lead to the very large errors that had been predicted, save in exceptional cases which can be specified.

The other paper (1950*b*) arising out of his interest in computing machines is of a very different nature. This paper, on computing machines and intelligence, contains Turing's views on some questions about which he had thought a great deal. Here he elaborates his notion of an 'examination' to test machines against men, and he examines systematically a series of arguments commonly put forward against the view that machines might be said to think. Since the paper is easily accessible and highly readable, it would be pointless to summarize it. The conversational style allows the natural clarity of Turing's thought to prevail, and the paper is a masterpiece of clear and vivid exposition.

The proposals (1953*b*) for making a computing machine play chess are amusing, and did in fact produce a defence lasting 30 moves when the method was tried against a weak player; but it is possible that Turing underestimated the gap that separates combinatory from position play.

4. *Chemical theory of morphogenesis*

For the following account of Turing's final work I am indebted to Dr N. E. Hoskin, who with Dr B. Richards is preparing an edition of the material for publication.

The work falls into two parts. In the first part, published (1952) in his lifetime, he set out to show that the phenomena of morphogenesis (growth and form of living things) could be explained by consideration of a system of chemical substances whose concentrations varied only by means of chemical reactions, and by diffusion through the containing medium. If these substances are considered as form-producers (or 'morphogens' as Turing called them) they may be adequate to determine the formation and growth of an organism, if they result in localized accumulations of form-producing substances. According to Turing the laws of physical chemistry are sufficient to account for many of the facts of morphogenesis (a view similar to that expressed by D'Arcy Thompson in *Growth and form*).

Turing arrived at differential equations of the form

$$\frac{\partial X_i}{\partial t} = f_i(X_1 \ldots, X_n) + \mu \nabla^2 X_i, \quad (i = 1, \ldots, n)$$

for n different morphogens in continuous tissue; where f_i is the reaction function giving the rate of growth of X_i, and $\nabla^2 X_i$ is the rate of diffusion of X_i. He also considered the corresponding equations for a set of discrete ceils. The function f_i involves the concentrations, and in his 1952 paper Turing considered the X_i's as variations from a homogeneous equilibrium. If, then, there are only small departures from equilibrium, it is permissible to linearize the f_i's, and so linearize the differential equations. In this way he was able to arrive at the conditions governing the onset of instability. Assuming initially a state of homogeneous equilibrium disturbed by random disturbances at $t = 0$, he discussed the various forms instability could take, on a continuous ring of tissue. Of the forms discussed the most important was that which eventually reached a pattern of stationary waves. The botanical situation corresponding to this would be an accumulation of the relevant morphogen in several evenly distributed regions around the ring, and would result in the main growth taking place at these points. (The examples cited are the tentacles *Hydra* and whorled leaves.) He also tested the theory by obtaining numerical solutions of the equations, using the electronic computer at Manchester. In the numerical example, in which two morphogens were supposed to be present in a ring of twenty cells, he found that a three or four lobed pattern would result. In other examples he found two-dimensional patterns, suggestive of dappling; and a system on a sphere gave results indicative of gastrulation. He also suggested that stationary waves in two dimensions could account for the phenomena of phyllotaxis.

In his later work (as yet unpublished) he considered quadratic terms in the reaction functions in order to take account of larger departures from the state of homogeneous equilibrium. He was

attempting to solve the equations in two dimensions on the computer at the time of his death. The work is in existence, but unfortunately is in a form that makes it extremely difficult to discover the results he obtained. However, B. Richards, using the same equations, investigated the problem in the case where the organism forms a spherical shell and also obtained numerical results on the computer. These were compared with the structure of *Radiolaria*, which have spikes on a basic spherical shell, and the agreement was strikingly good. The rest of this part of Turing's work is incomplete, and little else can be obtained from it. However, from Richards's results it seems that consideration of quadratic terms is sufficient to determine practical solutions, whereas linear terms are really only sufficient to discuss the onset of instability.

The second part of the work is a mathematical discussion of the geometry of phyllotaxis (i.e. of mature botanical structures). Turing discussed many ways of classifying phyllotaxis patterns and suggested various parameters by which a phyllotactic lattice may be described. In particular, he showed that if a phyllotactic system is Fibonacci in character, it will change, if at all, to a system which has also Fibonacci character. This is in accordance with observation. However, most of this section was intended merely as a description preparatory to his morphogenetic theory, to account for the facts of phyllotaxis; and it is clear that Turing did not intend it to stand alone.

The wide range of Turing's work and interests have made the writer of this notice more than ordinarily dependent on the help of others. Among many who have given valuable information I wish to thank particularly Mr R. Gandy, Mr J. H. Wilkinson, Dr B. Richards and Dr N. E. Hoskin; and Mrs Turing, Alan Turing's mother, for constant help with biographical material.

BIBLIOGRAPHY

1935*a*. On the Gaussian Error Function (King's College Fellowship Dissertation)
1935*b*. Equivalence of left and right almost periodicity. J. Lond. Math. Soc. **10**, 284.
1937*a*. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. (2), **42**, 230.
1937*b*. Computability and λ-definability. J. Symbolic Logic, **2**, 153.
1937*c*. The *p*-function in λ-K-conversion. J. Symbolic Logic, **2**, 164.
1937*d*. Correction to 1937*a*. Proc. Lond. Math. Soc. (2), **43**, 544.
1938*a*. Finite approximations to Lie groups. Ann. Math. Princeton, **39**, 105.
1938*b*. The extensions of a group. Comp. Math. **5**, 357.
1939. Systems of logic based on ordinals. Proc. Lond. Math. Soc. (2), **45**, 161.
1942*a*. (With M. H. A. Newman) A formal theorem in Church's theory of types. J. Symbolic Logic, **7**, 28.
1942*b*. The use of dots as brackets in Church's system, J. Symbolic Logic, **7**, 146.
1943*. A method for the calculation of the zeta-function. Proc. Lond. Math. Soc. (2), **48**, 180.
1948*a*. Rounding-off errors in matrix processes. Quart. J. Mech. App. Math. **1**, 287.
1948*b*. Practical forms of type-theory. J. Symbolic Logic, **13**, 80.
1950*a*. The word problem in semi-groups with cancellation. Ann. Math. **52**, 491.
1950*b*. Computing machinery and intelligence. Mind **59**, 433.
1950*c*. Programmers' Handbook for the Manchester electronic computer
1952. The chemical basis of morphogenesis. Phil. Trans. B **237**, 37.
1953*a*. Some calculations of the Riemann zeta-function. Proc. Lond. Math. Soc. (3), **3**, 99.
1953*b*. Digital computers applied to games: chess, pp. 288–295 of Faster than thought, ed. B. V. Bowden. Pitman, London.
1954. Solvable and unsolvable problems. Sci. News **31**, 7.

[A second paper on morphogenesis is being prepared for publication by N. E. Hoskin and B. Richards, based on work left by Turing.]

* Received four years earlier (7 March 1939).

On Computable Numbers, with an Application to the Entscheidungsproblem

(Proc. Lond. Math. Soc., ser. 2 vol. **42** (1936–37), pp. 230–265)

– A Correction

(ibid. vol. **43** (1937), pp. 544–546)

Christos Papadimitriou on —

ALAN AND I

During my sad college years, I often dreamed of Alan. At the time I did not know it was Alan Turing that I was dreaming of, but it was. I was studying a subject that did not excite me (electrical engineering), in the inflexible educational system of an oppressive society (Greece of the colonels). I had no access to a proper scientific library. My life as a fledgling scientist was one of frustration, blind longing, and episodes of false epiphany. A few subjects (systems theory, communication theory), even though they were taught at school in the most mundane way, enabled one to imagine a courageous intellectual universe in which questions of the most fundamental nature are confronted rigorously and head on, and I was aching to enter that universe.

I don't remember when, in which outdated textbook, handed to me by whom, I got my first glimpse of the Turing machine. I did not get it all at once, but I knew immediately that this abstract device is an important exemplar of the higher sphere I had been dreaming. I looked up in the dictionary the verb 'to ture' (I really did). I sought more information, every book I opened those days I opened it on the index page where 'Turing machine' should be.

Eventually I did put it all together, how a British mathematician named Alan Turing answered through his machine the world's most fundamental question, 'what can be computed?' and did so with amazing rigor, elegance, imagination and economy. But those days I was thinking of the Turing machine as a singular breakthrough, the end of a story, something of the past. Two years later, in 1973 – after a year in the Greek army that was even bleaker than my tertiary studies – I was fortunate to find myself at Princeton (Turing's Princeton, by the way, where I lived for a year in room 2B of the Graduate College rumored to be Alan's room 35 years before). At Princeton I was introduced to the Theory of Computation, the rich and vibrant scientific tradition essentially built on Turing's formalism. I remember how grateful I felt that my prayers had been answered, so to speak, and I was finally entering the realm of my dreams, far more elegant and exciting than I could ever imagine. And 5 years later, whilst teaching at Harvard with my friend Harry Lewis, we wrote a textbook on the subject.

But even though my life now revolved around his intellectual heritage, the truth is, I did not know Alan. I understood next to nothing about the man's life, personality, and breadth of achievement. In 1983, 2 years after our textbook was published, I read one of the books that have influenced me most: Turing's powerful and definitive biography by Andrew Hodges. Alan became my hero, a giant and relentless intellect, a fascinating and complex personality, a man of immense accomplishment, impact, and tragedy. When, more than a decade later, the second edition of our textbook

Alan Turing: His Work and Impact. DOI: 10.1016/B978-0-12-386980-7.00002-3

was published, Harry and I decided that we must have Alan Turing's image on the cover. Since the mid-1980s, every time I teach about Turing machines and undecidability, I stop to tell the class about Alan, about his ingenious work and about his tragic end. Once in a while I teach a course at Berkeley on 'Reading the Classics', and in it we spend a month on Turing, because I believe that every graduate student should be exposed directly to the exacting self-conscious greatness of Alan's opus.

There is a scene in Gibson's *Neuromancer*, at the very end of Part Three, where the hero returns to his hotel room to find it full of cops. *'Turing',* they tell him. *'You are under arrest'*. They mean 'Turing police', a fictional force bent on rooting out AI from the planet, but it does not matter, for me this line, read literally, contained the germ of an idea: What if Alan Turing were alive and turned up some place, unbidden? This strange fantasy lived inside me for a few years.

Turing is not my only intellectual hero. In poetry, my hero is Constantine Cavafy, a Greek from Alexandria who died when Alan Turing was 21. He wrote some of the greatest poems of the twentieth century, a stunning opus sharply divided between subtle historical metaphor and rather unsubtle eroticism. (I do not know why both my heroes happen to be homosexuals.) In 1997 I was in Greece, and I went to see a film titled *Cavafy*. I liked it so-so, but I remember coming out of the theatre impressed by the director's gesture: to honor one's hero by creating a work of art bearing his name. And then, right there in the theatre lobby, I had a vision: there was a blue paperback hovering above, and the title in front read *Turing (A Novel)*.

Writing a novel had never occurred to me before. I had never written short stories or poetry. I had of course noticed over the years that writing was not my weakest point, and neither was it for me the hated chore that comes after research, as it was for many of my colleagues. That night I thought about a plot.

This was 1997, when it was slowly becoming apparent to many computer scientists that the true object of our science is not the computer, but the Internet (by which I mean both the network of networks and the World Wide Web). In a sense, the Internet is the ultimate legacy of Alan Turing. The reason it spread like wildfire ever since a physicist named Tim Burners-Lee invented 'click' in 1989 is because there were millions of computers on the desks of people at that time, and these computers were all universal and so, in addition to everything else, they could be easily made to click. But universality was a minority opinion among computer dreamers during the 1930s and 1940s. By making his universal machine so compelling, Turing influenced deeply von Neumann and the way computers turned out to be. Universality and software would have probably taken root at some point in computers no matter what, but we can only speculate about the setbacks and delays this would have required without Turing.

But why did Turing envision universality? The reason is, he did not set out just to answer the question 'What can (and what cannot) be computed?' per se, as I inaccurately mentioned above. If he had only wanted to establish the existence of unsolvable problems, Turing could just use diagonalisation or growth, and the universal Turing machine might never come to light. Fortunately, he wanted to do something more ambitious and specific – and central to the scientific agenda of the time – he wanted to show that the *Entscheidungsproblem* (the decision problem for sentences not of arithmetic, but even of first-order logic) cannot be solved by computers. In other words, his goal was to sharpen Gödel's negative result, to extinguish the last glimmer of hope left by the Incompleteness Theorem. And to accomplish this he needed the universal Turing machine.

The night after I saw that movie, all this was in my mind. If, through this long chain of logic, the Internet is Alan's ultimate creation, why would not the Internet return the favor, and bring its creator back to life? The thought did not let me sleep. The Internet does confer a lame kind of immortality (just search for 'Alan Turing' on the web). Imagining a more explicit form, an Internet spirit residing somewhere and everywhere, a kind of impromptu, hacked-up SETI, was only a

modest step forward. And if I were to bring my hero back to life, why wouldn't I load him with gifts of gratitude, especially focusing on things he missed in life? I could give him, for example, a happy love life – after all obstacles are overcome, of course – make him a gifted teacher, give him a faithful pupil who would be a Greek man my age, yes, an archeologist perhaps, pining for a lost love, for an American woman, a software wiz maybe, why not? When I was a child, I happened to visit the island of Corfu with my father during the same summer Alan Turing was there. As the night advanced, the plot thickened. In the morning I recounted the story to my wife Martha, as I always do when I want to rid myself of an idea – because she can be a pitiless critic – but she went extra soft on it, she actually liked it. Two days later I was flying to California, and during that flight I drafted the first chapter – taking place, as it happens, in an airplane. For the next two and a half years I wrote every day, usually the first hours of my day, until the book was finished.

This book was a watershed. Whilst writing it, I understood things about myself that surprised me utterly. One of them was, I would be writing fiction again. Once more, Alan Turing had changed my life.

Alan inspires my papers and my stories, he fires my talks and my courses, inhabits my memories and my dreams. And because he's so intimate, impossible to examine anew and from a distance in order to discern something fresh, I could only speak here of this intimacy.

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[1]. These results have valuable applications. In particular, it is shown (§11) that the Hilbertian Entseheidungsproblem can have no solution.

In a recent paper Alonzo Church[2] 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 Entseheidungsproblem[3]. 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.

[1] Gödel, "Über formal unentscheidbaro Sätze der Principia Mathematica und ver-wandter Systeme, I", *Monatshefte Math. Phys.,* 38 (1931), 173–198.

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

[3] Alonzo Church "A note on the Entseheidungsproblem", *J. of Symbolic Logic*, 1 (1936), 40–41.

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 $q_1, q_2, \ldots, q_R$, 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 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

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 "𝔟", "𝔠", "𝔢" "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 𝔟 with a blank tape.

Configuration		*Behaviour*	
m-config.	*symbol*	*operations*	*final m-config.*
𝔟	None	$P0, R$	𝔠
𝔠	None	R	𝔢
𝔢	None	$P1, R$	𝔢
𝔢	None	R	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 00101101110111011111…. 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 "ɘɘ0"; 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.*
$\mathfrak{b}$		$P\mathfrak{d}, R, P\mathfrak{d}, R, P0, R, R, P0, L, L$	$\mathfrak{o}$
$\mathfrak{o}$	1	R, Px, L, L, L	$\mathfrak{o}$
	0		$\mathfrak{q}$
$\mathfrak{q}$	Any (0 or 1)	R, R	$\mathfrak{q}$
	None	$P1, L$	$\mathfrak{p}$
$\mathfrak{p}$	x	E, R	$\mathfrak{q}$
	$\mathfrak{d}$	R	$\mathfrak{f}$
	None	L, L	$\mathfrak{p}$
$\mathfrak{f}$	Any	R, R	$\mathfrak{f}$
	None	$P0, L, L$	$\mathfrak{o}$

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 :
$\mathfrak{b}$ $\mathfrak{o}$ $\mathfrak{q}$ $\mathfrak{q}$ $\mathfrak{q}$ $\mathfrak{p}$
ə ə 0 0 1 : ə ə 0 0 1 : ə ə 0 0 1 : ə ə 0 0 1 :
$\mathfrak{p}$ $\mathfrak{p}$ $\mathfrak{f}$ $\mathfrak{f}$
ə ə 0 0 1 : ə ə 0 0 1 : ə ə 0 0 1 0 :
$\mathfrak{f}$ $\mathfrak{f}$ $\mathfrak{o}$
ə ə 0 0 1 x 0 : ...
$\mathfrak{o}$

This table could also be written in the form

$$\mathfrak{b}: \mathfrak{d}\ \mathfrak{d}\ \mathfrak{o}\ 0 \qquad 0: \mathfrak{d}\ \mathfrak{d}\ \mathfrak{q}\ 0 \qquad 0: \ldots, \tag{C}$$

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. 1 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 a 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 a will be called marking β (or S) with a.

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 a 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 a 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 less explicitly : they may include expressions such as $\mathfrak{p}(e, x)$; 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 $\mathfrak{E}$ in $\mathfrak{p}(\mathfrak{E})$. Then it would have $\mathfrak{q}, \mathfrak{p}(\mathfrak{q}), \mathfrak{p}(\mathfrak{p}(\mathfrak{q})), \mathfrak{p}(\mathfrak{p}(\mathfrak{p}(\mathfrak{q}))), \ldots$ 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 "→" is used to signify "the machine goes into the m-configuration….")

$\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$	$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha)\mathfrak{B}, \alpha)$		From $\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$ the first α is erased
$\mathfrak{c}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$ E	$\mathfrak{C}$		and → $\mathfrak{B}$. If there is no $\alpha \rightarrow \mathfrak{B}$.
$\mathfrak{c}(\mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{c}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$		From $\mathfrak{e}(\mathfrak{B}, \alpha)$ all letters α are erased and → $\mathfrak{B}$.

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

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

Or, in greater detail:

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

$\mathfrak{pc}(\mathfrak{C}, \beta)$			$\mathfrak{f}(\mathfrak{pc}_1(\mathfrak{C}, \beta), \mathfrak{C}, \mathfrak{d})$	From $\mathfrak{pe}(\mathfrak{C}, \beta)$ the machine
$\mathfrak{pe}_1(\mathfrak{C}, \beta)$	Any	R, R	$\mathfrak{pc}_1(\mathfrak{C}, \beta)$	prints β at the end of the
	None	$P\beta$	$\mathfrak{C}$	sequence of symbols and → $\mathfrak{C}$.
$\mathfrak{l}(\mathfrak{C})$	L		$\mathfrak{C}$	From $\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$ it does the same
$\mathfrak{r}(\mathfrak{C})$	R		$\mathfrak{C}$	as for $\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$ but moves to the left before → $\mathfrak{C}$.
$\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{f}(\mathfrak{l}(\mathfrak{C}), \mathfrak{B}, \alpha)$	
$\mathfrak{f}''(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{f}(\mathfrak{r}(\mathfrak{C}), \mathfrak{B}, \alpha)$	
$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{f}'(\mathfrak{c}_1(\mathfrak{C}), \mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$. The machine writes
$\mathfrak{c}_1(\mathfrak{C})$	β		$\mathfrak{pe}(\mathfrak{C}, \mathfrak{B})$	at the end the first symbol marked α and → $\mathfrak{C}$.

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.

$\mathfrak{ce}(\mathfrak{C},\mathfrak{B},\alpha)$ $\mathfrak{ce}(\mathfrak{B},\alpha)$	$\mathfrak{c}(\mathfrak{e}(\mathfrak{C},\mathfrak{B},\alpha),\mathfrak{B},\alpha)$ $\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B},\alpha),\mathfrak{B},\alpha)$	$\mathfrak{ce}(\mathfrak{B},\alpha)$. The machine copies down in order at the end all symbols marked α and erases the letters α; $\to \mathfrak{B}$.
$\mathfrak{re}(\mathfrak{C},\mathfrak{B},\alpha,\beta)$ $\mathfrak{re}_1(\mathfrak{C},\mathfrak{B},\alpha,\beta)\ \ E,P\beta$ $\mathfrak{re}(\mathfrak{B},\alpha,\beta)$	$\mathfrak{f}_1(\mathfrak{re}_1(\mathfrak{C},\mathfrak{B},\alpha,\beta)\mathfrak{B},\alpha)$ $\mathfrak{C}$ $\mathfrak{re}(\mathfrak{re}(\mathfrak{B},\alpha,\beta),\mathfrak{B},\alpha,\beta)$	$\mathfrak{re}(\mathfrak{C},\mathfrak{B},\alpha,\beta)$. The machine replaces the first α by β and $\to \mathfrak{C} \to \mathfrak{B}$ if there is no α. $\mathfrak{re}(\mathfrak{B},\alpha,\beta)$. The machine replaces all letters α by β; $\to \mathfrak{B}$
$\mathfrak{cr}(\mathfrak{C},\mathfrak{B},\alpha)$ $\mathfrak{cr}(\mathfrak{B},\alpha)$	$\mathfrak{c}(\mathfrak{re}(\mathfrak{C},\mathfrak{B},\alpha,\alpha),\mathfrak{B},\alpha)$ $\mathfrak{cr}(\mathfrak{cr}(\mathfrak{B},\alpha),\mathfrak{re}(\mathfrak{B},\alpha,\alpha),\alpha)$	$\mathfrak{cr}(\mathfrak{B},\alpha)$ differs from $\mathfrak{ce}(\mathfrak{B},\alpha)$ only in that the letters a are not erased. The *m*-configuration $\mathfrak{cr}(\mathfrak{C},\alpha)$ is taken up when no letters "α" are on the tape.

$$\mathfrak{cp}(\mathfrak{C},\mathfrak{A},\mathfrak{E},\alpha,\beta) \qquad \mathfrak{f}'(\mathfrak{cp}_1(\mathfrak{C}_1,\mathfrak{A},\beta),\mathfrak{f}(\mathfrak{A},\mathfrak{E},\beta),\alpha)$$

$$\mathfrak{cp}_1(\mathfrak{C},\mathfrak{A},\beta) \qquad \gamma \qquad \mathfrak{f}'(\mathfrak{cp}_2(\mathfrak{C},\mathfrak{A},\gamma),\mathfrak{A},\alpha)$$

$$\mathfrak{cp}_2(\mathfrak{C},\mathfrak{A},\gamma) \qquad \begin{cases} \gamma & \mathfrak{C} \\ \text{not } \gamma & \mathfrak{A}. \end{cases}$$

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

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

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

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

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

$\mathfrak{q}(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$	$\mathfrak{q}(\mathfrak{C},\alpha)$. The machine finds the last symbol of form α. $\to \mathfrak{C}$.
	None	R	$\mathfrak{q}_1(\mathfrak{C})$	
$\mathfrak{q}_1(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$	
	None		$\mathfrak{C}$	
$\mathfrak{q}(\mathfrak{C},\alpha)$			$\mathfrak{q}(\mathfrak{q}(\mathfrak{C},\alpha)$	
$\mathfrak{q}_1(\mathfrak{C},\alpha)$	α		$\mathfrak{C}$	
	not α	L	$\mathfrak{q}_1(\mathfrak{C},\alpha)$	
$\mathfrak{pe}_2(\mathfrak{C},\alpha,\beta)$			$\mathfrak{pe}(\mathfrak{pe}(\mathfrak{C},\beta),\alpha)$	$\mathfrak{pe}_2(\mathfrak{C},\alpha,\beta)$. The machine prints $\alpha\beta$ at the end

$\mathfrak{ce}_2(\mathfrak{B},\alpha,\beta)$		$\mathfrak{ce}(\mathfrak{cc}(\mathfrak{B},\beta),\alpha)$	$\mathfrak{ce}_3(\mathfrak{B},\alpha,\beta,\gamma)$. 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.
$\mathfrak{ce}_3(\mathfrak{B},\alpha,\beta,\gamma)$		$\mathfrak{ce}(\mathfrak{cc}_2(\mathfrak{B},\beta,\gamma),\alpha)$	
$\mathfrak{e}(\mathfrak{C})$	$\begin{cases}\ni & R\\ \text{Not } \ni & L\end{cases}$	$\begin{matrix}\mathfrak{c}_1(\mathfrak{C})\\ \mathfrak{c}(\mathfrak{C})\end{matrix}$	From $\mathfrak{c}(\mathfrak{C})$ the marks are erased form all marked symbols $\to \mathfrak{C}$.
$\mathfrak{e}_1(\mathfrak{C})$	$\begin{cases}\text{Any} & R,E,R\\ \text{None} & \end{cases}$	$\begin{matrix}\mathfrak{e}_1(\mathfrak{C})\\ \mathfrak{C}\end{matrix}$	

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. 19, 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. 18. That is to say, that the entry in the operations column is always of one of the forms $E:E$, $R:E$, $L:P\alpha;P\alpha$, $R:P\alpha$, $L:R:L$: 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	S_i	E,R	q_m

are to be written as

q_i	S_j	PS_0, R	q_m

and lines such as

q_i	S_j	R	q_m

to be written as

q_i	S_j	PS_j, R	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 $q_iS_jS_kLq_m$; from each line of form (N_2) we form an expression $q_iS_jS_kRq_m$; and from each line of form (N_3) we form an expression $q_iS_jS_kNq_m$.

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^3"

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	S_1	PS_1, R	q_2

Our first standard form would be

$$q_1S_0S_1Rq_2;\ q_2S_0S_0Rq_3;\ q_3S_0S_2Rq_4;\ q_4S_0S_0Rq_1;.$$

The standard description is

$$DADDCRDAA;\ DAADDRDAAA;\ DAAADDCCRDAAAA;\ DAAAADDRDA;$$

A description number is

$$3133253117311335311173111332253111173111133531 7$$

and so is

$$313325311731133531117311133225311117311113353173132325311 7$$

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 $\mathcal{U}$ 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 $\mathcal{U}$.

Let us first suppose that we have a machine $\mathcal{M}'$ which will write down on the *F*-squares the successive complete configurations of $\mathcal{M}$. These might be expressed in the same form as on p. 19 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 "ɒ" by "*DAA*", "ə" by "*DCCC*", "ɋ" by "*DAAA*", then the sequence (C) becomes:

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

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

It is not difficult to see that if $\mathcal{M}$ can be constructed, then so can $\mathcal{M}'$. The manner of operation of $\mathcal{M}'$ could be made to depend on having the rules of operation (*i.e.*, the S.D) of $\mathcal{M}$. written somewhere within itself (*i.e.* within $\mathcal{M}'$); 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 $\mathcal{M}'$ 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

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

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. 22)

$$\mathfrak{e}(\mathfrak{anf}) \qquad \begin{cases} \mathfrak{ə} & R & \mathfrak{e}_1(\mathfrak{anf}) \\ \text{not } \mathfrak{ə} & L & \mathfrak{e}(\mathfrak{anf}) \end{cases}$$

$$\mathfrak{e}_1(\mathfrak{anf}) \qquad \begin{cases} \text{Any} & R,E,R & \mathfrak{e}_1(\mathfrak{anf}) \\ \text{None} & & (\mathfrak{anf}) \end{cases}$$

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

When $\mathcal{U}$ is ready to start work the tape running through it bears on it the symbol a on an *F*-square and again a 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 $\mathcal{U}$ 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_α,R	$\mathfrak{con}_1(\mathfrak{C},\alpha)$	$\mathfrak{con}(\mathfrak{C},\alpha)$ 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. $\rightarrow \mathfrak{C}$.
$\mathfrak{con}_1(\mathfrak{C},\alpha)$	A R,P_α,R	$\mathfrak{con}_1(\mathfrak{C},\alpha)$	
	D R,P_α,R	$\mathfrak{con}_2(\mathfrak{C},\alpha)$	
$\mathfrak{con}_2(\mathfrak{C},\alpha)$	C R,P_α,R	$\mathfrak{con}_2(\mathfrak{C},\alpha)$ $\mathfrak{C}$	$\mathfrak{con}(\mathfrak{C},\alpha)$. 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.
	Not C R,R		

The table for $\mathcal{U}$.

$\mathfrak{b}$		$\mathfrak{f}(\mathfrak{b}_1,\mathfrak{b}_2,::,\alpha)$	$\mathfrak{b}$. The machine prints: *DA* on the F-squares after :: $\rightarrow \mathfrak{anf}$.
$\mathfrak{b}_1$	$R,R,P:,R,R,$	$\mathfrak{anf}$	
$\mathfrak{anf}$	PD,R,R,PA	$\mathfrak{g}(\mathfrak{anf}_1,:)$	$\mathfrak{anf}$. The machine marks the configuration in the last complete configuration with y. $\rightarrow \mathfrak{fom}$.
$\mathfrak{anf}_1$	$\mathfrak{con}(\mathfrak{fom},y)$		
$\mathfrak{fom}$	; R,P_z,L	$\mathfrak{con}(\mathfrak{fmp},x)$	$\mathfrak{fom}$. 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.
	z L,L	$\mathfrak{fom}$	
	not z nor; L	$\mathfrak{fom}$	
$\mathfrak{fmp}$	$\mathfrak{cpe}(e(\mathfrak{fom},x,y),$ $\mathfrak{sim},x,y)$		$\mathfrak{fmp}$. 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, P_u, R, R, R	$\mathfrak{sim}_2$
$\mathfrak{sim}_3$	not A L, P_y	$\mathfrak{e}(\mathfrak{mf}, z)$
	A L, P_y, R, R, R	$\mathfrak{sim}_3$
$\mathfrak{mf}$		$\mathfrak{g}(\mathfrak{mf}, :)$
$\mathfrak{mf}_1$	not A R, R	$\mathfrak{mf}_1$
	A L, L, L, L	$\mathfrak{mf}_2$
$\mathfrak{mf}_2$	C R, P_x, L, L, L	$\mathfrak{mf}_2$
	:	$\mathfrak{mf}_4$
	D R, P_x, L, L, L	$\mathfrak{mf}_3$
$\mathfrak{mf}_3$	not : R, P_v, L, L, L	$\mathfrak{mf}_3$
	:	$\mathfrak{mf}_4$
$\mathfrak{mf}_4$		$\mathfrak{con}(\mathfrak{l}(\mathfrak{l}(\mathfrak{mp}_5)),)$
$\mathfrak{mf}_5$	Any R, P_w, R	$\mathfrak{mf}_5$
	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.

$\mathfrak{mf}$. 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.

$\mathfrak{inst}$			$\mathfrak{g}(\mathfrak{l}(\mathfrak{inst}_1), u)$	
$\mathfrak{inst}_1$	α	R, E	$\mathfrak{inst}_1(\alpha)$	$\mathfrak{inst}$ The next complete configuration is written down, carrying out the marked instructions. The letters u, v, w, x, y are erased. $\rightarrow \mathfrak{anf}$.
$\mathfrak{inst}_1(L)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, y, x, u, w)$	
$\mathfrak{inst}_1(R)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, y, x, u, w)$	
$\mathfrak{inst}_1(N)$			$\mathfrak{ec}_5(\mathfrak{ov}, v, y, x, u, w)$	
$\mathfrak{ov}$			$\mathfrak{e}(\mathfrak{anf})$	

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[4]. 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 a_n be the n-th computable sequence, and let $\phi_n(m)$ be the m-th figure in a_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 $\phi_n(n)$.

Let us suppose that there is such a process; that is to say, that we can invent a machine $\mathcal{D}$ 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 $\mathcal{D}$ and $\mathcal{U}$ we could construct a machine $\mathcal{H}$ I to compute the sequence β'. The machine $\mathcal{D}$, 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 $\mathcal{D}$ is erased.

The machine $\mathcal{H}$ has its motion divided into sections. In the first $N-1$ sections, among other things, the integers 1, 2,...,$N-1$ have been written down and tested by the machine $\mathcal{D}$. 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 $\mathcal{D}$ 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

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

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 $\mathcal{H}$. 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 $\mathcal{H}$ we can see that $\mathcal{H}$. is circle-free. Each section of the motion of $\mathcal{H}$ comes to an end after a finite number of steps. For, by our assumption about $\mathcal{D}$ 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 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 $\mathcal{H}$ is circle-free.

Now let K be the D.N of $\mathcal{H}$. What does.1 $\mathcal{H}$ 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 $\mathcal{H}$ and since $\mathcal{H}$ 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 $\mathcal{H}$, 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 $\mathcal{H}$. 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.*, $\mathcal{H}$ 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 $\mathcal{D}$.

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 $\overline{0}$. $\mathcal{M}_2$ is to have the first two symbols 0 replaced by $\overline{0}$ and so on. Thus, if. $\mathcal{M}$. were to print

$$ABA01AAB0010AB\ldots,$$

then $\mathcal{M}_1$ would print

$$ABA\overline{0}1AAB0010AB\ldots$$

and. $\mathcal{M}_2$ would print

$$ABA\overline{0}1AAB\overline{0}010AB\ldots.$$

Now let $\mathcal{R}$ 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 $\mathcal{R}$ with $\mathcal{E}$. and obtain a new machine, }. In the motion of $\mathcal{R}$ first $\mathcal{R}$ 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 $\mathcal{R}$ 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 } with $\mathcal{E}$. If it is found that }, never prints 0, then $\mathcal{M}$ prints 0 infinitely often; if } 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.

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[5]. 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 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.

[5] 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 Bet 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.

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 reognisable. 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) CChanges 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. 30, 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[6] 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[7] machine $\mathcal{K}$, which will find all the provable formulae of the calculus[8].

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[9] $G_\alpha(x)$ 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.,

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

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.

$$\mathfrak{A}\ \&\ F^{(n)} \rightarrow G_\alpha(u^{(n)}), \qquad (\mathrm{A}_n)^{10}$$

$$\mathfrak{A}\ \&\ F^{(n)} \rightarrow \left(-G_\alpha(u^{(n)})\right), \qquad (\mathrm{B}_n),$$

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

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

We divide the motion of $\mathcal{K}_\alpha$ 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 $\mathcal{K}_\alpha$ then starts to do the work of $\mathcal{K}$, 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 $\mathcal{K}$ 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 $\mathcal{K}$. Hence the n-th section will eventually be finished. $\mathcal{K}_\alpha$. 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.

---

[6] The expression "the functional calculus" is used throughout to mean the *restricted* Hilbert functional calculus.

[7] 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 t h a t 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 $1, \ldots, i_n = 0$ or 1), and hence the number $2'' + i_1 2^{n-1} + i_2 2^{n-2} + \ldots + i_n$ completely determines the proof. The automatic machine carries out successively proof 1, proof 2, proof 3,....

[8] The author has found a description of such a machine.

[9] The negation sign is written before an expression and not over it.

[10] A sequence of r primes is denoted by $^{(r)}$.

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[7] 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',

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

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 γn be the number computed by $\mathcal{M}(n)$, and let

$$a_n = \tan\left(\pi(\gamma_n - \tfrac{1}{2})\right),$$

[7] 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[8]. Now let $\phi(n)$ be a computable function which can be shown to be such that for any satisfactory argument its value is satisfactory,[9]. Then the function f, defined by $f(a_n) = a_{\phi(n)}$, 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).

(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

$$\eta(0) = r,$$
$$\eta(n) = \phi(n,\ \eta(n-1)).$$

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

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

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

$$(b) \quad G(\alpha)\ \&\ (-G(\beta)) \rightarrow (\alpha < \beta),$$

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

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

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

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

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

[9] 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.

(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 $e = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \ldots$ 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

$$\begin{aligned}\mathfrak{A}_\phi \;\&\; P\&\; (F(x, y) \to G(x,y)) \;\&\; (G(x, y) \;\&\; G(y, z) \to G(x, z))\\ \&\left(F^{(r)} \to H(u, u^{(r)})\right) \;\&\; (F(v, w) \;\&\; H(v, x) \;\&\; K(w, x, z) \to H(w, z))\\ \&\; [H(w, z) \;\&\; G(z, t) \mathrm{v} G(t,z) \to (-H(w, t))]\,.\end{aligned}$$

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

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

then, for some M,

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

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

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

and

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

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

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

Hence for each n some formula of the form

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

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

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

and

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

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

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 $\mathcal{N}$, which, starting with a tape bearing on it ə ə followed by a sequence of any number of letters "F" on F-squares and in the m-configuration b, will compute a sequence γ_n depending on the number n 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 $\mathcal{N}$ 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, $\overline{0}$, and $\overline{1}$ do not occur in the table, and we replace ə throughout by Θ, 0 by $\overline{0}$, and 1 by $\overline{1}$. Further substitutions are then made. Any line of form

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

we replace by

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

and any line of the form

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

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 u and 1 for 0 together with the following line

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

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

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[10]. 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".

$I(x,\ y)$ 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 $\{q_iS_jS_kLq_l\}$ is to be an abbreviation for

$$\begin{aligned}(x,\ y,\ x',\ y')\big\{&(R_{S_j}(x,\ y)\ \&\ I(x,\ y)\ \&\ K_{q_i}(x)\ \&\ F(x,\ x')\ \&\ F(y',\ y))\\ &\to (I(x',y')\ \&\ R_{s_k}(x',\ y)\ \&\ K_{q_l}(x')\\ &\qquad \&\ (z)\ [F(y',\ z)\mathrm{v}(R_{S_j}(x,z) \to R_{S_k}(x',\ z))])\big\}.\end{aligned}$$

Inst $\{q_iS_jS_kRq_l\}$ and Inst $\{q_iS_jS_kNq_l\}$

are to be abbreviations for other similarly constructed expressions.

Let us put the description of $\mathcal{M}$ into the first standard form of §6. This description consists of a number of expressions such as "$q_iS_jS_kLq_l$" (or with R or N substituted for L). Let us form

[10] Loc. cir.

all the corresponding expressions such as Inst $\{q_iS_jS_kLq_l\}$ and take their logical sum. This we call Des $(\mathcal{M})$.

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

$$\begin{aligned}(\exists u)[N(u)\ \&\ (x)(N(x) \rightarrow (\exists x')F(x,\ x'))\\ \&\ (y,z)(F(y,\ z) \rightarrow N(y)\ \&\ N(z))\ \&\ (y)R_{S_0}(u,\ y)\\ \&\ I(u,\ u)\ \&\ K_{q_1}(u)\ \&\ Des(\mathcal{M})]\\ \rightarrow (\exists s)(\exists t)[N(s)\ \&\ N(t)\ \&\ R_{S_1}(s,\ t)].\end{aligned}$$

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

When we substitute the meanings suggested on p. 37–38 we find that Un $(\mathcal{M})$ has the interpretation "in some complete configuration of $\mathcal{M}$, 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 $\mathcal{M}$, then Un $(\mathcal{M})$ is provable.
(b) If Un $(\mathcal{M})$ is provable. then S_1 appears on the tape in some complete configuration of $\mathcal{M}$.

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 $\mathcal{M}$, then* Un *$(\mathcal{M})$ is provable.*

We have to show how to prove Un $(\mathcal{M})$. 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

$$\begin{aligned}R_{S_{r(n,0)}}(u^{(n)},\ u)\ \&\ R_{S_{r(n,1)}}(u^{(n)},\ u')\ \&\ \ldots\ \&\ R_{S_{r(n,n,}}(u^{(n)},\ u^{(n)})\\ \&\ I(u^n, u^{(i(n))})\ \&\ K_{qk_{(n)}}(u^{(n)})\\ \&\ (y)F((y,\ u')\text{v}F(u,\ y)\text{v}F(u',\ y)\text{v}\ \ldots\ \text{v}F(u^{(n-1)},\ y)\text{v}R_{S_0}(u^{(n)}, y)),\end{aligned}$$

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 $A(\mathcal{M})\ \&\ F^{(n)} \rightarrow CC_n$ (abbreviated to CF_n) are provable. The meaning of CF_n is "The n-th complete configuration of $\mathcal{M}$ is so and so", where "so and so" stands for the actual n-th complete configuration of $\mathcal{M}$. 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

$$(y)R_{S_0}(u,\ y)\ \&\ I(u,\ u)\ \&\ K_{q1}(u).$$

$A(\mathcal{M}) \rightarrow CC_0$ is then trivial.

We next show that $CF_n \rightarrow CF_{n+1}$ 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

$$r(n,\ i(n)) = a,\ r(n+1, i(n+1)) = c,\ k(i(n)) = b, \text{ and } k(i(n+1)) = d,$$

then Des $(\mathcal{M})$ must include Inst $\{q_aS_bS_dLq_c\}$ as one of its terms, *i.e.*

$$\text{Des}\ (\mathcal{M}) \rightarrow \text{Inst}\ \{q_aS_bS_dLq_c\}.$$

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

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

is provable, and so therefore is

$$A(\mathcal{M}) \;\&\; F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$$

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

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

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

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

This completes the proof of Lemma 1.

Lemma 2 *If* 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. 37–38 in 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 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 Un ($\mathcal{M}$) in a form in which all quantors are at the beginning. 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 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 $I + \phi_\gamma(\mu)$ is a λ-definable function of n, *i.e.* if there is a W.F.F. M_γ such that, for all integers n,

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

i.e. $\{M_\gamma\}(N_n)$ is convertible into $\lambda xy.\ x(x(y))$ 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 (of. foot-note p. 32). 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

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

and with

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

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,

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

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

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

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

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

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

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

Let W_γ stand for

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

so that, for each integer n

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

and let Q be a formula such that

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

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

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

it will have the required property[11]

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

[11] 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

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

where

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

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

etc.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM. A CORRECTION

By A. M. TURING

In a paper entitled "On computable numbers, with an application to the Entscheidungsproblem"[1] the author gave a proof of the insolubility of the Entscheidungsproblem of the "engere Funktionenkalkül". This proof contained some formal errors[2] which will be corrected here: there are also some other statements in the same paper which should be modified, although they are not actually false as they stand.

The expression for Inst $\{q_i S_j S_k L q_l\}$ on p. 37 of the paper quoted should read

$$\begin{aligned}(x,\, y,\, x',y')\{(R_{S_j}(x,\, y)\ \&\ I(x,\, y)\&K_{qi}(x)\ \&\ F(x,x')\ \&\ F(y',\, y))\\ \rightarrow (I(x',\, y')\ \&\ R_{s_k}(x',y)\ \&\ K_{ql}(x')\ \&\ F(y',z)\mathrm{v}[(R_{s_0}(x,z) \rightarrow R_{s_0}(x',z))\\ \&(R_{S_1}(x,\, z) \rightarrow R_{s_1}(x',z))\ \&\ \ldots\ \&\ (R_{s_M}(x,z) \rightarrow R_{s_M}(x',z))])\},\end{aligned}$$

$S_0,\ S_1, \ldots, S_M$ being the symbols which. $\mathcal{M}$ can print. The statement on p. 39, line 4, viz.

$$\text{"Inst } \{q_a S_b S_d L q_c\}\ \&\ F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$$

is provable" is false (even with the new expression for Inst $\{q_a S_b S_d L q_c\}$): we are unable for example to deduce $F^{(n+1)} \rightarrow \big(-F(u,\, u'')\big)$ and therefore can never use the term

$$F(y',\, z)\mathrm{v}[(R_{S_0}(x,\, z) \rightarrow R_{S_0}(x',\, z)\ \&\ \ldots\ \&\ (R_{S_M}(x,z) \rightarrow R_{S_M}(x',z))]$$

in Inst $\{q_a S_b S_d L q_c\}$. To correct this we introduce a new functional variable $G[G(x,y)$ to have the interpretation "x precedes y"]. Then, if Q is an abbreviation for

$$\begin{aligned}(x)(\exists w)(y,z)\{F(x,w)\ \&\ (F(x,y) \rightarrow G(x,y))\ \&\ (F(x,z)\ \&\ G(x,y) \rightarrow G(x,y))\\ \&\ [G(z,x)\mathrm{v}(G(x,y)\ \&\ F(y,z))\mathrm{v}(F(x,y)\ \&\ F(z,y)) \rightarrow (-F(x,z))]\end{aligned}$$

the corrected formula Un $(\mathcal{M})$ is to be

$$(\exists u)A(\mathcal{M}) \rightarrow (\exists s)(\exists t)R_{s_1}(s,t).$$

where $A(\mathcal{M})$ is an abbreviation for

$$Q\ \&\ (y)R_{S_0}(u,\, y)_1\ \&\ I(u,\, u)\ \&\ K_{q_1}(u)\ \&\ \mathrm{Des}(\mathcal{M}).$$

The statement on p. 39 (line 3) must then read

$$\text{Inst } \{q_a S_b S_d L q_c\}\ \&\ Q\ \&\ QF^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1}),$$

and line 29 should read

---

[1] Proc. London Math. Soc. (2), 42 (1936–7), 230–265.

[2] The author is indebted to P. Bernays for pointing out these errors.

$$r(n,i(n)) = b,\ r(n+1,i(n)) = d,\ k(n) = a,\ k(n+1) = c.$$

For the words "logical sum" on p. 38, line 4, read "conjunction". With these modifications the proof is correct. Un ($\mathcal{M}$) may be put in the form (I) (p. 39) with n = 4.

Some difficulty arises from the particular manner in which "computable number" was defined (p. 18). If the computable numbers are to satisfy intuitive requirements we should have:

If we can give a rule which associates with each positive integer n two rationals a_n, b_n satisfying $a_n \leqslant a_{n+1} < b_{n+1} \leqslant b_n, b_n - a_n < 2^{-n}$, then there is a computable number a for which $a_n \leqslant a \leqslant b_n$ each n. (A)

A proof of this may be given, valid by ordinary mathematical standards, but involving an application of the principle of excluded middle. On the other hand the following is false:

There is a rule. whereby, given the rule of formation of the sequences, a_n, b_n in (A) we can obtain a D.N. for a machine to compute a. (B)

That (B) is false, at least, if we adopt the convention that the decimals of numbers of the form $m/2^n$ shall always terminate with zeros, can be seen in this way. Let $\mathcal{N}$ be some machine, and define c_n as follows: $c_n = \frac{1}{2}$ if $\mathcal{N}$ has not printed a figure 0 by the time the n-th complete configuration is reached $c_n = \frac{1}{2} - 2^{m-3}$ if 0 had first been printed at the m-th, complete configuration ($m \leqslant n$). Put $a_n = c_n - 2^{-n-2}, b_n = c_n + 2^{-n-2}$. Then the inequalities of (A) are satisfied, and the first figure of α is 0 if $\mathcal{N}$ ever prints 0 and is 1 otherwise. If (B) were true we should have a means of finding the first figure of a given the D.N. of $\mathcal{N}$ *i.e.* we should be able to determine whether $\mathcal{N}$ ever prints 0, contrary to the results of §8 of the paper quoted. Thus although (A) shows that there must be machines which compute the Euler constant (for example) we cannot at present describe any such machine, for we do not yet know whether the Euler constant is of the form $m/2^n$.

This disagreeable situation can be avoided by modifying the manner in which computable numbers are associated with computable sequences, the totality of computable numbers being left unaltered. It may be done in many ways[3] of which this is an example. Suppose that the first figure of a computable sequence γ, is i and that this is followed by 1 repeated n times, then by 0 and finally by the sequence whose r-th figure is c_r; then the sequence γ, is to correspond to the real number

$$(2i-1)n + \sum_{r=1}^{\infty} (2c_r - 1)(\tfrac{2}{3})^r$$

If the machine which computes γ is regarded as computing also this real number then (B) holds. The uniqueness of representation of real numbers by sequences of figures is now lost, but this is of little theoretical importance, since the D.N.'s are not unique in any case.

The Graduate College,
Princetom, N.J., U.S.A.

[3] This use of overlapping intervals for the definition of real numbers is due originally to Brouwer.

Examining the Work and Its Later Impact

Stephen Wolfram on —

The Importance of Universal Computation

In the long view of intellectual history, I believe universal computation will stand as the single most important idea to emerge in the twentieth century. And this paper is where it first appeared with clarity.

The paper certainly did not set out to have such significance. Instead, its purpose was to address a technical question – albeit one thought to be important – about the foundations of mathematics. But to address that question, Turing created the concept of a universal computer – which in time led to the notion of software, the computer revolution, and an increasing fraction of all our technology today.

In retrospect, it seems almost bizarre that it took until 1936 for such a basic idea to emerge. For today, immersed as they are in modern technology, even quite young children seem to have a decent grasp of the basic idea of programmability and universal computation.

Even in antiquity, there was already the notion that any single human language could describe the same basic range of facts and processes. Leibniz tightened this up in the 1600s, imagining a universal language based on logic, and even discussing encoding logic with numbers (Leibniz, 1966).

Then in the 1800s, there were punched-card machines that could be programmed for different functions. And with the increasing abstraction and formalisation of mathematics, there emerged by the 1920s ideas like combinators and string-rewrite systems.

But it was Gödel's theorem (Gödel, 1931) that highlighted the importance of such abstractions. And in fact, inside the proof of Gödel's theorem is in effect exactly the idea of universal computation – but framed in the context of purely mathematical constructs.

The great significance of Turing's paper was to give concreteness to universal computation: to make it seem that universal computation might somehow be inevitable in any constructible system.

At the time, it was far from clear how general Turing's results might be. After all, for example, until the 1920s, there had been the idea that any reasonable mathematical function could be represented by primitive recursion – but that idea was immediately exploded by the discovery of the Ackermann function (Wilhelm, 1928).

And indeed, after his 1936 paper, Turing himself set about looking at systems involving oracles and so on, that would be more powerful than his universal Turing machine. But gradually an increasing collection of 'implementable' abstract models seemed instead to be precisely equivalent in their power to ordinary Turing machines.

Meanwhile, electronic computers were emerging. McCulloch and Pitts (1943) had used Turing's idea of a universal machine to argue that brains could in effect just be like networks of electronic components. And von Neumann (Burks, Goldstine and von Neumann, 1947) then applied these ideas to develop architectures for practical electronic computers.

There continued to be some theoretical work on Turing machines, but for the most part, electronic computers were treated purely as technology, with little discussion of foundational issues. And indeed, it was only in about the 1980s that Turing's work began to become more widely known.

And at that time, there tended to be the view that Turing machines were relevant as idealisations of what could be implemented with electronics, and perhaps with mathematics, but not necessarily much more. And indeed it was usually assumed (as it had been by Turing himself) that when it came to typical systems in nature, traditional mathematical equations – and not something like discrete Turing machines – would be the relevant models to use.

In the early 1980s, I became interested in a variety of natural systems that exhibited complex behaviour, and that had never been very usefully described by traditional mathematical equations. And I set about trying to find the simplest models that might describe such systems.

I had experience both with practical computers and with models in statistical physics based on discrete components. And in trying to find the simplest possible model, I quickly settled on one-dimensional cellular automata.

My main initial methodology for studying cellular automata was experimental: to just run computer experiments and see how the cellular automata behaved (see Figure 1).

The results were remarkable, and to me deeply surprising. For I found that even when the underlying rules for the system were extremely simple, the behaviour of the system as a whole could be immensely complex (see Figure 2).

And gradually, through my work (Wolfram, 2002) and the work of many others, it began to be clear that a great many systems in nature could successfully be modelled using these kinds of simple programs.

But as I searched for an understanding of the basic phenomenon by which complexity was generated, I was quickly led to Turing machines and universal computation. And I came to speculate that even in the simple cellular automata I was studying, there must be universal computation which in turn I then argued led to perceived complexity, and a variety of other fundamental phenomena.

Before my work, one might have assumed that systems in nature would typically need to be described by the standard continuous differential equations of mathematical physics – and would therefore presumably not act like Turing machines. But after seeing so many examples of natural systems successfully described by systems like cellular automata, it began to seem much more plausible that nothing with power beyond Turing machines was needed.

I do not think that Alan Turing ever directly simulated a Turing machine. He was interested in the theoretical issue of whether a Turing machine that is universal could be constructed. And indeed in this paper he showed that that was possible – though with a machine of considerable complexity. It was not until the beginning of the 1990s that I actually started simulating Turing machines in large numbers. I decided to see if my results on cellular automata would carry over to Turing machines – which operate in a sequential, rather than parallel, way.

And what I found was that much like in cellular automata, one does not have to go far in the universe of possible Turing machines before one starts to find examples that exhibit highly complex behaviour.

For a long time, it was not clear what the very simplest universal Turing machine might be. But now we know. It is a machine with two states and three colors that I first identified in the mid-1990s (see Figure 3), and that was finally proved universal in 2007 as a result of a competition we held (Smith, 2007).

Traditional intuition from looking at practical computers might have suggested that to get universality would require a system with a complicated structure, typical of what might be set up by human engineers. But from my studies in the computational universe of possible programs, I had

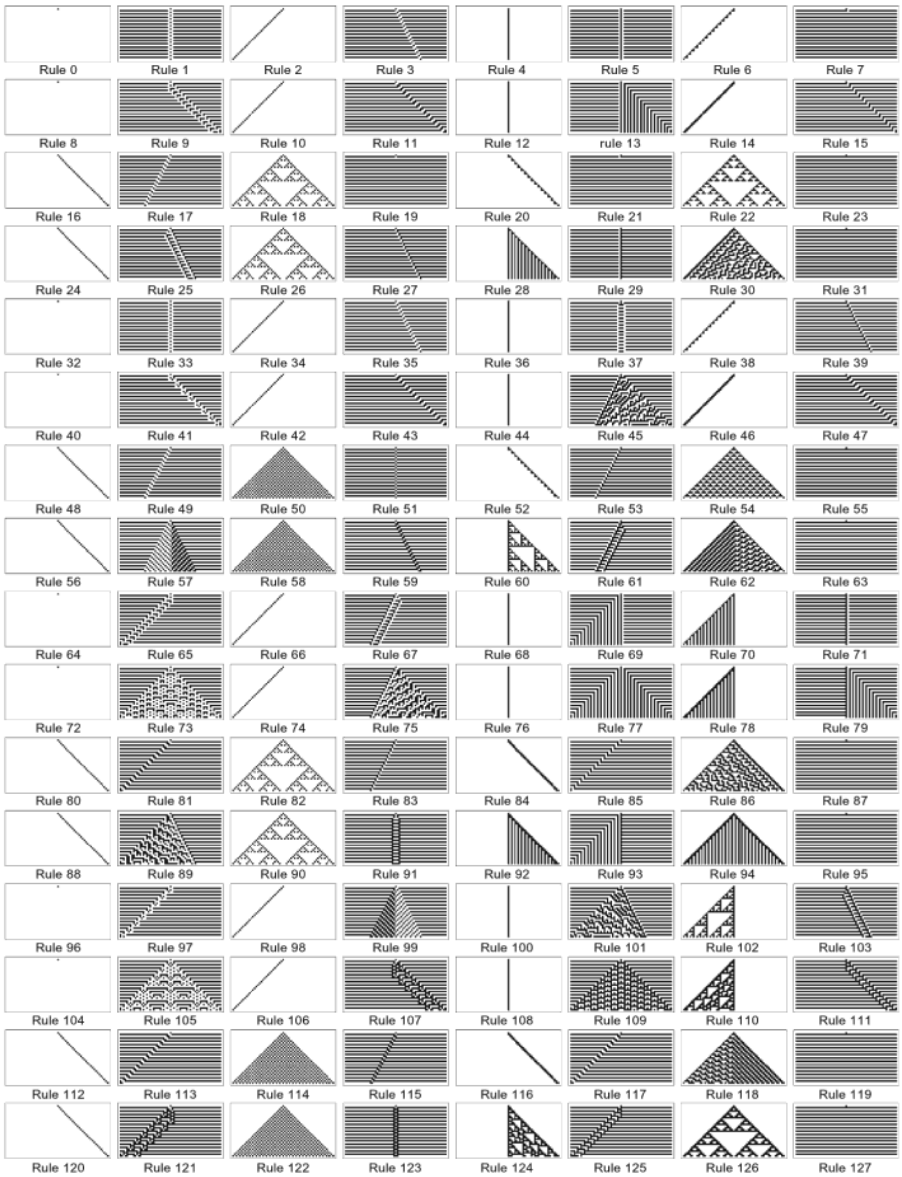

Fig. 1: Evolution of the first 128 elementary cellular automata, starting in all cases from a single black cell.

formulated a general principle that I called the Principle of Computational Equivalence, which predicted, among many other things, that universal computation should actually be very common, even among systems with simple underlying structures.

The result of this is to even further enhance the importance of Turing's paper.

At first, we might have thought that things like universal Turing machines would be relevant only to specific kinds of mathematical-type systems. But gradually we came to learn that all sorts of systems – including practical ones made with electronic components – could be set up to behave like universal Turing machines.

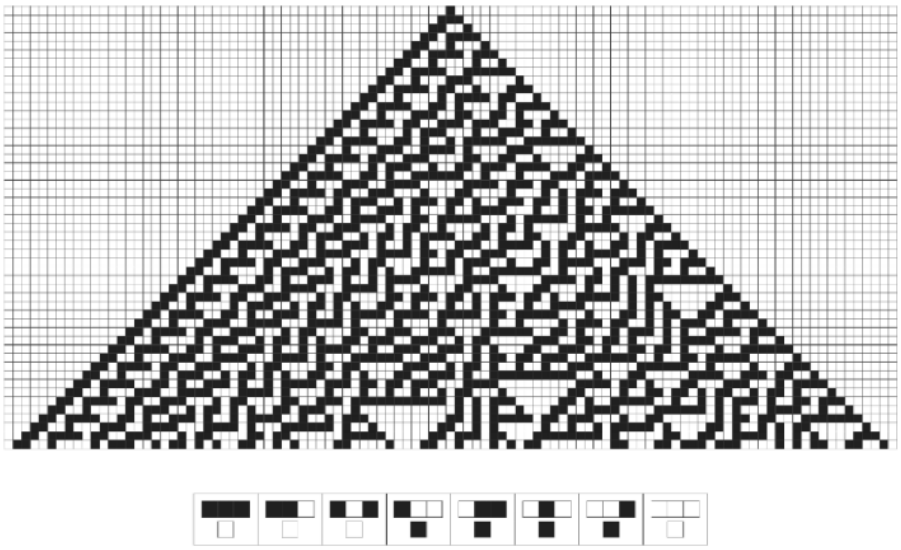

Fig. 2: A cellular automaton with a simple rule that generates a pattern, which seems in many respects random. The rule used is of the same type as in the previous examples, and the cellular automaton is again started from a single black cell. The pattern that is obtained is highly complex, and shows almost no overall regularity.

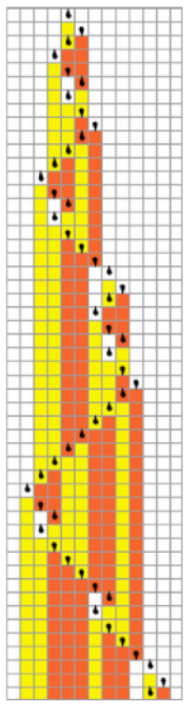

Fig. 3: The first 50 steps in evolution (from a blank tape) of the simplest Turing machine capable of universal computation.

But what we learn now is that setup is in a sense not required. Instead, the phenomenon of universality seems to be ubiquitous – in systems in nature, mathematics and elsewhere.

What is the significance of this? At a practical level, the occurrence of universality in so many kinds of systems makes it easier to imagine creating practical computation systems in a much wider variety of ways. But universality also implies some important limitations on science – particularly a phenomenon I call computational irreducibility, which fundamentally limits the predictability of processes.

In the past, one might have assumed that most systems in nature would be computationally much simpler than universal Turing machines. But now we have evidence that essentially whenever we see complex behaviour in nature, it is associated with processes that achieve universal computation.

But could the systems do more than universal Turing machines? We do not yet know for sure. But as we successfully model more and more systems without the need to go beyond Turing machines, it seems less and less plausible that this will be necessary.

The ultimate test, however, is whether we can model our whole physical universe using something equivalent to a universal Turing machine. We can certainly imagine a universe that operates like some behaviour of a Turing machine. But the issue is whether our actual universe does so.

Until we know the ultimate theory of physics, we will not be sure of the answer. The history of traditional physics might seem to suggest that as one goes to smaller and smaller scales, it must take something more and more complicated to describe the physical world. But from my exploration of cellular automata and similar systems, I have developed a quite different intuition – which now makes it seem quite plausible that there could be a simple rule that underlies everything in our universe.

The evidence I have increasingly seems to support this view. For more and more phenomena familiar from known physics seem to emerge from extremely simple underlying rules – operating underneath such traditional notions as space and time. And if it is true that there is a rule that in effect reproduces our universe, then we will know for certain that everything in our universe can in fact, in principle, be described by something equivalent to a universal Turing machine.

The Turing machine has then gone from a mathematical idealisation, to a model for computational processes – to a complete means of describing everything that can exist in our universe. It also has gone from something of importance only for questions of mathematical or theoretical computation to something whose features and properties must pervade all systems that we experience – and that must be considered fundamental to science.

We can trace the foundations of so much modern technology to the idea of universal computation set forth in Turing's paper. Increasingly, universal computation seems destined to be central to all sorts of issues in science. In a sense, though, I believe we are still early on the curve of seeing the full significance of universal computation. Computers, for example, became widespread though the development of first databases, then word processing, then the web – none of which make central use of universal computation. But now, as we see knowledge begin to become computable on a large scale, there is finally starting to be deeper use of universal computation. And there is still much more to come.

Almost everything we build – whether with molecules, social systems, whatever – will rely on universal computation. Our intuition about how the world works – and what we can know and predict about the world – will be based on thinking about universal computation. Even the future of the human condition will rely centrally on universal computation.

Newton's Principia Mathematica is often cited as the most important single work in the history of science – for it was the key milestone that enabled the development of the exact sciences and the tradition of engineering that arose from them. And certainly Newton was more aware of the

significance of his work than Turing. But in the modest paper by Alan Turing reproduced here lie the seeds of what is surely the most important single intellectual development of the twentieth century, and possibly of all of modern human history.

References

Hilbert, D., Bernays, P., 1934. Grundlagen der Mathematik, vol.1, Springer, Berlin, p. 209.
Burks, A.W., Goldstine, H., von Neumann, J., 1947. Preliminary discussion of the Logical Design of an Electronic Computing Instrument, Institute for Advanced Study, Princeton, NJ.
Church , Alonzo 1936a. An unsolvable problem of elementary number theory, Am. J. Math. 58, 345–363.
Church, Alonzo, 1936b. A note on the Entseheidungsproblem, J. Symb. Logic 1, 40–41.
Gödel, K., 1931. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme, I. Monatshefte für Mathematik und Physik, 38, 173–198.
Hobson, E.W., 1921. Theory of Functions of a Real Variable, second edn. 87–88.
Kleene, S.C., 1935. A theory of positive integers in formal logic. Amer. J. Math. 57, 153–173, 219–244.
Leibniz, G.W., 1966. Zur allgemeinen Charakteristik. Hauptschriften zur Grundlegung der Philosophie. pp. 30–31, Philosophische Werke Band 1, Translated by Artur Buchenau. Reviewed and with introduction and notes published by E. Cassirer. Hamburg, Felix Meiner.
McCulloch, W., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. 7, 115–133.
Smith, A., 2007. The Wolfram 2,3 Turing Machine Research Prize. http://www.wolframscience.com/prizes/tm23/solved.html. (accessed April 1, 2012)
Turing, A.M., 1936. On computable numbers with an application to the Entscheidungsproblem. Proc. London Math. Soc. 2 (42) 230–265.
Wilhelm, A., 1928. Zum hilbertschen aufbau der reellen zahlen. Mathematische Ann. 99, 118–133.
Wolfram, S., 2002. A New Kind of Science, Wolfram Media, Champaign, IL. USA.

Martin Davis illuminates —

THREE PROOFS OF THE UNSOLVABILITY OF THE ENTSCHEIDUNGSPROBLEM

The Entscheidungsproblem

What is usually called first order logic is an encapsulation of logical reasoning, especially as it occurs in proofs in mathematics. In this setup propositions become 'sentences' in which symbols for basic logical notions (like *and, not, all*) are combined with non-logical symbols for the specific items being reasoned about.

The Entscheidungsproblem may be stated in the following three equivalent ways:

(1) Find an algorithm to determine whether a given sentence of first order logic is *valid,* that is, true regardless of what specific objects and relationships are being reasoned about.
(2) Find an algorithm to determine whether a given sentence of first order logic is satisfiable, that is, true for some specific objects and relationships.
(3) Find an algorithm to determine given some sentences of first order logic regarded as *premises* and another sentence, being a desired *conclusion*, whether that conclusion is provable from the premises using the rules of proof for first order logic.

That the first two are equivalent is clear because the sentence A is valid if and only if not-A is not satisfiable. To show that the third formulation is equivalent to the first two, it suffices to note that, according to the *Completeness Theorem* that Gödel established in his doctoral dissertation, a sentence A is a logical consequence of premises $P_1, P_2, \ldots, P_n$ if and only if the sentence

$$P_1 \& P_2 \& \ldots \& P_n \Rightarrow A$$

is valid.

If we think of the premises as the axioms of some mathematical domain, we can see that an actual algorithm solving the Entscheidungsproblem would reduce all of mathematics, at least in principle, to mechanical calculation. Presumably it was this that led Hilbert to characterise the Entscheidungsproblem as 'the fundamental problem of mathematical logic'. This was enough to convince G. H. Hardy that there could be no such algorithm. He declared:

> There is of course no such theorem, and this is very fortunate, since if there were we should have a mechanical set of rules for the solution of all mathematical problems, and our activities as mathematicians would come to an end.[1]

In a different context, Poincaré made it clear that he found the whole idea of formalisation of mathematics ridiculous:

> Thus it will be readily understood that in order to demonstrate a theorem, it is not necessary or even useful to know what it means. ... we might imagine a machine where we should put in axioms at one end and take out theorems at the other, like that legendary machine in Chicago where pigs go in alive and come out transformed into hams and sausages. It is no more necessary for the mathematician than it is for these machines to know what he is doing.[2]

If indeed a solution of the Entscheidungsproblem would have reduced all of mathematics to mechanical calculation, then from the existence of any mathematical problem provably algorithmically unsolvable, the unsolvability of the Entscheidungsproblem itself should follow. The unsolvability proofs of Church and of Turing each follow this approach.

1. Church's Proof

Alonzo Church, in his historic (Church, 1936), provided a rigorous formal characterisation of what it means to be solvable by means of an algorithm, what has come to be known as *Church's Thesis*. This made it possible for him to prove that one specific problem is algorithmically unsolvable. In his work, Church (1936a) specified a finite set of premises that encapsulate this specific problem so faithfully that an algorithm for testing whether a given conclusion follows from those premises would also provide an algorithmic solution to that specific problem, although the problem is known to be unsolvable. From this contradiction Church could conclude that the Entscheidungsproblem itself is unsolvable.

Instead of indicating his satisfaction at having settled a fundamental problem, Church expressed a doubt. He noted the fact that the proof of Gödel's Completeness Theorem is necessarily non-constructive, dealing as it must, with the notion of validity that refers to arbitrary sets and

[1] Davis (2000), Chapter 7.

[2] Davis (2000), Chapter 5.

relations. And showing a surprisingly extreme constructivist stance, Church cast doubt on his own accomplishment saying:

> The unsolvability of [all three forms] of the Entscheidungsproblem ... cannot, therefore, be regarded as established beyond question.

2. Turing's Proof

Alan Turing began working on the Entscheidungsproblem with no knowledge of what Church was doing. He began with his own explication of algorithmic solvability, or as he called it, computability, in terms of extremely simple abstract computing machines, what are now called 'Turing machines'. By analyzing what someone actually does when computing something, he provided a convincing argument for the adequacy of his formulation. (Later he proved that his concept was equivalent to Church's.) Like Church, he used what he had done to prove that a certain specific problem is unsolvable. In Turing's case the unsolvable problem was to determine algorithmically whether one of his given machines would ever produce a particular symbol, the so-called 'printing problem'. His remarkable paper (Turing, 1936) contained as a byproduct a construction showing that a particular one of his machines, his *universal machine*, could all by itself duplicate anything that any of his machines could do, and thereby showed in schematic form the possibility of an all-purpose computer.

By using sentences of first order logic to mimic the step-by-step behaviour of his machines he was able to associate with any one of his machines a corresponding sentence of first order logic that is valid if and only if that machine eventually produces the symbol 0. Thus an algorithm for validity (the first form of the Entscheidungsproblem in our list of three) would automatically provide a solution to the printing problem, although it is, in fact, unsolvable. Thus Turing could conclude that the Entscheidungsproblem is algorithmically unsolvable. Turing's method turned out to be quite fertile and was later used successfully to obtain significant new results.[3]

3. The should-have-been Gödel–Kleene Proof

Church and Turing's proofs were both published in 1936. Gödel's epochal paper (Gödel, 1931) in which he showed that formal logical systems in which a modicum of mathematics could be developed would inevitably be incomplete: there would be straightforward mathematical statements which could be neither proved nor disproved in that system. But the paper is extremely rich and, in particular contains an application to the Entscheidungsproblem. This application involves a class of functions from natural numbers to natural numbers called *primitive recursive*.[4] The key thing about this class is that included functions that can be defined *recursively*. As an example, we may consider the function 2^x which can be defined by the recursion:

$$2^0 = 1; \quad 2^{k+1} = 2 \times 2^k.$$

What Gödel proved relating to the Entscheidungsproblem in his remarkable 1931 paper is that corresponding to any given primitive recursive function $f(x)$, there is a sentence of first order logic, which is satisfiable if and only if $f(x) = 0$ for all x.[5] So to get the unsolvability of the Entscheidungsproblem (in our second form) it suffices to show that there is no algorithm for determining of a given primitive recursive function $f(x)$ whether it is equal to 0 for all x.

[3] See Börger et al. (1997).

[4] Not quite the term Gödel used.

[5] Gödel (1931) Theorem X.

Stephen Kleene was a student of and collaborator with Church during the exciting period when these dramatic results were being developed. Kleene's paper (Kleene, 1936) is mainly remembered for his *Normal Form Theorem* in which arbitrary computable functions on the natural numbers are seen to be closely related to primitive recursive functions. Thus for any computable function $f(x)$. Kleene showed that there are a pair of primitive recursive functions $g(x), h(x,y)$ such that:

$$f(x) = g(\min_y(h(x,y) = 0)).$$

In addition, Kleene found a particular primitive recursive function $t(z,x)$ such that the problem of determining for a given value of z whether $t(z,x) = 0$ for all x is algorithmically unsolvable.[6] From this special case, it is clear that there can be no general algorithm to determine of an arbitrary given primitive recursive function $f(x)$ whether it is equal to 0 for all x. So Gödel's result immediately yields the unsolvability of the Entscheidungsproblem.

It seems remarkable that Kleene, who certainly had studied Gödel's 1931 paper and knew it very well, apparently had not noticed this connection.

References

Börger, E., Grädel, E., Gurevich, Y., 1997. The Classical Decision Problem, Springer, Berlin, Heidelberg.

Church, A., 1936. An unsolvable problem in elementary number theory. Am. J. Math. 58, 345–363. Reprinted in Davis, M. (1965).

Church, A., 1936a. A note on the Entscheidungsproblem. J. Symb. Log. 1, 40–41. Correction ibid pp. 101–102. Reprinted with corrections incorporated: Davis, M. (1965).

Copeland, B.J., (Ed.), 2004. The Essential Turing. Oxford University Press.

Davis, M. (Ed.), 1965. The Undecidable. Raven Press, 1965. Reprinted: Dover 2004.

Davis, M., 1982. Why Gödel didn't have Church's Thesis, Inf. Control 54, 3–24.

Davis, M., 2000. The Universal Computer. Turing Centenary edition, A.K. Peters, Norton, New York, 2011.

Gödel, K., 1931. Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I, Monatshefte für Mathematik und Physik, Band 38, pp. 173–198. Reprinted with English translation from: Gödel, K., 1986. On Formally Undecidable Proposition of Principia Mathematica and Related Systems I, pp. 144–195.

Gödel, K., 1986. Collected Works, vol. I, Solomon Feferman et al, eds., Oxford University Press.

Kleene, S.C., 1936. General recursive functions of general numbers, Mathematische Annalen, Band 112, 727–742. Reprinted in Davis, M. (1965).

Petzold, C., 2008. The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine, Wiley, Indianapolis.

Turing, A., 1936. On computable numbers with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. ser. 2, 42, 230–265. Correction: ibid, 43 (1937), pp. 544–546. Reprinted in Davis, M. (1965) pp. 116–154. Reprinted in Turing (2001) pp. 18–56. Reprinted in Copeland, J. (2004) pp. 58–90; 94–96. Reprinted in Petzold, C. (2008) (the original text interspersed with commentary).

Turing, A., 2001. Collected Works: Mathematical Logic. Gandy, R.O., Yates, C.E.M. (Eds.), North-Holland, Amsterdam, London.

[6] See Kleene (1936) XVI. To coordinate the present notation with Kleene's, we would define

$$t(z,x) = \begin{cases} 0 & \text{if } T_1(z,z,x) \\ 1 & \text{otherwise} \end{cases}$$

Samson Abramsky detects —

TWO PUZZLES ABOUT COMPUTATION

1. Introduction

Turing's classical analysis of computation Turing (1936) gives a compelling account of the nature of the computational process; of *how* we compute. This allows the notion of *computability*, of what can in principle be computed, to be captured in a mathematically precise fashion.

The purpose of this note is to raise two different questions, which are rarely if ever considered, and to which, it seems, we lack convincing, systematic answers. These questions can be posed as:

- Why do we compute?
- What do we compute?

The point is not so much that we have no answers to these puzzles, as that we have no established body of theory which gives satisfying, systematic answers, as part of a broader understanding. By raising these questions, we hope to stimulate some thinking in this direction.

These puzzles were raised in Abramsky (2008); see also Adriaans and van Emde Boas (2011).

2. Why Do We Compute?

The first puzzle is simply stated:

Why do we compute?

By this we mean: why do we perform (or build machines and get them to perform) actual, physically embodied computations?

There is, indeed, an obvious answer to this question:

To gain information (which, therefore, we did not previously have).

But — how is this possible?[1] Two problems seems to arise, one stemming from physics, and one from logic.

Problem 1: Doesn't this contradict the second law of thermodynamics?

Problem 2: Isn't the output *implied* by the input?

We shall discuss each of these in turn.

[1] Indeed, I was once challenged on this point by an eminent physicist (now knighted), who demanded to know how I could speak of information increasing in computation when Shannon Information theory tells us that it cannot! My failure to answer this point very convincingly at the time led me to continue to ponder the issue, and eventually gave rise to this discussion.

Problem 1

The problem is that, presumably, information is conserved in the *total* system. The natural response is that, nevertheless, there *can* be information flow between, and information increase in, *subsystems*; just as a body can gain heat from its environment. More precisely, while the entropy of an isolated (total) system cannot decrease, a sub-system *can* decrease its entropy by consuming energy from its environment.

Thus if we wish to speak of information flow and increase, this must be done relative to subsystems. Indeed, the fundamental objects of study should be *open systems*, whose behaviour must be understood in relation to an external environment. Subsystems which can observe incoming information from their environment, and act to send information to their environment, have the capabilities of *agents*.

Moral: Agents and their interactions are intrinsic to the study of information flow and increase in computation. The classical theories of information do not reflect this adequately.

Observer-dependence of information increase? Yorick Wilks (personal communication) has suggested the following additional twist. Consider an equation such as

$$3 \times 5 = 15.$$

The forward direction $3 \times 5 \rightarrow 15$ is obviously a natural direction of computation, where we perform a multiplication. But the reverse direction $15 \rightarrow 3 \times 5$ is also of interest — finding the prime factors of a number! So it seems that the *direction of possible information increase* must be understood as relative to the observer or user of the computation!

Can we in fact find an objective, observer-independent notion of information increase? This seems important to the whole issue of whether information is inherently subjective, or whether it has an objective structure.

Problem 2

The second puzzle is the computational version of what has been called the *scandal of deduction* DAgostino and Floridi (2009); Hintikka (1970); Sequoiah-Grayson (2008). The logical problem is to find the sense in which logical deduction can be informative, since, by the nature of the process, the conclusions are 'logically contained' in the premises. So what has been added by the derivation? This is a rather basic question, which it is surprisingly difficult to find a satisfactory answer to.

Computation can be modelled faithfully as deduction, whether in the sense of deducing the steps that a Turing maching takes starting from its initial configuration, or more directly via the Curry-Howard isomorphism Curry, Feys and Craig (1958); Howard (1980), under which computation can be viewed as performing cut-elimination on proofs, or normalization of proof terms. Thus the same question can be asked of computation: since the result of the computation is logically implied by the program together with the input data, what has been added by computing it?

The same issue can be formulated in terms of the logic programming paradigm, or of querying a relational database: in both cases, the result of the query is a logical consequence of the data- or knowledge-base.

It is, of course, tempting to answer in terms of the complexity of the inference process; but this seems to beg the question. We need to understand first what the inference process is doing for us!

We can also link this puzzle to another well-known issue in logic, namely the principle of *logical omnisicience* in epistemic logic, which is unrealistic yet hard to avoid. This principle can be formulated as follows:

$$[K_a\phi \wedge (\phi \rightarrow \psi)] \rightarrow K_a\psi.$$

It says that the knowledge of agent a is deductively closed: if a knows a proposition ϕ, then he knows all its logical consequences. This is patently untrue in practice, and brings us directly back

to our puzzle concerning computation. We compute to gain information we did not have. We start from the information of knowing the program and its input, and the computation provides us with explicit knowledge of the output. But what does 'explicit' mean?

The computational perspective may indeed provide a usefully clarifying perspective on the issue of logical omniscience, since it provides a context in which the distinction between 'explicit' and 'implicit' knowledge can be made precise. Let us start with the notion of a function. In the 19th century, the idea of a function as a 'rule' — as given by some defining expression — was replaced by its 'set-theoretic semantics' as a set of ordered pairs. In other terminology, a particular defining expression is an *intensional description* of a function, while the set of ordered pairs which it denotes is its *extension*.

A program is exactly an intensional description of a function, with the additional property that this description can be used to explicitly calculate outputs from given inputs in a stepwise, mechanical fashion.[2] We can say that implicit knowledge, in the context of computation, is knowledge of an intensional description; while explicit knowledge, of a data item such as a number, amounts to possessing the *numeral* (in some numbering system) corresponding to that number; or more generally, to possessing a particular form of intensional description which is essentially isomorphic to the extension.

The purpose of computation in these terms is precisely to convert intensional descriptions into extensional ones, or implicit knowledge of an input-output pair into explicit knowledge. The *cost* of this process is calibrated in terms of the resources needed — the number of computation steps, the workspace which may be needed to perform these steps, etc. Thus we return to the usual, 'common-sense' view of computation. The point is that it rests on this distinction between intension and extension, or implicit vs. explicit knowledge.

Another important aspect of why we compute is *data reduction*—getting rid of a lot of the information in the input. Note that normal forms are in general *unmanagably big* Vorobyov (1997). Note also that it is *deletion of data* which creates thermodynamic cost in computation Landauer (1961). Thus we can say that much (or all?) of the actual usefulness of computation lies in getting rid of the hay-stack, leaving only the needle.

The challenge here is to build a useful theory which provides convincing and helpful answers to these questions. In our view these puzzles, naive as they are, point to some natural questions which a truly comprehensive theory of computation, incorporating a 'dynamics of information', should be able to answer.

3. What Do We Compute?

The classical notion of computability as pioneered by Turing Turing (1936) focusses on the key issue of *how* we compute; of what constitutes a computation. However, it relies on pre-existing notions from mathematics as to *what* is computed: numbers, functions, sets, etc.

This idea also served computer science well for many years: it is perfectly natural in many situations to view a computational process in terms of computing an output from an input. This computation may be deterministic, non-deterministic, random, or even quantum, but essentially the same general paradigm applies.

However, as computation has evolved to embrace diverse forms and purposes: distributed, global, mobile, interactive, multi-media, embedded, autonomous, virtual, pervasive, ... the adequacy of this view has become increasingly doubtful.

Traditionally, the *dynamics* of computing systems — their unfolding behaviour in space and time — has been a mere means to the end of computing the function which specifies the algorithmic problem which the system is solving.[3] In much of contemporary computing, the situation is

[2] We refer e.g. to Gandy (1980); Sieg (2002) for attempts to give a precise mathematical characterization of 'mechanical'.

[3] Insofar as the dynamics has been of interest, it has been in quantitative terms, counting the resources which the algorithmic process consumes — leading of course to the notions of algorithmic complexity. Is it too fanciful to speculate

reversed: the *purpose* of the computing system is to exhibit certain behaviour. The *implementation* of this required behaviour will seek to reduce various aspects of the specification to the solution of standard algorithmic problems.

What does the Internet compute?

Surely not a mathematical function ...

Why Does It Matter?

We shall mention two basic issues in the theory of computation which become moot in the light of this issue.

There has been an enormous amount of work on the first, namely the theory of concurrent processes. Despite this huge literature, produced over the past four decades and more, no consensus has been achieved as to what processes *are*, in terms of their essential mathematical structure. Instead, there has been a huge proliferation of different models, calculi, semantics, notions of equivalence. To make the point, we may contrast the situation with the λ-calculus, the beautiful, fundamental calculus of functions introduced by Church at the very point of emergence of the notion of computability Church (1941). Although there are many variants, there is essentially a unique, core calculus which can be presented in a few lines, and which delineates the essential ideas of functional computation. In extreme contrast, there are a huge number of process calculi, and none can be considered as definitive.

Is the notion of process too amorphous, too open to different interpretations and contexts of use, to admit a unified, fundamental theory? Or has the field not yet found its Turing? See Abramsky (2006) for an extended discussion.

The second issue follows on from the first, although it has been much less studied to date. This concerns the Church-Turing thesis of universality of the model of computation. What does this mean when we move to a broader conception of what is computed? And are there any compelling candidates? Is there a widely accepted universal model of interactive or concurrent computation?

As a corollary to the current state of our understanding of processes as described in the previous paragraphs, there is no such clear-cut notion of universality. It is important to understand what is at issue here. If we are interested in the process of computation itself, the structure of interactive behaviour, then on what basis can we judge if one such process is faithfully simulated by another? It is not of course that there are no candidate notions of this kind which have been proposed in the literature; the problem, rather, is that there are far too many of them, reflecting different intuitions, and different operational and application scenarios.

Once again, we must ask: is this embarrassing multitude of diverse and competing notions a necessary reflection of the nature of this notion, or may we hope for an incisive contribution from some future Turing which will unify and organize the field?

References

Abramsky, S., 2006. What are the Fundamental Structures of Concurrency? We still dont know! Electronic Notes in Theoretical Computer Science 162, 37–41.

Abramsky, S., 2008. Information, processes and games. In: van Benthem, J., Adriaans, P. (eds.), Handbook of the Philosophy of Information, Elsevier Science Publishers, Amsterdam, pp. 483–549.

Adriaans, P., van Emde Boas, P., 2011. Computation, information, and the arrow of time. In: Cooper, S.B., Sorbi, A. (eds.), Computability in Context, World Scientific, Singapore, pp. 1–18.

that the lack of an adequate structural theory of processes has been an impediment to fundamental progress in complexity theory?

Church, A., 1941. The Calculi of Lambda Conversion, Vol. 6 of Annals of Mathematics Studies, Princeton University Press, Princeton.
Curry, H.B., Feys, R. and Craig, W., 1958. Combinatory Logic: Vol. 1. North-Holland, Amsterdam.
D'Agostino, M., Floridi, L., 2009. The enduring scandal of deduction. Synthese, 167(2), 271–315.
Gandy, R., 1980. Churchs thesis and principles for mechanisms, Studies in Logic and the Foundations of Mathematics, 101, 123–148.
Hintikka, J., 1970. Information, deduction, and the a priori. Nous 4(2), 135–152.
Howard, W.A., 1980. The formulae-as-types notion of construction. In: Selden, J.P., Hindley, J.R. (eds.), To HB Curry: essays on combinatory logic, lambda calculus and formalism, Academic Press, pp. 479–490.
Landauer, R., 1961. Irreversibility and heat generation in the computing process, IBM Journal of Research and Development 5, 183–191.
Sequoiah-Grayson, S., 2008. The scandal of deduction, Journal of Philosophical Logic 37(1), 67– 94.
Sieg, W., 2002. Calculations by man and machine: Mathematical presentation. In: Gärdenfors, P., Wolenski, J., and Kijania-Placek, K. (eds.): In the scope of logic, methodology, and philosophy of science: volume two of the 11th International Congress of Logic, Methodology and Philosophy of Science, Cracow, August 1999, Kluwer Academic Publishers, Dordrecht, Boston, London, p. 247.
Turing, A.M., 1936. On computable numbers, Proc. of the Lond. Math. Soc. 2(42), 230–65, 1936.
Vorobyov, S.G., 1997. The "hardest" natural decidable theory. In: Proceedings of the 12th Symposium on Logic in Computer Science, IEEE Press, 294–305.

Paul Vitányi illustrates the importance of —

TURING MACHINES AND UNDERSTANDING COMPUTATIONAL COMPLEXITY

1. Introduction

A *Turing machine* refers to a hypothetical machine proposed by Alan M. Turing (1912–54) in 1936 (Turing, 1936) whose computations are intended to give an operational and formal definition of the intuitive notion of computability in the discrete domain. It is a digital device and sufficiently simple to be amenable to theoretical analysis and sufficiently powerful to embrace everything in the discrete domain that is intuitively computable. As if that were not enough, in the theory of computation many major complexity classes can be easily characterised by an appropriately restricted Turing machine; notably, the important classes P and NP and consequently the major question whether P equals NP.

Turing gave a brilliant demonstration that everything that can be reasonably said to be computed by a human computer using a fixed procedure can be computed by such a machine. As Turing claimed, any process that can be naturally called an effective procedure is realised by a Turing machine. This is known as Turing's thesis. Enter Alonzo Church (1903–95). Over the years, all serious attempts to give precise yet intuitively satisfactory definitions of a notion of effective procedure (what Church called effectively calculable function) in the widest possible sense have turned out to be equivalent – to define essentially the same class of processes. The Church–Turing thesis states that a function on the positive integers is effectively calculable if and only if it is computable. An informal accumulation of the tradition in S. C. Kleene (1952) has transformed it to the Computability thesis: there is an objective notion of effective computability independent of a particular formalisation. The informal arguments Turing sets forth in his 1936 paper are as lucid and convincing now as they were then. To us it seems that it is the best introduction to the subject. It gives

the intuitions that lead up to the formal definition, and is in a certain sense a prerequisite of what follows. The reader can find this introduction in Turing (1936) included in this volume. It begins with:

> "All arguments are bound to be, fundamentally, appeals to intuition, and for that 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 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."

2. Formal definition of the Turing machine

We formalise Turing's description as follows: A Turing machine consists of a finite program, called the finite control, capable of manipulating a linear list of cells, called the tape, using one access pointer, called the head. We refer to the two directions on the tape as *right* and *left*. The finite control can be in any one of a finite set of states Q, and each tape cell can contain a 0, a 1, or a *blank* B. Time is discrete and the time instants are ordered $0,1,2,\ldots$, with 0 the time at which the machine starts its computation. At any time, the head is positioned over a particular cell, which it is said to *scan*. At time 0 the head is situated on a distinguished cell on the tape called the *start cell*, and the finite control is in a distinguished state q_0. At time 0 all cells contain Bs, except for a contiguous finite sequence of cells, extending from the start cell to the right, which contain 0's and 1's. This binary sequence is called the *input*. The device can perform the following basic operations:

(1) It can write an element from $A = \{0,1,B\}$ in the cell it scans; and

(2) it can shift the head one cell left or right.

When the device is active it executes these operations at the rate of one operation per time unit (a *step*). At the conclusion of each step, the finite control takes on a state from Q. The device is constructed so that it behaves according to a finite list of rules. These rules determine, from the current state of the finite control and the symbol contained in the cell under scan, the operation to be performed next and the state to enter at the end of the next operation execution.

The rules have format (p,s,a,q): p is the current state of the finite control; s is the symbol under scan; a is the next operation to be executed of type (1) or (2) designated in the obvious sense by an element from $S = \{0,1,B,L,R\}$; and q is the state of the finite control to be entered at the end of this step.

For now, we assume that there are no two distinct quadruples that have their first two elements identical, the device is deterministic. Not every possible combination of the first two elements has to be in the set; in this way we permit the device to perform 'no' operation. In this case we say that the device halts. Hence, we can define a Turing machine by a mapping from a finite subset of $Q \times A$ into $S \times Q$. Given a Turing machine and an input, the Turing machine carries out a uniquely determined succession of operations, which may or may not terminate in a finite number of steps.

Strings and natural numbers are occasionally identified according to the pairing

$$(\epsilon,0),(0,1),(1,2),(00,3),(01,4),(10,5),(11,6),\ldots, \tag{2.1}$$

where ϵ denotes the empty string (with no bits). In the following, we need the notion of a *self-delimiting* code of a binary string. If $x = x_1 \ldots x_n$ is a string of n bits, then its self-delimiting code is $\bar{x} = 1^n 0x$. Clearly, the length $|\bar{x}| = 2|x| + 1$. Encoding a binary string self-delimitingly enables a machine to determine where the string ends reading it from left to right in a single pass and without reading past the last bit of the code.

2.1. *Computable functions*

We can associate a partial function with each Turing machine in the following way: The input to the Turing machine is presented as an n-tuple $(x_1,\ldots,x_n)$ consisting of self-delimiting versions of the x_i's. The integer represented by the maximal binary string (bordered by blanks) of which some bit is scanned, or 0 if a blank is scanned, by the time the machine halts, is called the *output* of the computation. Under this convention for inputs and outputs, each Turing machine defines a partial function from n-tuples of integers onto the integers, $n \geq 1$. We call such a function partial computable. If the Turing machine halts for all inputs, then the function computed is defined for all arguments and we call it total computable. (Instead of *computable* the more ambiguous *recursive* has also been used.) We call a function with range $\{0,1\}$ a *predicate*, with the interpretation that the predicate of an n-tuple of values is *true* if the corresponding function assumes value 1 for that n-tuple of values for its arguments and is *false* or *undefined* otherwise. Hence, we can talk about *partial (total) computable predicates*.

2.2. *Examples of computable functions*

Consider x as a binary string. It is easy to see that the functions $|x|, f(x) = \bar{x}, g(\bar{x}y) = x$, and $h(\bar{x}y) = y$ are partial computable. Functions g and h are not total since the value for input 1111 is not defined. The function $g'(\bar{x}y)$ defined as 1 if $x = y$ and as 0 if $x \neq y$ is a computable predicate. Consider x as an integer. The following functions are basic n-place total computable functions: the *successor* function $\gamma^{(1)}(x) = x+1$, the *zero* function $\zeta^{(n)}(x_1,\ldots,x_n) = 0$, and the *projection* function $\pi_m^{(n)}(x_1,\ldots,x_n) = x_m (1 \leq m \leq n)$. The function $\langle x,y\rangle = \bar{x}y$ is a total computable one-to-one mapping from pairs of natural numbers into the natural numbers. We can easily extend this scheme to obtain a total computable one-to-one mapping from k-tuples of integers into the integers, for each fixed k. Define $\langle n_1,n_2,\ldots,n_k\rangle = \langle n_1,\langle n_2,\ldots,n_k\rangle\rangle$. Another total recursive one-to-one mapping from k-tuples of integers into the integers is $\langle n_1,n_2,\ldots,n_k\rangle = \bar{n}_1\ldots\bar{n}_{k-1}\bar{n}_k$.

3. Computability thesis and the universal Turing machine

The class of algorithmically computable numerical functions (in the intuitive sense) coincides with the class of partial computable functions. Originally intended as a proposal to henceforth supply intuitive terms such as 'computable' and 'effective procedure' with a precise meaning, the Computability thesis has come into use as shorthand for a claim that from a given description of a procedure in terms of an informal set of instructions we can derive a formal one in terms of Turing machines.

It is possible to give an effective (computable) one-to-one pairing between natural numbers and Turing machines. This is called an *effective enumeration*. One way to do this is to encode the table of rules of each Turing machine in binary, in a canonical way.

The only thing we have to do for every Turing machine is to encode the defining mapping $T : Q \times A \to S \times Q$. Giving each element of $Q \bigcup S$ a unique binary code requires s bits for each such element, with $s = \lceil \log(|Q|+5) \rceil$. Denote the encoding function by e. Then the quadruple $(p,0,B,q)$ is encoded as $e(p)e(0)e(B)e(q)$. If the number of rules is r, then $r \leq 3|Q|$. We agree to consider the state of the first rule as the start state. The entire list of quadruples,

$$T = (p_1,t_1,s_1,q_1),(p_2,t_2,s_2,q_2),\ldots,(p_r,t_r,s_r,q_r),$$

is encoded as

$$E(T) = \bar{s}\bar{r}e(p_1)e(t_1)e(s_1)e(q_1)\ldots e(p_r)e(t_r)e(s_r)e(q_r).$$

Note that $l(E(T)) \leq 4rs + 2\log rs + 4$. (Moreover, E is self-delimiting, which is convenient in situations in which we want to recognise the substring $E(T)$ as prefix of a larger string.)

We order the resulting binary strings lexicographically (according to increasing length). We assign an index, or Gödel number, $n(T)$ to each Turing machine T by defining $n(T) = i$ if $E(T)$ is the i-th element in the lexicographic order of Turing machine codes. This yields a sequence of Turing machines $T_1, T_2, \ldots$ that constitutes the effective enumeration. One can construct a Turing machine to decide whether a given binary string x encodes a Turing machine, by checking whether it can be decoded according to the scheme above, that the tuple elements belong to $Q \times A \times S \times Q$, followed by a check whether any two different rules start with the same two elements. This observation enables us to construct *universal* Turing machines.

A universal Turing machine U is a Turing machine that can imitate the behaviour of any other Turing machine T. It is a fundamental result that such machines exist and can be constructed effectively. Only a suitable description of T's finite program and input needs to be entered on U's tape initially. To execute the consecutive actions that T would perform on its own tape, U uses T's description to simulate T's actions on a representation of T's tape contents. Such a machine U is also called *computation universal*. In fact, there are infinitely many such U's.

We focus on a universal Turing machine U that uses the encoding above. It is not difficult, but tedious, to define a Turing machine in quadruple format that expects inputs of the format $E(T)p$ and is undefined for inputs not of that form. The machine U starts to execute the successive operations of T using p as input and the description $E(T)$ of T it has found so that $U(E(T)p) = T(p)$ for every T and p. We omit the explicit construction of U.

For the contemporary reader there should be nothing mysterious in the concept of a general-purpose computer which can perform any computation when supplied with an appropriate program. The surprising thing is that a general-purpose computer can be very simple: M. Minsky (1967) has shown that four tape symbols and seven states suffice easily in the above scheme. This machine can be changed to, in the sense of being simulated by, our format using tape symbols $\{0, 1, B\}$ at the cost of an increase in the number of states. The last reference contains an excellent discussion of Turing machines, their computations and related machines. The effective enumeration of Turing machines $T_1, T_2, \ldots$ determines an effective enumeration of partial computable functions $\varphi_1, \varphi_2, \ldots$ such that φ_i is the function computed by T_i, for all i. It is important to distinguish between a function ψ and a name for ψ. A name for ψ can be an algorithm that computes ψ, in the form of a Turing machine T. It can also be a natural number i such that ψ equals φ_i in the above list. We call i an index for ψ. Thus, each partial computable ψ occurs many times in the given effective enumeration, that is, it has many indices.

4. Undecidability of the halting problem

Turing's paper (Turing, 1936), and more so Kurt Gödel's paper (Gödel, 1931), where such a result first appeared, are celebrated for showing that certain well-defined questions in the mathematical domain cannot be settled by any effective procedure for answering questions. The following 'machine form' of this undecidability result is due to Turing and Church: 'which machine computations eventually terminate with a definite result, and which machine computations go on forever without a definite conclusion?' This is sometimes called the halting problem.

Since all machines can be simulated by the universal Turing machine U, this question cannot be decided in the case of the single machine U, or more generally for any other individual universal machine. The following theorem due to Turing (1936), formalises this discussion. Let $\varphi_1, \varphi_2, \ldots$ be the standard enumeration of partial computable functions and write $\varphi(x) < \infty$ if $\varphi(x)$ is defined and write $\varphi(x) = \infty$ otherwise. Define $K_0 = \{\langle x, y\rangle : \varphi_x(y) < \infty\}$ as the *halting set*.

Theorem 4.1. *The halting set K_0 is not computable.*

The theorem of Turing on the incomputability of the halting set was preceded by (and was intended as an alternative way to show) the famous (first) Incompleteness Theorem of Kurt Gödel in 1931. Recall that a formal theory T consists of a set of well-formed formulas, formulas for short.

For convenience these formulas are taken to be finite binary strings. Invariably, the formulas are specified in such a way that an effective procedure exists that decides which strings are formulas and which strings are not.

The formulas are the objects of interest of the theory and constitute the meaningful statements. With each theory we associate a set of true formulas and a set of provable formulas. The set of true formulas is *true* according to some (often non-constructive) criterion of truth. The set of provable formulas is *provable* according to some (usually effective) syntactic notion of proof.

A theory T is simply any set of formulas. A theory is axiomatisable if it can be effectively enumerated. For instance, its axioms (initial formulas) can be effectively enumerated and there is an effective procedure that enumerates all proofs for formulas in T from the axioms. A theory is decidable if it is a recursive set. A theory T is consistent if not both formula x and and its negation $\neg x$ are in T. A theory T is sound if each formula x in T is true (with respect to the standard model of the natural numbers).

Hence, soundness implies consistency. A particularly important example of an axiomatisable theory is Peano arithmetic, which axiomatises the standard elementary theory of the natural numbers.

Theorem 4.2. *There is a computably enumerable set, say the set K_0 defined above, such that for every axiomatisable theory T that is sound and extends Peano arithmetic, there is a number n such that the formula '$n \notin K_0$' is true but not provable in T.*

In his original proof, Gödel uses diagonalisation to prove the incompleteness of any sufficiently rich logical theory T with a computably enumerable axiom system, such as Peano arithmetic. By his technique he exhibits for such a theory an explicit construction of an undecidable statement y that says of itself *I am unprovable in T*. The formulation in terms of computable function theory is due to A. Church and S. C. Kleene.

Turing's idea was to give a formal meaning to the notion of 'giving a proof.' Intuitively, a proof is a sort of computation where every step follows (and follows logically) from the previous one, starting from the input. To put everything as broad as possible, Turing analyses the notion of 'computation' from an 'input' to an 'output' and uses this to give an alternative proof of Gödel's theorem.

Prominent examples of uncomputable functions are the Kolmogorov complexity function and the universal algorithmic probability function. These are the fundamental notions in Li and Vitányi (2008) and, among others, Downey and Hirschfeldt (2010); Nies (2009).

5. Complexity of computations

Theoretically, every intuitively computable (effectively calculable) function is computable by a personal computer or by a Turing machine. But a computation that takes 2^n steps on an input of length n would not be regarded as practical or feasible. No computer would ever finish such a computation in the lifetime of the universe even with n merely 1000. For example, if we have 10^9 processors each taking 10^9 steps/s, then we can execute $3.1 \times 10^{25} < 2^{100}$ steps/year. Computational complexity theory tries to identify problems that are feasibly computable.

In computational complexity theory, we are often concerned with languages. A language L over a finite alphabet Σ is simply a subset of Σ^*. We say that a Turing machine accepts a language L if it outputs 1 when the input is a member of L and outputs 0 otherwise. That is, the Turing machine computes a predicate.

Let T be a Turing machine. If for every input of length n we have that T makes at most $t(n)$ moves before it halts, then we say that T runs in time $t(n)$, or has time complexity $t(n)$. If T uses at most $s(n)$ tape cells in the above computations, then we say that T uses $s(n)$ space, or has space complexity $s(n)$.

For convenience, we often give the Turing machine in Fig. 1 a few more work tapes and designate one tape as a read-only input tape. Thus, each transition rule will be of the form $(p, \bar{s}, a, q)$, where $\bar{s}$ contains the scanned symbols on all the tapes, and p, a, q are as above, except that an operation now involves moving maybe more than one head.

We sometimes also make a Turing machine non-deterministic by allowing two distinct transition rules to have identical first two components. That is, a non-deterministic Turing machine may have different alternative moves at each step. Such a machine accepts if one accepting path leads to acceptance. Turing machines are deterministic unless it is explicitly stated otherwise.

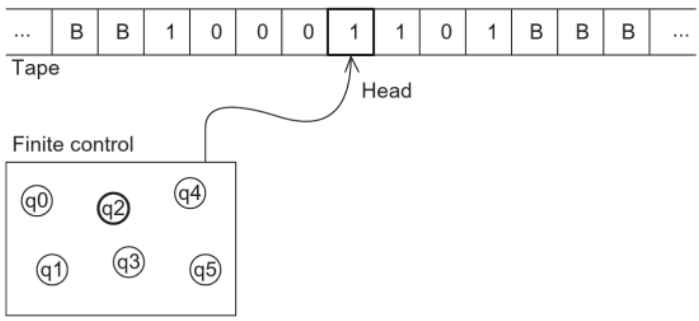

Fig. 1: Turing machine.

- DTIME[$t(n)$] is the set of languages accepted by multitape deterministic Turing machines in time $O(t(n))$;
- NTIME[$t(n)$] is the set of languages accepted by multitape non-deterministic Turing machines in time $O(t(n))$;
- DSPACE[$s(n)$] is the set of languages accepted by multitape deterministic Turing machines in $O(s(n))$ space;
- NSPACE[$s(n)$] is the set of languages accepted by multitape non-deterministic Turing machines in $O(s(n))$ space.
- With c running through the natural numbers:
  P is the complexity class $\bigcup_c \text{DTIME}[n^c]$;
  NP is the complexity class $\bigcup_c \text{NTIME}[n^c]$;
  PSPACE is the complexity class $\bigcup_c \text{DSPACE}[n^c]$.

Languages in P, that is, languages acceptable in polynomial time, are considered feasibly computable. The non-deterministic version for PSPACE turns out to be identical to PSPACE by Savitch's Theorem (Savitch, 1970), which states that $\text{NSPACE}[s(n)] = \text{DSPACE}[(s(n))^2]$. The following relationships hold trivially, $\text{P} \subseteq \text{NP} \subseteq \text{PSPACE}$. It is one of the most fundamental open questions in computer science and mathematics to prove whether either of the above inclusions is proper. Research in computational complexity theory focuses on these questions. In order to solve these problems, one can identify the hardest problems in NP or PSPACE. The Bible of this area is the works of Garey and Johnson (1979).

6. Importance of the Turing machine

In the last three quarters of a century, the Turing machine model has proven to be of priceless value for the development of the science of data processing. All theory development reaches back to this format. The model has become so dominant that new other models that are not polynomial-time reducible to Turing machines are viewed as not realistic (the so-called polynomial-time Computability thesis).

Without explaining terms, the *random access machine* (RAM) with *logarithmic cost*, or *unit cost* without multiplications, is viewed as realistic, while the unit cost RAM with multiplications or

the *parallel random access machine* (PRAM) are not so viewed. New notions, such as randomised computations as in the works by Motwani and Raghavan (1995) (like the fast primality tests used in Internet cryptographical protocols) are analysed using *probabilistic* Turing machines.

In 1980 the Nobelist Richard Feynman proposed a *quantum computer*, in effect an analogous version of a quantum system. Contrary to digital computers (classical, quantum or otherwise), an analogue computer works with continuous variables and simulates the system we want to solve directly: for example, a wind tunnel with a model aircraft simulates the aeroflow and in particular non-laminar turbulence of the aimed-for actual aircraft. In practice, analogue computers have worked only for special problems. In contrast, the digital computer, where everything is expressed in bits, has proven to be universally applicable. Feynman's innovative idea was without issue until D. Deutsch (1985) put the proposal in the form of a quantum Turing machine, that is, a digital quantum computer. This digital development exploded the area both theoretically and applied to the great area of *quantum computing*.

References

Deutsch, D., 1985. Quantum theory, the Church–Turing principle and the universal quantum computer. Proc. Royal Soc. Lond. A. 400, 97–117.

Downey, R.G., Hirschfeldt, D., 2010. Algorithmic Randomness and Complexity (Theory and Applications of Computability), Springer, New York.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, New York.

Gödel, K., 1931. Über formal unentscheidbare Stze der Principia Mathematica und verwandter Systeme, I. Monatshefte für Mathematik und Physik, 38, 173–198.

Kleene, S.C., 1952. Introduction to Metamathematics, Van Nostrand, New York.

Li, M., Vitányi, P.M.B., 2008. An Introduction to Kolmogorov Complexity and Its Applications, third ed. Springer, New York.

Minsky, M., 1967. Computation: Finite and Infinite Machines, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Motwani, R., Raghavan, P., 1995. Randomised Algorithms, Cambridge Univ. Press, Cambridge, UK.

Nies, A., 2009. Computability and Randomness, Oxford Univ. Press, USA.

Savitch, W.J., 1970. Relationships between nondeterministic and deterministic tape complexities. J. Comput. Syst. Sci. 4 (2), 177–192.

Turing, A.M., 1936. On computable numbers, with an application to the entscheidungsproblem. Proc. Lond. Math. Soc. 2 (42), 230–265. "Correction", 43(1937), 544–546.

Gregory Chaitin traces the path —

FROM THE HALTING PROBLEM TO THE HALTING PROBABILITY

In this remarkable paper, Turing starts by observing that most real numbers are uncomputable – indeed this is the case with probability one. Furthermore he observes that incompleteness is a corollary of uncomputability, because if it is always possible to prove whether or not something is the case using a fixed formal axiomatic theory, then there is in principle a mechanical procedure for searching through all possible proofs and mechanically deciding the answer.

This makes incompleteness much more natural and fundamental than the assertion 'I am unprovable!' that is true if and only if it is unprovable, that was constructed by Gödel. So, following Turing 1936, we have a much bigger problem than following Gödel (1931).

On the one hand he taketh away, on the other he giveth, for although Turing shows that formal languages for mathematical reasoning are necessarily incomplete, he also shows that formal programming languages can have a kind of completeness called *universality*. No formal language can express all possible proofs, but programming languages are commonly universal, that is to say, capable of expressing essentially any algorithm.

In our *Search for the Perfect Language* (to echo Umberto Eco), let us now consider how expressive different programming languages can be. Given a particular programming language, two important things to consider are the complexity $H(x)$, namely the size in bits of the smallest program to calculate x as a function of x, and the corresponding probability $P(x)$ that a program whose bits are chosen by using independent tosses of a fair coin will compute x.

We are thus led to select a subset of the Turing universal languages that minimise H and maximise P; one way to define such a language is to consider a universal computer U that runs self-delimiting binary computer programs $\pi_C p$ defined as follows:

$$U(\pi_C p) = C(p).$$

In other words, the result of running $\pi_C p$ on U is the same as the result of running p on C.

Any two such universal languages U and V will necessarily have

$$|H_U(x) - H_V(x)| < c$$

and

$$P_U(x) > P_V(x) \times 2^{-c}, \qquad P_V(x) > P_U(x) \times 2^{-c}.$$

It is in this precise sense that such a universal U minimises H and maximises P.

Using such a U we can define the halting probability Ω, for example, as follows:

$$\Omega = \sum P(n)$$

over all positive integers n, or alternatively

$$\Omega' = \sum 2^{-H(n)},$$

which has a slightly different numerical value but essentially the same paradoxical properties.

What are these properties? Ω is a form of concentrated mathematical creativity, or, alternatively, a particularly economical Turing oracle for the halting problem, because knowing N bits of the dyadic expansion of Ω enables one to solve the halting problem for all programs that compute a positive integer that are up to N bits in size. It follows that the bits of the dyadic expansion of Ω are irreducible mathematical information; they cannot be compressed into a theory smaller than they are.

More precisely, it takes a formal theory of complexity $\geq N - c$ (one requiring a $\geq N - c$ bit program to enumerate all its theorems) to enable us to determine N bits of Ω. From this it follows that Ω is Borel normal, so that Ω is a particularly natural example of a normal number. In 1933, Turing's friend David Champernowne found a natural example of a number normal for blocks of all size in base-ten; Ω provably has this property in any base.[1]

From a philosophical point of view, however, the most striking thing about Ω is that it provides a perfect simulation in pure mathematics, where all truths are necessary truths, of contingent, accidental truths – i.e., of truths such as historical facts or biological frozen accidents.

[1] Andrew Hodges conjectures that Turing's work on normal numbers helped Turing to formulate the notion of a computable real; see Hodges' review of Copeland's *The Essential Turing* in the November 2006 *AMS Notices* and Turing's *A Note on Normal Numbers* in Part II.

Indeed, I have just recently come to understand that the most important property of Ω is that it opens a door for us from mathematics to biology. The halting probability Ω contains infinite irreducible complexity and in a sense shows that pure mathematics is even more biological then biology itself, which merely contains extremely large finite complexity. For each bit of the dyadic expansion of Ω is one bit of independent, irreducible mathematical information, while the human genome is merely 3×10^9 bases $= 6 \times 10^9$ bits of information.

Robert Irving Soare expands on —

TURING AND THE ART OF CLASSICAL COMPUTABILITY

1. Mathematics as an art

Mathematics is an art as well as a science. It is an art in the sense of a *skill* as in Donald Knuth's series, *The Art of Computer Programming,* but it is also an art in the sense of an *esthetic endeavor* with inherent *beauty*, which is recognised by all mathematicians.

One of the world's leading art treasures is Michelangelo's statue of *David* as a young man displayed in the Accademia Gallery in Florence. There is a long aisle to approach the statue of David. The aisle is flanked by the statues of Michelangelo's unfinished slaves struggling as if to emerge from the blocks of marble. These figures reveal Michelangelo's work process. There are practically no details, and yet they possess a weight and power beyond their physical proportions. Michelangelo thought of himself not as carving a statue but as seeing clearly the figure within the marble and then chipping away the marble to release it. The unfinished slaves are perhaps a more revealing example of this talent than the finished statue of David.

Similarly, it was Alan Turing (1936) and (1939) who saw the figure of computability in the marble more clearly than anyone else. Finding a formal definition for effectively calculable functions was the first step, but *demonstrating* that it captured effective calculability was as much an artistic achievement as a mathematical one.

2. Defining the effectively calculable functions

The *Entscheidungsproblem,* the decision problem for first order logic, was described in the works of Hilbert and Ackermann (1928). To show this problem unsolvable one first had to mathematically define the effectively calculable functions. From 1931 to 1934, Church and his student Kleene developed the λ-definable functions. Church privately proposed to Gödel in 1934 that λ-definable functions should be identified with the effectively calculable functions. Gödel rejected this as 'thoroughly unsatisfactory.'

Gödel (1934) defined the *Herbrand–Gödel (HG) recursive functions*, a class of functions as a deductive formal system with initial functions and with two rules of inference to derive new functions. Church (1936) proposed *Church's Thesis* that a function is effectively calculable if and only if it is Herbrand–Gödel recursive. Gödel still did not accept it. Kleene (1936) then defined the *μ-recursive functions* by combining the (Gödel) numbering of syntax in Gödel's Incompleteness Theorem (1931) with the HG recursive functions. This definition is mathematically correct

and prevailed for several decades in research papers from 1935 to 1965, but it is not intuitive, being based on two unintuitive formalisms. By 1936 Gödel knew these definitions, and their formal mathematical equivalence but he did not accept any of them. Indeed, Gödel suggested that it might not be possible to give a mathematical definition of calculability, and he wrote in footnote 3 of (1934) '*... the notion of finite computation is not defined, but serves as a heuristic principle'*.

Turing (1936) brought a new vision of human computability. Turing's remarkable achievement consisted of several parts that we sketch only briefly because they are very well-known. Turing: (1) defined an automatic machine (*a*-machine) based on his model of how a human being might carry out a calculation; (2) defined a universal Turing machine whose inputs included both programs and integers and could simulate any Turing machine on any input; (3) gave an extraordinary demonstration that any function calculated by a human being could be computed by an *a*-machine. Turing then stated what was later known as *Turing's Thesis* that a function on the integers is computable by a finite procedure if and only if it is computable by a Turing machine.

First, Turing gave a model based on a mechanistic approach to human computing, something the previous models lacked. Perhaps, even more impressive was Turing's careful analysis in component parts of how a human being might calculate and then an argument why his Turing machine could simulate this calculation. By comparison, Church (1936) tried to carry out a similar argument that any calculable function is HG recursive, but Gandy (1988, p. 79) and Sieg (1994, pp. 80, 87) pointed out the flaws in Church's argument. Gödel never accepted Church's Thesis, but he accepted Turing's Thesis at once, and stated:

'That this is really the correct definition of mechanical computability was established beyond any doubt by Turing'. *Gödel Collected Works Volume III, 1995, Section 3.3*:

'But I was completely convinced only by Turing's paper'. *Gödel: letter to Kreisel of May 1, 1968 (Sieg, 1994, p. 88).*

'The greatest improvement was made possible through the precise definition of the concept of finite procedure, ... This concept, ... is equivalent to the concept of a 'computable function of integers' *Gödel (1951, pp. 304–305), Gibbs lecture*.

Kleene (1981b, p. 49) wrote, 'Turing's computability is intrinsically persuasive' but 'λ-definability is not intrinsically persuasive' and 'general recursiveness scarcely so (its author Gödel being at that time not at all persuaded).' Kleene wrote in his second book (1967, p. 233), "Turing's machine concept arises from a direct effort to analyze computation procedures as we know them intuitively into elementary operations. Turing argued that repetitions of his elementary operations would suffice for any possible computation. For this reason, Turing computability suggests the thesis more immediately than the other equivalent notions and so we choose it for our exposition."

Church, in his review (1937) of Turing (1936) wrote, Computability by a Turing machine, 'has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately – i.e., without the necessity of proving preliminary theorems'.

3. Why Turing and not Church?

Why give so much credit to Turing and not to Church? In 1923–27 Church was explaining the Hilbert papers to his Princeton thesis adviser, Oswald Veblen. Turing heard of them only a decade later in the Cambridge seminar of M.H.A Newman. In 1936 when Church proposed Church's Thesis, he was a full professor at Princeton in 1936 when Turing was a mere graduate student. Church used the model of the Herbrand–Gödel recursive functions, defined by Gödel, the most eminent logician at that time. They used the concept of recursion (induction) that had appeared in mathematics since Dedekind (1888).

Turing machines were a fanciful new invention without such a well-known, mathematical foundation as recursion. By 1934 Church and Kleene had shown that most number theoretic functions were λ-definable and therefore recursive, giving clear evidence for Church's Thesis. Church was the first to propose Church's Thesis first in 1934 for the λ-definable functions and then in 1935–36 for the Herbrand–Gödel recursive functions, even though Gödel did not believe it. Church got it right and he got it first. The effectively calculable functions *are* the recursive functions.

If this had been the solution to a purely mathematical problem in number theory, Church would have received at least half the credit and Turing would have been credited with a later but independent solution to the problem. By any purely quantifiable evaluation Church's contribution was at least as important as Turing's. Gödel's Incompleteness Theorem (1931) and his proof (1940) of the consistency of CH and AC were purely mathematical problems not requiring one to make mathematically precise an informal concept like calculability. However, characterising human computability was *not* a purely quantifiable process. Gödel (1946, p. 84) wrote, 'one [Turing] has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e., one not depending on the formalism chosen'.

4. Why Michelangelo and not Donatello?

Donatello (1386–1466) was a sculptor in Florence. In 1430 he created the bronze statue of *David,* his most famous work. This was a remarkable work, innovative in many ways, the first free-standing nude statue since ancient times, the first major work of Renaissance sculpture. Now compare this to Michelangelo's *David,* in 1504, the most famous statue in the world. Michelangelo broke away from the traditional way of representing David, with sword in hand and with the giant's head at his feet (as with Donatello). Michelangelo has caught David tense with increasing power as he is about to go to battle. Michelangelo places him in perfect contraposto outdoing the Greek representations of heros.

Michelangelo and Turing both completely transcended conventional approaches. They created something *completely new* from their own visions, something that went far beyond the achievements of their contemporaries. Second, both emphasised the *human form*. Michelangelo brought out the human form in his statues and the Sistine ceiling. Turing invented a system that simulates how a human being computes and then demonstrated that his creation did capture human computing.

Frank Zölner wrote in his book *Michelangelo Life and Work* (2010, p. 7)

> "As innovative as Leonardo da Vinci, who was a generation older, as productive as his slightly younger contemporary Raphael of Urbino, as secretive as Giorgione in Venice and blessed like Titian with a long life and unbridled creativity, Michelangelo Buonarroti embodies, perhaps most completely, the concept of the artist in the modern era."

Likewise, Alan Turing embodies, perhaps most completely, the concept of human computability in the modern era. Regarding the creative process, Turing 1939, Sec. 11 wrote,

> "Mathematical reasoning may be regarded rather schematically as the exercise of a combination of two faculties, which we may call *intuition* and *ingenuity*. The activity of the intuition consists in making spontaneous judgments which are not the result of conscious trains of reasoning. These judgments are often but by no means invariably correct. . . .
> The exercise of ingenuity in mathematics consists in aiding the intuition through suitable arrangements of propositions, and perhaps geometrical figures or drawings."

Turing's first great contribution was by intuition. Although others were studying deductive systems like HG recursion, Turing's intuition drew him to a completely new model more clearly

reflecting in mechanical terms how a calculation is carried out. His second contribution was his exercise of ingenuity which led him to a demonstration that anything computed by a human being could be computed by a Turing machine. Gandy (1988, p. 82) observed, 'Turing's analysis does much more than provide an argument for' Turing's Thesis, '*it proves a theorem*.' Furthermore, as Gandy (1988, pp. 83–84) pointed out, 'Turing's analysis makes no reference whatsoever to calculating machines. Turing machines appear as a result, a codification, of his analysis of calculations by humans.' Sieg (1994, 2006, 2009), gives a full analysis of Turing's contribution. Wittgenstein remarked about Turing machines, 'These machines are *humans* who calculate'.

5. Classical art and classical computability

The term 'recursive' to mean 'computable' began with Church (1936) and was developed by Kleene to prevail for 60 years in the form, 'recursive function,' 'recursively enumerable set' and 'recursive function theory'. In June 1979, Gerald Sacks and I had each been invited to give a series of four lectures at the Italian Mathematical Summer Center (C.I.M.E.). Gerald was to lecture on *generalised recursion theory (GRT)* and I on recursion theory on the integers ω, at that time sometimes called *ordinary recursion theory (ORT)*. I began the trip with a few days in Florence revisiting the Renaissance art treasures of the Uffizi gallery and Michelangelo's statues in the Accademia. As the train made its way to Bressanone at the very north of Italy in the Dolomite alps, I was still basking in the memories of the art of the Renaissance.

As our courses began in Bressanone, Gerald kept repeating the term 'ordinary recursion theory (ORT)'. He was doing nothing wrong, simply using the term as it had come to be used in the previous decade to distinguish ω-recursion theory from GRT. And yet as the phrase kept cascading down it clashed more and more with my esthetic sense. It seemed far too impoverished to describe the magnificent theory created by Turing(1936, 1939), Post (1944) and the others. My colleagues at the University of Chicago, Alberto Calderone and Antoni Zygmund, worked in singular integrals and classical analysis, but classical analysis was never called 'ordinary analysis' to distinguish it from functional analysis. No one ever called the art of the Renaissance 'ordinary art' to distinguish it from Baroque art or Impressionism.

By my third lecture it all came together. I coined the term *'classical recursion theory (CRT)'* and developed a whole lecture about the analogies between CRT and the classical art of the Italian High Renaissance. The lectures and art analogies were published in the works done by Soare (1981), but it is a rather obscure reference and not widely read. The lectures were expanded at the Cornell AMS meeting in July 1982, but not published there. Some of the analogous characteristics are these.

5.1. *Human scale*

A Roman arch such as the Arch of Constantine next to the Colosseum in Rome is designed to arouse awe and to dwarf the human figure. In contrast the Loggia della Signoria in Florence is on a human scale and designed to display statues of human size. The art and sculpture of the High Renaissance were designed to display the human form. Analogously, the computability theory of Turing and Post works on the integers, which can be represented as in Turing by a finite sequence of ones and blanks. GRT works on infinite ordinals or on functionals of higher type.

5.2. *Composition and balance*

The paintings of the Renaissance were characterised by highly complex compositions which were balanced to keep the eye from leaving the painting. Leonardo's *The Virgin and Saint Anne* has a very complicated and carefully designed composition around the three figures, Mary, her mother, Anne and her son Jesus. The heads and feet form one large triangle. The arms and child form an inner rectangle. Everything holds the eye and prevents it from leaving the painting as it might in a Baroque painting. In classical computability, theorems such as the Friedberg Muchnik theorem are proved by

a delicate balance of opposing requirements, positive requirements wanting to enumerate elements into a set A and negative requirements wanting to keep elements out. These constructions are often defined with as much intricacy and balance as a Renaissance composition. Other characteristics will be developed in later papers.

6. Computability and recursion

Church (1936) and Kleene began to use the term 'recursive' to mean 'computable' as well as 'inductively defined'. Since Kleene was the main figure in the subject after 1940, this term had became standard for 60 years from 1936 to 1996. By the 1990s this usage had become problematic. When one referred to a recursive function did one mean 'inductively defined' or 'effectively computable?' Also, the term 'recursive' was not well understood in the mathematical and scientific community and, if understood at all, it was identified with the elementary methods of iteration and recursion in a first programming course. Neither Turing nor Gödel ever used the word 'recursive' to mean 'computable' or 'recursive function theory' to name the subject. When others did, Gödel reacted sharply negatively stating, 'the term in question [recursive] should be used with reference to the kind of work Rosza Peter did'.

Soare (1996, 1999a) analysed the history and meaning of computability and recursion and suggested that the terms 'Computability Theory' and 'computably enumerable set' be used in place of the recursive version. This was largely adopted within a few years.

7. The art of exposition

In the art of exposition it is not sufficient to have a correct theorem with a correct proof. It must be the right theorem with the right proof, relating the results which came before and those which will come after in an aesthetically pleasing mix. The entire work must be artistically beautiful and must appeal to the imagination.

The initial expositions in the study by Turing (1936) and Post (1944) were clear, intuitive and very well motivated. In contrast, Kleene (1936) had developed the Kleene T-predicate as a Gödel coding of the Herbrand–Kleene recursive functions, which had little appeal to the imagination. Kleene's mathematical results were very difficult but his T-predicate notation was hard to read. It dominated the proofs in the subject for over 30 years. For example, Friedberg (1957a) used the Kleene T-predicate style proofs in his solution to Post's problem and his completeness criterion (Friedberg, 1957a), which made the proofs difficult to read. Compare these proofs with the informal style of Rogers' (1967) book's written in a clear and intuitive style, which opened the subject to a generation of students and which was continued in Soare's (1987) book.

8. Conclusion

Mathematicians are not assigned projects like building bridges. Like artists, they choose which problems to work on according to taste and beauty. Like artists, what they produce is evaluated on the basis of beauty as well as mathematical results. The greatest results are those arising from a completely new vision and a profound intuition into the area.

References

Church A., 1936. An unsolvable problem of elementary number theory. Am. J. Math., 58, 345–363.
Feferman S., 2007. Turing's Thesis. Notices of the Amer. Math. Soc. 53, 1200–1206.
Friedberg, R.M., 1957. Two recursively enumerable sets of incomparable degrees of unsolvability. Proc. Natl. Acad. Sci. USA 43, 236–238.

Friedberg, R.M., 1957. A criterion for completeness of degrees of unsolvability. J. Symb. Log. 22, 159–160.
Friedberg, R.M., 1958. Three theorems on recursive enumeration: I. decomposition, II. maximal set, III. enumeration without duplication. J. Symb. Log. 23, 309–316.
Gandy, R., 1988. The confluence of ideas in 1936. Herken, 55–111.
Gödel K., 1931. Über formal unentscheidbare sätze der Principia Mathematica und verwandter systeme. I. Monatsch. Math. Phys. 38, 173–178.
Gödel K., 1934. On undecidable propositions of formal mathematical systems. Notes by S.C. Kleene and J.B. Rosser on lectures at the Institute for Advanced Study, Princeton, New Jersey, 1934, p. 30.
Gödel K., 1995. Collected works vol. III: unpublished essays and lectures, Feferman, S., et. al. (Eds.), Oxford Univ. Press, Oxford.
Kleene, S.C., 1936. General recursive functions of natural numbers. Math. Ann. 112, 727–742.
Post, E.L., 1944. Recursively enumerable sets of positive integers and their decision problems. Bull. Am. Math. Soc. 50, 284–316.
Rogers, Jr., H., 1967. Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York.
Sieg, W., 1994. Mechanical procedures and mathematical experience. In: George, A. (Ed.), Mathematics and Mind, Oxford Univ. Press, Oxford, , pp. 71–117
Sieg, W., 1997. Step by recursive step: Church's analysis of effective calculability. Bull. Symb. Log. 3, 154–180.
Sieg, W., 2006. Gödel on computability. Philosophia Mathematica 14, 189–207.
Sieg, W., 2009. On computability. In: Irvine, A.D. (Ed.), Handbook of the Philosophy of Mathematics, Elsevier, pp. 535–630.
Soare, R.I., 1981. Constructions in the recursively enumerable degrees. In: Lolli, G. (Ed.), Recursion Theory and Computational Complexity, Proceedings of Centro Internazionale Matematico Estivo (C.I.M.E.), June 14–23, 1979, in Bressanone, Italy, Liguori Editore, Naples, Italy.
Soare, R.I., 1987. Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets, Springer-Verlag, Heidelberg.
Soare, R. I., 1996. Computability and Recursion. Bull. Symb. Log. 2, 284–321.
Soare, R.I., 1999. The history and concept of computability. In: Griffor, E.R. (Ed.), Handbook of Computability Theory, North-Holland, Amsterdam, pp. 3–36.
Soare, R.I., 2007. Computability and Incomputability, Computation and Logic in the Real World. In: Cooper, S.B., Löwe, B., Sorbi, Andrea (Eds.), Proceedings of the Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18–23, 2007, Lecture Notes in Computer Science, No. 4497, Springer-Verlag, Berlin, Heidelberg.
Soare, R.I., 2009. Turing oracle machines, online computing, and three displacements in computability theory. Ann. Pure Appl. Log. 160, 368–399.
Soare, R.I., 2012. Formalism and intuition in computability. In: Abramsky, S., Cooper, S.B. (Eds.), Philosophical Transactions of the Royal Society A. 370, 3277–3304.
Soare, R.I., in press a. Computability Theory and Applications: The Art of Classical Computability, Computability in Europe Series, Springer-Verlag, Heidelberg.
Soare, R.I., in press b. Turing-post relativized computability and interactive computing. In: Copeland, J., Posy, C., Shagrir, O. (Eds.), Computability: Gödel, Church, Turing, and Beyond, MIT Press.
Soare, R.I., in press c. The impact of Turing on computable reducibility and information content. In: Downey, R. (Ed.), Association for Symbolic Logic Lecture Notes in Logic (CUP).
Turing, A.M., 1936.[1] Turing, A.M., 1936. On computable numbers, with an application to the entscheidungsproblem. Proc. Lond. Math. Soc. 2 (42), 230–265. "Correction", 43 (1937), 544–546.
Turing, A.M., 1965. Systems of logic based on ordinals. Proc. Lond. Math. Soc. 45 Part 3, 161–228. Reprinted in Davis [1965], 154–222.
Zöllner, F., Thoenes, C., Taschen, B. (Eds.), 2010. Michaelangelo Life and Work, Taschen GmbH, Köln, 2010. ISBN: 978-3-8365-2117-8

[1] Many papers, Kleene (1943, p. 73; 1987a;1987b), Davis (1965, p. 72), Post (1943, p. 200) and others, mistakenly refer to this paper as 'Turing (1937)'. The journal states that Turing's manuscript was 'received on May 28 1936 and read on November 12 1936'. It appeared in two sections, the first section of pages 230–240 in volume **42**, part 3, issued on November 30 1936, and the second section of pages 241–265 in volume **42**, part 4, issued on December 23 1936. No part of Turing's paper appeared in 1937, but the two page minor correction (1937a) did.

Rainer Glaschick takes us on a trip back to —

TURING MACHINES IN MÜNSTER

In the University of Münster (Germany) existed a room with Alan Turing's name, the *Turingraum*. It contained documents and, as tangible objects, small machines made from post office relays, designed and built by Gisbert Hasenjaeger and Dieter Rödding. While the *Turingraum* does no longer exist, a number of the artefacts have been kept by the Hasenjaeger and Rödding families, and now made accessible for analysis and maybe reconstruction.

One of these artefacts is a universal Turing machine with 16 relays, which has 4 states, 2 symbols, and 3 non-erasable tapes, apparently smaller than any machine of this type known so far. Also, it needs only 15 bits to encode a program to add a mark at the end of a chain of marks on the result tape.

1. Introduction

The *Institut für mathematische Logik und Grundlagenforschung* of the University of Münster in Germany had a *Turingraum*, dedicated to Alan Turing's work and the Turing machine in particular, initiated and run from 1960 until 1985 by Gisbert Hasenjaeger and Dieter Rödding.

The Institute in Münster was the first and for a longtime leading one in the area of mathematical logic in Germany. It was established in 1936 by Heinrich Scholz, followed in 1953 by Hans Hermes (until 1966) and Dieter Rödding (until 1985). H. Scholz was one of the two persons who asked Alan Turing for a reprint of his "On computable numbers", which, including a dedication, is still in the archives of the University. H. Hermes is well known for his books on computability and logic, where he established a prominent role for Turing machines. H. Hermes was also the first one who proved under the title *Die Universalität programmgesteuerter Rechenmaschinen* (Hermes, 1954) that an idealised programmable electronic computer could be programmed to duplicate the behaviour of any Turing machine.

The *Turingraum* contained a collection of physical machines that were either Turing machines or register machines.[1] The devices in the *Turingraum* were conceived and built by Gisbert Hasenjaeger and Dieter Rödding.

Unfortunately, the *Turingraum* does no longer exist, and no specific documents could be found any more. Fortunately, Irmhild Hasenjaeger and Walburga Rödding have preserved several of those objects. Norbert Ryska[2] found this out and convinced the families to entrust us the objects for further study, which we thankfully acknowledge here.[3]

2. The object

The device selected for in-depth analysis has a central box and three peripherals, one peripheral obviously being a Turing tape, see Fig. 1. This object was shown in Oxford during 2012 and then in Paderborn in the special exhibition on Alan Turing in the Heinz Nixdorf Museum.

As E. Börger wrote in his obituary for Dieter Rödding (Börger, 1987), a universal Turing machine was built between 1958 and 1960, which is probably the machine covered here. He also

[1] They are also called counter machines.

[2] The director of the Heinz Nixdorf Museum.

[3] Special thanks to E. Börger for his continuous support.

reported from his own experience, confirmed by others, that machines of this kind were shown and operated in lectures on computability.

The machine is shown in Fig. 1. It seemed to be a fairly complete and attractive object for investigation. The detailed structure could be reconstructed now.

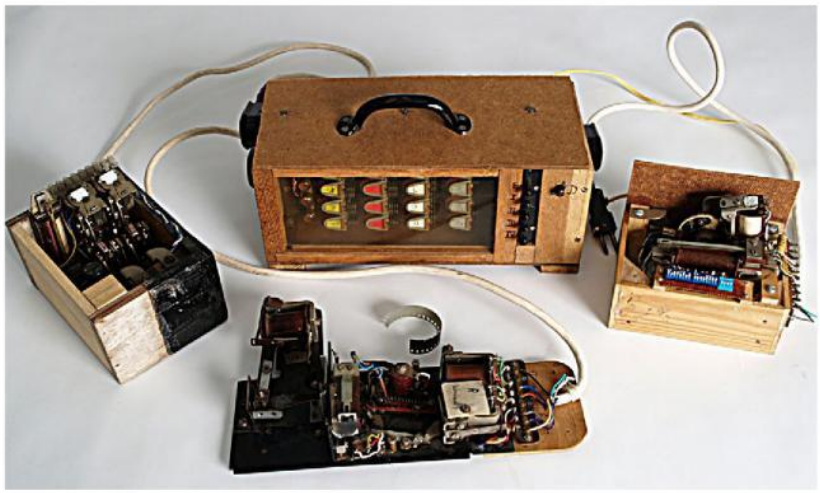

Fig. 1: Hasenjaeger's machine.

2.1. *Hardware layout*

The machine has a main cabinet, with 16 relays, 3 connectors, a switch, 5 buttons and 4 lamps. Three different kinds of 'tapes' can be plugged to the main box:

- Tape P, the program tape. It is a circular tape with 18 positions, that can be set by small switches. Using a selector switch, it can be 'moved' in only one direction.
- Tape Q, the counter tape. Two selector switches are connected back-to-back, so that a signal is generated iff both have the same position. It is thus a counter (modulo 18), that can be incremented and decremented, and zero sensed. In terms of Turing tapes, it is an immutable tape with a marker every 18th position.
- Tape R, the result tape.[4] It can be moved in both directions, but once marked, the mark cannot be erased. Used was 35-mm film, split longitudinally, so that one border has a perforation. The marks were punched as triangular notches at the opposite border. A lever sensed the marks, and was raised by a magnet automatically whenever the tape was moved or punched.

The main cabinet has a fixed logic, where 4 of the 16 relays provide a two-phase clock, and the remaining 12 relays are pairwise connected to provide 6 flipflops. Two pairs of flipflops were connected in a master-slave mode, so that during one phase of the clock the information was transferred from the master to the slave. In the other phase, the state table is evaluated using contact trees, either moving (or marking) the tapes, and sets the master flipflops for state transitions.

Thus, the machine has a two-bit state memory; the states labelled with roman numerals I, II, III and IV. Four lamps show the current state, and four of the push buttons allow to directly set a state. The other push button allows single-step advance, and the switch allows to run in continuous mode.

[4] Labelled *Rechenband*, i.e., calculating or computing tape.

The reconstructed state table is shown in A. Slightly different from what Hasenjaeger wrote (Hasenjaeger, 1987), the encoding is: `1`=*, `01`=R, `001`=L, `000n1`, where *n* stands for *n* zeroes and * for mark. `n` is the the equivalent of the conditional transfer, that Wang (1957) denoted *Cn*, but in our case it is a relative jump address, i.e., a skip of *n* instructions. As all instructions end in a mark, the state machine skips marks on tape P, if tape R is marked; otherwise the *n* zeroes just read are discarded. Because tape P is cyclic, a long-enough skip goes back, resulting in a program loop. Because all instructions except mark have zeroes, forward jumps are possible.

2.2. *Examples*

The first example uses the bit pattern found on the machine, but this is not necessarily authentic, as over the years someone may have flipped some switches just for fun.

The bit string `101101010000111100` found on the tape produces this sequence of operations:

```
* R * R R 1 **** L R * R R 1 **** ...
```

resulting in the sequence `**_*_*_*...` on tape R. Note that the first bit is interpreted only at the beginning as a mark instruction; because in the other rounds, it is the end bit of the `L` instruction. Also, the `1` is redundant on a blank tape, as the previous instruction has just reached a blank square.

Wang (1957) gives the following program to position onto the space at the end of a chain of marks: `1.* 2.R 3.C2 4.R 5.L`. The equivalent would be `* R 3 R L`, using $1+2+7+2+3=15$ bits, but staying busy because of Wang's convention using `RL` as stop or return.

Slightly modifying the machine (so that it stops if it tries to mark a marked tape), a program to append a mark at the end of a (possibly empty) chain of marks could be done with `2 * * R`, using only $6+1+1+2=10$ bits.

When I began the analysis, I did not believe it would be possible to build a UTM with 16 relays only, because according to the the very recent work of D. Woods and T. Neary (Neary, 2008; Woods and Neary, 2009), a (4,2) UTM (in this configuration) is not yet known.[5]

The compactness of this machine description clearly demonstrates the superiority of Hasenjaeger's and Rödding's concept to use Wang's programmatic method instead of the encoding of state tables.

3. The papers

Two papers have been published by G. Hasenjaeger, which contain substantial information about his and D. Rödding's work on Turing machines. These papers were published relatively late; Hasenjaeger left Münster to become professor in Bonn in 1962.

3.1. *Universal Turing machines and Jones–Matiyasevich-masking*

In the article with the above title, Hasenjaeger (1984) made several remarks that are related to our objects.

The article starts with a section *Background* as follows:

> When I learned from reports given by BÖRGER (spring 1982) and JONES (fall 1982) about a new combination of coding of sequences with coding of Boolean algebras as a tool to describe the behaviour of register machines, I tried to apply this tool on my earlier variants of small universal Turing machines hoping these application should lead to some sufficiently simple solutions for exponential diophantine predicates universal for r.e. sets.

[5] T. Neary also accounts for the efficiency of a UTM, and I think, the number of bits to encode a certain problem is important too.

This confirms that Hasenjaeger desired to make a small *universal* Turing machine.

Section 4 in his paper has the title *Universal Turing machines*, and its complete text is as follows:

> As decoding a number (introduced as an additional argument indicating a program or a particular machine) certainly needs more states, smaller solutions are obtained by introducing an additional program tape or program loop. Instead of targets for conditional jumps (or "gotos"), an additional register to count the "in between" for suspended operations seems adequate. As besides conditional halt and jump three operations on the tape are sufficient (l, r, print, or l, r, change sign) hence are to be transcribed, a similar code should also serve for a multiple counter concept: Just one counter is on duty; operations are: add 1, take 1, change of counters in a given cyclic order.

Not only does he write about a circular program tape, there is also an alternate use of tapes, like explained above for the tapes P and Q for the Hasenjaeger machine.

Note also the indication that state changes are coded as a distance to the next entry on the programme tape, accumulated to and then consumed from a counter tape, instead of absolute state numbers.

Section 5 has the title *From Turing tapes to counting registers* and starts as follows:

> As 25 years ago finding it harder to materialize a Turing tape with an operation: change symbol (instead of: print 1, not erasing) we introduced a multitape version: one 1 on each tape, and moving for counting. By changing the tape "on duty" in a cyclic order all tapes can be operated

It is not clear which machine was meant having tapes with a single mark and moving for counting, i.e., if the tapes using selector switches were included.

More important appears the idea not to use $3n$ different instructions for n counter tapes to enlarge, decrease and test, but instead 4 for enlarge, decrease, test and cyclic change. Whether this attempt really is advantageous, depends on the problem and code, because switching to a specific register may need up to $n-1$ change instructions.

3.2. *On the early history of register machines*

In 1987, G. Hasenjaeger published a short note with the above title (Hasenjaeger, 1987) that had the following footnote on the title page: *This report on my collaboration with D. Rödding is not restricted to the item on the title; but that item seems to be the most remarkable result of our joint activities.* Reading it in the context of the above device, this publication reveals important information.

The second part, with the heading *Turing Machines*, reveals a lot of information in relation to Alan Turing.[6] In the first paragraph, the Turing machines to be built are characterised as *theoretical*, i.e., not for practical use.

It is also mentioned that the people in Münster were not aware of the practical work of Alan Turing at that time, in particular the ACE.[7] In the next paragraph, Hasenjaeger mentions that Wang's article (Wang, 1957) had a high influence on the following activities to *materialise theoretical machines*.

[6] G. Hasenjaeger had worked for the German military and been *assigned the task of examining the security of Enigma. He detected weak points, but he could not foresee that the Allies had long since been taking advantage of these weaknesses* (Schmeh, 2009).

[7] Astonishing enough, that in 1987 Hasenjaeger mentioned the ACE, which is still unknown to many experts in this field.

Note the remark that the unreliablitiy of the tape punch lead to the search for alternatives, and, via Moore's practical proposal (Moore, 1952), to the idea of separate counting and programme tapes, and finally perhaps to Rödding's theoretical work on register machines and decomposition of automata. This clearly indicates that attempts to *materialise* theoretical constructs can lead to fruitful advances in theory.

The idea of alternatively scanning tapes, as mentioned here, has already been present in Hasenjaeger's machine, described in detail above. I originally suspected that the fact that tape Q could only be used in states II and IV, and tape R only in states I and III, was caused by a lack of relays and contacts, once the 16 relays were nicely assembled. Maybe we have the not so uncommon case that the desire to use only 16 relays lead to the idea of using tapes alternatively, which was later more extensively used.

Hasenjaeger mentioned that Rödding *already reported his results in H. HERMES' colloquium, when similar results of M. MINSKY [1961] became known.* And some sentences later: *I think we were angry enough not to go into MINSKY's details. Thus, I realised only much later that these details were quite different.* This is quite a pity, as his (4,2) UTM should have been published, not only for the small size, but also for the observation that using Wang's programme oriented encoding was also practically superior over the traditional attempt of encoding state tables.[8] E. Börger remembered (Börger, 1987) in this context that the unreliability of the first tape drive rose the desire to avoid punching, and became the abstract task to use a fixed, finite maximum of marks on a tape, leading Hasenjaeger and Rödding to the idea of using a tape with a single immutable marker as a counter.

Noting that using counters is a basic building block for Turing machines, D. Rödding followed this idea to use (infinite) counters to define computability, not only because this model made it easier to teach computability at elementary level. The final version used only two operations, increment and decrement, with just a (backwards) loop if the decremented register is not yet zero. He published a very clear and comprehensible description (Rödding, 1972) in German. Early publications on register machines are by Minsky (1961), and Shepherdson and Sturgis (1963), the latter citing the above mentioned proof of Hermes (1954) at the end of Section 1.

4. Conclusion

From the legacy of G. Hasenjaeger and D. Rödding, a small machine, made from old relays, was obtained, which was used to practically demonstrate Turing machines. Restricted to really small machines and due to difficulties in building tape drives, a machine was built with a mark-only result tape, a counter tape and a read-only program tape. Using this configuration, a (4,2) UTM was built that could encode a simple program in only 15 bits of the program tape. This was achieved not by encoding state tables, but by following H. Wang's theoretical proposal of using programs instead. The use of relative instead of absolute (preferably backward) jumps mirrored the transition to structured programming, and influenced D. Röddings final concept of register (or counter) machines.

5. State machine

The reconstructed state table is printed below. State numbers are in arabic digits, column 1 shows the state number, column 2 the conditions for the tapes P, Q and R (dot means do not care), column 3 shows the actions (dot for no action) and column 4 the next state (dot for no state change):

```
S PQR PQR S'
```

[8] In an early version of this paper, I started to argue that using state tables is more efficient, as it allows more actions to be done in parallel. While this is true for hardware logic, it is obviously is wrong for programs on Turing tapes.

```
        I: P=1 is punch, P=0 other instruction
1 1.0 +.* .   mark if not marked, next instruction 1 1.1 +.. . no
need to mark if marked, next instruction 1 0.. +.. 2   other
instruction, take the 0

        II: R, L or other; Q is zero on entry
2 10. +.R 1   go right, next instruction 2 00. ++. .   save 0 in
Q, check next P bit 2 11. +-L 1   next P bit is 1, go left, clear
Q, next inst. 2 01. +-. 3   next P bit is 0, this is a skip

        III: skip part 1: count zeroes to Q, if mark
3 0.0 +.. .   R has space, skip zeroes until P=1 3 0.1 ++. . R has
mark, count zeroes until P=1 3 1.0 +.. 1   end found; R has space:
next instruction 3 1.1 .+. 4   end found; R has mark, need to skip

        IV: skip part 2: execute
4 01. +.. .   while Q>0, skip zeroes on P, leave Q 4 11. +-. .
while Q>0, skip a one, decrement Q 4 .0. ... 1   Q=0, next
instruction
```

Remark:

Meanwhile, some of Hasenjaeger's notes and a bidirectional uniselector have been found, showing that the machine was built for a bidirectional tape P. Thus, in the state table for state 4 tape P is advanced backwards instead of forwards:

```
S PQR PQR S'
4 01. -.. . while Q>0, skip zeroes backwards on P, leave Q
4 11. --. . while Q>0, skip a one backward, decrement Q
4 .0. +.. 1 Q=0, end of backjump
```

The machine still never did jump conditionally, until today, when we have built a new bidirectional tape P from material left by Hasenjaeger.

References

Börger, E., 1987. D. Rödding: Ein Nachruf., Jber.d.dt.Math.-Verein. 89, 144–148. http://dml.math.uni-bielefeld.de/JB_DMV/JB_DMV_090_3.pdf

Hasenjaeger, G., 1984. Universal Turing machines (UTM) and Jones–Matiyasevich-Masking. In: Börger, E., Hasenjaeger, G., Rödding, D. (Eds.), Logic and Machines: Decision Problems and Complexity, vol. 171 of Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 248–253. http://dx.doi.org/10.1007/3-540-13331-3_44.

Hasenjaeger, G., 1987. On the early history of register machines. In: Börger, E. (Ed.), Computation Theory and Logic, vol. 270 of Lecture Notes in Computer Science Springer Berlin/Heidelberg, 181–188. http://dx.doi.org/10.1007/3-540-18170-9_165.

Hermes, H., 1954. Die Universalität programmgesteuerter Rechenmaschinen, Mathematisch-Physikalische Semesterberichte der Universität Göttingen 4, 42–53.

Minsky, M.L., 1961. Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. Ann. Math. 74 (2), 437–455, ISSN 0003-486X.

Moore, E.F., 1952. A simplified universal Turing machine. In: Proceedings of the 1952 ACM national meeting (Toronto), ACM '52, ACM, New York, NY, USA, pp. 50–55. http://doi.acm.org/ 10.1145/800259.808993.

Neary, T., 2008. Small universal Turing machines, Ph.D. thesis, NUI, Maynooth.

Rödding, D., 1972. Registermaschinen, Der Mathematikunterricht 18, 32–41, ISSN 0025-5807.

Schmeh, K., 2009. Enigmas contemporary witness: Gisbert hasenjaeger, cryptologia 33, 343–346, ISSN 0161-1194, doi:10.1080/01611190903186003.
Shepherdson, J.C., Sturgis, H.E., 1963. Computability of recursive functions. J. ACM 10, 217–255, ISSN 0004-5411. http://doi.acm.org/10.1145/321160.321170.
Turing, A.M., 1936. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. ser. 2. 24, 230–265.
Wang, H., 1957. A variant to Turing's theory of computing machines. J. ACM 4, 63–92, ISSN 0004-5411, http://doi.acm.org/10.1145/320856.320867.
Woods, D., Neary, T. 2009. The complexity of small universal Turing machines. Theor. Comput. Sci. 410, 443–450.

From K. Vela Velupillai —

Reflections on Wittgenstein's Debates with Turing during his *Lectures on the Foundations of Mathematics*[1]

Andrew Hodges (2008) recalled Max Newman's characterisation of Alan Turing as 'at heart more an applied than a pure mathematician', and went on (p. 4; italics added):

> "It might be more true to say that Turing had resisted this Cambridge classification from the outset. He attacked every kind of problem – from *arguing with Wittgenstein*, to the characteristics of electronic components, to the petals of a daisy."

This prompts me to return to Turing's 'debates' with Wittgenstein – now remembering Max Newman's characterisation – during the latter's *Lectures on the Foundations of Mathematics* (Wittgenstein, 1939) [1976]. It is little realised – indeed, to the best of this writer's knowledge, never mentioned – that when Turing attended these lectures, in the Lent and Easter terms of 1939, he was the young (Turing was not quite 27 years old and Wittgenstein turning a *vintage* 40!) author of *Systems of logic based on ordinals* (Turing, 1939) where 'ways in which systems of logic may be associated with constructive ordinals' (couched in the language of the λ-calculus) was a main theme. Feferman, in his perceptive *Preface* to '*Systems of Logic*' (Turing, 2001, p. 79) observed, correctly in my opinion: Turing never tried to develop an over-all philosophy of mathematics…' Yet, he (Turing) was engaging one of the great philosophers of the twentieth century on *his* (Wittgenstein's) interpretation – even 'deconstruction' – of Cantor's work on *Transfinite Numbers*!

It is a pity that in these famous lectures by Wittgenstein, Turing was 'set up' as the 'strawman' representing orthodox mathematics and mathematical logic, defending the conventional notion of consistency (not related to its specialised version in the Gödel–Rosser work) in mathematics. Had

[1] I write as an economist who is painfully aware that uninformed references to this dialogue between one of the great philosophers and the pioneer of computability recur in the methodological literature of economics (cf., e.g., the muddled invoking of a particular part of this famous dialogue by McCloskey (1991, pp. 13–4)).

the protagonists been privy to the Newman–Hodges picture of Alan Turing, who 'began (and ended) with the *physical world*' (Hodges, op.cit., p. 4), the subsequent misrepresentation of Wittgenstein's stance[2] may have been prevented.

The context for the particularly (in)famous part of the Wittgenstein–Turing dialogue on consistency/contradiction in mathematics (and mathematical logic), may well be the original few remarks in Wittgenstein (op.cit, pp. 211–212, Lecture XXII; italics added):

> It was suggested last time [i.e., Lecture XXI] that the danger with a contradiction in logic or mathematics is in the application. Turing suggested that a bridge might collapse.
>
> Now it does not sound quite right to say *that a bridge might fall down because of a contradiction* [in logic or mathematics]."

Now, to place this in proper historical perspective, compare Stanislaw Ulam's dialogue with Gian-Carlo Rota on collapsing bridges and logical contradictions (Rota, 1986, p. 2; italics added):

> "However, out of curiosity I [Rota] decided to play devil's advocate, and watch his reaction.
>
> But if what you [Ulam] say is right, what becomes of objectivity, an idea that is so definitively formalized by *mathematical logic* and by the theory of sets, on which you [Ulam] yourself have worked for many years of your youth?
>
> There was a visible emotion in his [Ulam's] answer. Really? What makes you [Rota] so sure that *mathematical logic corresponds to the way we think*?[3] You are suffering from what the French call a 'deformation professionelle.' *Look at the bridge over there. It was built following logical principles. Suppose that a contradiction were to be found in a set theory. Do you honestly believe that the bridge might fall down?*
>
> Do you [Ulam] then propose that we give up mathematical logic? said I [Rota], in fake amazement.
>
> Quite the opposite [said Ulam]. *Logic formalizes only very few of the processes by which we actually think*[4]. The time has come to enrich formal logic by adding to itsome other fundamental notions[5]."

In the years before Laurent Schwartz elegantly encapsulated the *Dirac delta function*[6] with his notion of generalised functions, von Neumann had 'banished' it from 'official' use in physics and

---

[2] Most egregiously represented by Charles Chihara (1977), only partially blunted by Shanker's brilliant counterattack in Shanker (1987).

[3] Brouwer had been there, and Wittgenstein may have remembered it, long before them, and had remarked, in his Inaugural Lecture of 1912 (Brouwer, 1913, p. 84; italics added), most perceptively:
"To the philosopher or to the anthropologist, but *not to the mathematician*, belongs the task of investigating why certain systems of symbolic logic rather than others may be effectively projected upon nature. *Not to the mathematician*, but to the psychologist, belongs the task of explaining why we believe in certain systems of symbolic logic and not in others, in particular *why we are averse to the so-called contradictory systems* in which the negative as well as the positive of certain propositions are valid."

[4] If, at this point, Ulam had added '*and act*', he would have completely encapsulated Wittgenstein's prescription for circumventing contradictions in mathematics and logic by means of '*rules*'.

[5] The original journal article has 'motions', but the context makes it clear that what is meant is 'notions'.

[6] Dirac himself attributed the origin of the idea to his 'early engineering training' (cf., Kragh, 1990, p. 41) – surely paralleling both Wittgenstein's early training as an aeronautical engineer and Turing's above characterisation by Newman and Hodges.

quantum mechanics for being mathematically 'improper'. Meanwhile, physicists, with princely unconcern for the prestigious embargo placed on the delta function, went on happily using it for calculations. Engineers, of course, were blissfully unaware of von Neumann's prestige or embargo and went on calculating with the Heaviside operational calculus.

So far as I know, neither the Feynman Integral, nor Bishop's constructivism, have been axiomatised. This has not prevented perfectly valid calculations using Feynman integrals in quantum electrodynamics. For all we know, there are, lurking in the inner recesses of the Platonic Universe, the eventually discoverable logical foundations, which will show that the use of the Feynman integral entails contradictions. No quantum physicist in his right mind would pay the slightest attention to such logical hair-splitting (cf., also Schwartz, 2001, Chapter VI).

I am suggesting, therefore, that a sympathetic reader (an always elusive creature), should approach this famous dialogue between a great philosopher of, among other things, mathematics, and a great logician and founding father of computability theory, remembering that both of these intellectual giants were also, fundamentally, wedded to the 'physical world' – one with an explicit engineering background and the other as an 'applied mathematician', both camouflaging as logicians and mathematicians perplexed by semantic paradoxes and grammatical nuances that they thought could be sorted out by dialogue.

References

Brouwer, L.E.J., 1913. Intuitionism and Formalism, Bull. Am. Math. Soc. 20 (2), 81–96.

Chihara, C., 1977. Wittgenstein's Analysis of the Paradoxes in His Lectures on the Foundations of Mathematics, Philos. Rev. 86 (3), 365–381.

Cooper, S.B., Löwe, B., Sorbi, A., 2008. New Computational Paradigms: Changing Conceptions of What is Computable, Springer, New York.

Feferman, S., 2001. Preface to: Systems of logic based on ordinals. In: Gandy, R.O., Yates, C.E.M. (Eds.), Collected Works of A.M. Turing – Mathematical Logic, North-Holland, Amsterdam.

Hodges, A., 2008. Alan Turing, logical and physical. In: Cooper, et al., 2008, pp. 3–15.

Kragh, H. S., 1990. Dirac: A Scientific Biography, Cambridge University Press, Cambridge.

McCloskey, D. N., 1991. Economic science: a search through the hyperspace of assumptions? Methodus 3 (1) 6–16.

Rota, G.-C., 1986. In memoriam of Stan Ulam: the barrier of meaning. Physica D, 2 (1–3), 1–3.

Schwartz, L., 2001. A Mathematician Grappling with His Century, Birkhäuser Verlag, Basel.

Shanker, S. G., 1987. Wittgenstein and the Turning-Point in the Philosophy of Mathematics, State University of New York Press, Albany, NY.

Turing, A.M., 1939. Systems of logic based on ordinals, Proc. Lond. Math. Soc., Series 2, 45, 161–228.

Turing, A.M., 2001. Collected Works of A.M. Turing: Mathematical Logic, Gandy, R.O., Yates, C.E.M. (Eds.), North-Holland, Amsterdam.

Wittgenstein, L., (1939) [1976]. Wittgenstein's Lectures on the Foundations of Mathematics – Cambridge 1939. In: Diamond, C. (Ed.), The University of Chicago Press, Chicago.

Jan van Leeuwen and Jiří Wiedermann on —

THE COMPUTATIONAL POWER OF TURING'S NON-TERMINATING CIRCULAR A-MACHINES

1. Introduction

For readers familiar with the concept of Turing machines as described in contemporary textbooks, reading the definition of a Turing machine in Turing's original paper (Turing, 1936) may present a surprise. It is not only the difference in notational style or in the vocabulary used when speaking about these machines (called *'automatic machines'*, or simply *'a-machines'*), which may be surprising. Astonishing may be the fact that properly designed a-machines never halt. a-Machines of this kind are called *circle-free* and their task is to output infinite sequences of binary digits representing computable real numbers $\in [0, 1]$. Obviously, for computing the infinite expansions of real numbers such a behaviour is perfectly desirable. Nevertheless, Turing noted that there may also be machines – so-called *circular machines* – which at some point stop producing output digits, i.e., they altogether produce only a finite number of output digits. This may happen in two different ways. Either the machine at hand reaches a configuration from which there is no possible further move, or the machine goes on moving without producing any further output digits.

The modern versions of circle-free a-machines are still being used as a formal model in so-called *computable analysis*, a field in which one studies the parts of real analysis and functional analysis that can be carried out in a computable manner (cf. Weihrauch (2000)).

The circular a-machines that terminate, i.e. that halt after performing a finite number of steps yield the basis of today's computability and complexity theory. In fact, they are the forerunner of the contemporary Turing machine model.

Non-terminating circular a-machines run forever but produce only a finite number of output symbols. It seems that no special attention has been paid to such machines. From a classical computational point of view, the machines are strange: in spite of the fact that their computation is infinite, they are doomed to produce but a finite number of outputs. Turing himself proved that the property of circularity of a-machines cannot be tested effectively by any other a-machine. What could such machines be good for?

Recently, we have investigated a new computational model of unbounded computational processes – so-called red-green Turing machines (van Leeuwen and Wiedermann, 2012).

The motivation for considering red-green Turing machines comes from the modern, typical computer applications in which the core mechanism is a multi-process system that is always up and running. Control goes from process to process and, whenever a process has its turn, the process computes until it executes an instruction that explicitly transfers control to another process. We have studied and appraised computationally the mechanism of control passing among the processes in the course of unbounded computation. The computing power of red-green Turing machines goes beyond that of classical Turing machines, reaching up to the second levels of the arithmetical hierarchy, viz. Δ_2 or even Σ_2.

We will show that red-green Turing machines can be seen as a modern variant of Turing's original circular non-terminating a-machines producing a finite number of output symbols. This

connection between the two models gives a new link between Turing's 1936 paper and contemporary computing. The connection straightforwardly leads to the characterisation of the computational power of the underlying a-machines and thus reveals an unexpected super-Turing computational potential of an authentic, 'old-fashioned' machine model of Turing.

The rest of this short note is organised as follows. In Section 2 we describe both models – circular non-terminating a-machines producing a finite number of output symbols and red-green Turing machines – in more detail. Next we prove their computational equivalence. In Section 3 we discuss the significance of our result. Some conclusions are given in Section 4.

2. a-Machines and red-green Turing machines

2.1. *a-Machines*

Using the contemporary terminology of Turing machines, an a-machine can be seen as a deterministic single tape Turing machine with working alphabet Σ and input and output alphabet $A = \{0, 1\}$, with $\Sigma \cap A \neq \emptyset$. In Turing's terminology, symbols from A are called 'figures' or *symbols of the first kind*, whereas the symbols from Σ are called *symbols of the second kind*.

The input – a string from $\{0, 1\}^*$ – is written on the tape at the beginning of the computation. The computation of an a-machine now proceeds as usual, reading, writing and rewriting the symbols on the tape and moving the head in accordance with the machine's program. At each time, the sequence of symbols from A printed on the tape (as a subsequence of all symbols printed by the machine) is called *the sequence computed by the machine* (cf. Turing (1936)), or simply the result at that time.

In Turing's own words (Turing, 1936): *"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."*

Thus, a circular a-machine is a machine which on a given input prints a finite number of output symbols (not necessarily into different cells), and then either halts, or goes on forever, performing an infinite number of steps in which no output symbol is printed anymore.

2.2. *Red-green Turing machines*

A red-green Turing machine is formally almost identical to the classical model of Turing machines. The only difference is that in red-green Turing machines the set of states is partitioned in two disjoint subsets: the set of green states, and the set of red states, respectively. There are no halting states. A computation of a red-green Turing machine proceeds as in the classical case, changing between green and red states in accordance with the transition function. A moment of change in state color is called a *mind change*. A formal language is said to be *recognised* just in case on the inputs of this language and precisely those, the machine computations 'stabilise' in green states, i.e., from a certain time on, the machine keeps entering only green states. Similarly, a language is said to be *accepted* if and only if the inputs from the language are recognised, and the computations on the inputs not belonging to this language eventually stabilise in red states.

The model captures in a neat way a main feature of the current thinking of computing: namely, viewing computations as potentially infinite sequences of communications between processes, oscillating between different states of mind but ultimately converging on a fixed behaviour.

2.3. *Relation between a-machines and red-green Turing machines*

For our purpose – comparing red-green machines with a-machines – we will only consider deterministic single tape red-green machines over input/output alphabet A computing partial functions $f : \{0,1\}^* \to \{0,1\}^*$. The result of computation will be defined similarly as in the case of a-machines. More specifically, if for input x $f(x)$ is defined, $f(x)$ will be written in binary as a finite sequence of symbols over A, as a subsequence of all symbols printed on the tape. If $f(x)$ is undefined then $f(x)$ is represented by an infinite sequence.

Theorem 2.1. *Let $\mathcal{A}$ be a circular a-machine, let $\mathcal{R}$ be a red-green Turing machine, and let $f : \{0,1\}^* \to \{0,1\}^*$ be a partial function. Then f is computed by $\mathcal{A}$ if and only if f is computed by $\mathcal{R}$ with $f(x)$ mind changes.*

Proof. (Sketch.) We show that, if f is computed by a non-terminating circular a-machine $\mathcal{A}$, then f can be computed by a red-green machine $\mathcal{R}$ performing $f(x)$ mind changes.

On input x, machine $\mathcal{R}$ works as follows. Each time $\mathcal{A}$ prints an output symbol, $\mathcal{R}$ prints the same symbol, and changes state to red and then to green. If $\mathcal{A}$ has no move from some configuration, then $\mathcal{R}$ switches to a red state and starts cycling in that state.

If $\mathcal{A}$ prints a finite number $f(x)$ of symbols without termination, then $\mathcal{R}$ accepts x in $f(x)$ mind changes and stabilises in a green state. If $\mathcal{A}$ keeps on producing an infinite number of output symbols, then $f(x)$ is undefined and obviously, $\mathcal{R}$ keeps changing its mind infinitely often as well.

Obviously, $\mathcal{R}$ computes f and, if $f(x)$ is a finite number, then $\mathcal{R}$ computes $f(x)$ in $f(x)$ mind changes.

Conversely, let $\mathcal{R}$ be a machine accepting x with $f(x)$ mind changes. Obviously, at the occasion of each mind change $\mathcal{R}$ can add one to the current number of mind changes and the total number presents $\mathcal{R}$'s current output (we assume that output symbols can be rewritten). Thus, after $f(x)$ mind changes the output from $\mathcal{R}$ represents the value of $f(x)$. Simulation of $\mathcal{R}$ by $\mathcal{A}$ is then a straightforward matter.

If x is not accepted by $\mathcal{R}$ then $\mathcal{R}$ oscillates forever between red and green states and thus $\mathcal{A}$ produces an infinite number of outputs, effectively rejecting the input x for which $f(x)$ is undefined. □

3. Significance of the result

By equating the computations of non-terminating circular a-machines with those of red-green Turing machines, we have opened a way of appraising the computational power of the former model via the latter. Results for this model are available.

The idea of the red-green computing goes back to the very notion of computability. This connection is strengthened by the present result that links red-green computing to the authentic ideas of Turing on computing. The original concept of computability, established in the middle of the twentieth century, has been established in the period of ideas on function calculation. But computational systems nowadays have a number of features that extend beyond pure function value calculation. The concept of red-green computations specifically addresses one important feature of contemporary usage of computers, viz. unbounded computations of multi-process systems. Several models of infinite computations have been studied in the past, as a natural generalisation of the classical notion of computations, without having any particular 'realistic' computational model of infinite computations in mind. Along these lines, without entering into details, let us mention Gold's notion of limiting recursion (Gold, 1965, 1967), and Putnam's related notion of trial-and-error predicates, inductive computing (Burgin, 1983), various kinds of ω-automata (Büchi, 1962; Rabin, 1969; Staiger, 1997; Thomas, 1990), tae-computing Hintikka and Mutanen

Table 1 Approaches to Unbounded Computation.

Model of Computation	Level	Reference	year
Non-term. circ. a-machines	Σ_2, Π_2	Turing (1936), *this paper*	1936
Number-theoretic predicates	Arithmetic sets	Kleene (1943)	1943
Oracle Turing machines	All sets	Turing (1939)	1939
Trial-and-error predicates	Δ_2	Putnam (1965)	1965
Limiting recursion	Δ_2	Gold (1965)	1965
Iterated limiting recursion	Δ_k	Schubert (1974)	1974
Alternating Turing machines	Arithmetic sets	Chandra et al. (1981)	1976
ω-Turing machines	NA	Cohen and Gold (1978)	1978
Inductive Turing machines	Arithmetic sets	Burgin (1983)	1983
Tae-computability	Σ_2	Hintikka and Mutanen (1988)	1988
Infinite time Turing machines	Hyper-arithmetic sets	Hamkins and Lewis (2000)	2000
Accelerating Turing machines	NA	Copeland (2002)	2002
Relativistic computing	Δ_2, Σ_2	Etesi and Nemeti (2002) and Wiedermann and van Leeuwen (2002)	
SAD computers	Arithmetic sets	Hogarth (2004)	2004
Zeno machines	NA	Potgieter (2006)	2006
Display Turing machines	Δ_3	Rovan and Steskal (2007, 2009)	2007
Red-green Turing machines	Σ_2, Π_2	van Leeuwen and Wiedermann (2012)	2012

(1988), and display Turing machines with control (Rovan and Steskal, 2007, 2009) . Also, so-called hypercomputers have been inspired by relativistic physics, cf. Hogarth (2004), Welch (2008), Wiedermann and van Leeuwen (2002). At the heart of all these alternative approaches to unbounded computations, the complexity classes of the arithmetical hierarchy have repeatedly emerged as the classes characterising the computational power of the underlying models, in particular the classes Δ_2 and Σ_2. An overview on various approaches to unbounded computations is given in Table 1.

In preliminary studies we have investigated various aspects of red-green computations from the viewpoint of the computability theory. For example, it appears (van Leeuwen and Wiedermann, 2012) that the computational power of red-green Turing machines increases with the number of mind changes allowed (it climbs along the so-called Ershov hierarchy, cf. Cooper (2004), Ershov (1968) and Rogers (1967)). Also, for any finite number of mind changes red-green Turing machines recognise languages in Σ_2 and accept languages from Δ_2. In fact, computations of red-green Turing machines exactly characterise the latter two classes. This, together with the similar results achieved with the help of other machine or logical models of unbounded computation mentioned in the beginning of this section, suggests that, due to their simplicity and mathematical elegance, red-green Turing machines can serve as a bridging model among the various alternative models of potentially infinite computations. Moreover, another interesting result is that red-green Turing machines can elegantly and straightforwardly be simulated by relativistic Turing machines (Etesi and Nemeti, 2002; Wiedermann and van Leeuwen, 2002); (and vice versa). This indicates the relation of red-green computing to hypercomputing. An overview of the known results on red-green Turing machines can be found in van Leeuwen and Wiedermann (2012). The first results are encouraging and it is nice to see, retrospectively, that the core ideas essentially have their roots in the Turing's work.

4. Conclusion

It is symptomatic that the hallmarks of modern computing have appeared in the mathematical work of Turing whose primary aim initially was to define the notion of computability rather than to lay down the theoretical fundamentals of computing machinery. (Turing himself was among the first to realise the impact of the latter later on.) In particular, Turing's a-machines were tailored to infinite rather than finite computations. This, together with the prevailing use of computers nowadays, has opened a way towards considering problems up to Δ_2 or even Σ_2 computable by unbounded processes, as we have tried to show in this note. These considerations may change our attitude towards what is computable. Would this trend lead to an extension of the notion of computability?

Acknowledgements

This research was partially supported by Czech National Science Foundation Grant No. P202/10/1333 and institutional research plan AV0Z10300504.

References

Büchi, J.R., 1962. On a decision method in restricted second order arithmetic. In: Logic, Methodology and Philosophy of Science, Proceedings of the 1960 International Congress, Nagel, E., Suppes, P., Tarski, A., Stanford University Press, Stanford, pp. 1–11.

Burgin, M., 1983. Inductive Turing Machines, Notices Acad Sci USSR, 270:6 1289–1293 (translated from Russian, v. 27, No. 3).

Burgin, M., 2005. Super-recursive Algorithms, Springer, Berlin, Heidelberg.

Chandra, A.K., Kozen, D.C., Stockmeyer, L.J., 1981. Alternation, J. ACM 28 (1) 114–133.

Cohen, R.S., Gold, A.Y., 1978. ω-Computations on Turing machines, Theor. Comput. Sci. 6 (1), 1–23.

Copeland, B.J., 2002. Accelerating turing machines. Minds Machines 12, 281–301.

Cooper, S.B., 2004. Computability Theory, Chapman & Hall/CRC, Boca Raton, London, New York.

Ershov, Y.L., 1968. A certain hierarchy of sets I. Algebra i Logika 7 (1) 47–74 (Russian), *Algebra and Logic* 7:1 (1968) 25–43 (English translation).

Etesi, G., Nemeti, I., 2002. Non-Turing computation via Malament-Hogarth space-times. Int. J. Theor. Phys. 41 (2), 341–370.

Gold, E.M., 1965. Limiting recursion. J. Symb. Log. 30 (1) 28–48.

Gold, E.M., 1967. Language identification in the limit. Inf. Control 10 (5) 447–474.

Hamkins, J.D., 2000. A. lewis, infinite time turing machines. J. Symb. Log. 65 (2) 567–604.

Hintikka, J. Mutanen, A., 1988. An alternative concept of computability. In: Hintikka, J. (Ed.), Language, Truth, and Logic in Mathematics, Kluwer, Dordrecht, pp. 174–188.

Hogarth, M., 2004. Deciding arithmetic using SAD computers. British J. Phil. Soc. 55, 681–691.

Kleene, S.C., 1943. Recursive predicates and quantifiers. Trans. Amer. Math. Soc. 53 (1) 41–73.

Potgieter, P.H., 2006. Zeno machines and hypercomputation. Theor. Comput. Sci. 358, 26–33.

Putnam, H., 1965. Trial and error predicates and the solution to a problem of Mostowski. J. Symb. Log. 30 (1) 49–57.

Rabin, M.O., 1969. Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. 141, 1–35.

Rogers, Jr., H., 1967. Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York.

Rovan, B., Steskal, L., 2007. Infinite computations and a hierarchy in Δ_3. In: Cooper, S.B., Löwe, B., Sorbi, A. (Eds.), Computability in Europe, Lecture Notes in Computer Science, vol. 4497, Springer, Berlin, pp. 660–669.

Rovan, B., Steskal, L., 2009. Infinite computations and a hierarchy in Δ_3 reconsidered. J. Log. Comput. 19 (1), 175–176.

Schubert, L.K., 1974. Iterated limiting recursion and the program minimization problem. J.ACM 21 (3), 436–445.

Staiger, L., 1997. ω-Languages. In: Rozenberg, G., Salomaa, A. (Eds.), Handbook of Formal Languages, Vol 3: Beyond Words, Springer, Berlin pp. 339–387.
Thomas, W., 1990. Automata on infinite objects. In: van Leeuwen, J. (Ed.), Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier, Amsterdam, pp. 133–192.
Turing, A.M., 1936. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. Ser. 2 (42) 230–265.
Turing, A.M., 1939. Systems of logic based on ordinals. Proc. Lond. Math. Soc. Ser. 2 (45) 161–228.
van Leeuwen, J., Wiedermann, J., 2012. Computation as an unbounded process. Theoretical Computer Science 429, 202–212 .
Weihrauch, K., 2000. Computable Analysis: An Introduction, Springer-Verlag, Berlin/Heidelberg.
Welch, P.D., 2008. The extent of computations in Malament-Hogarth spacetimes. Br. J. Phil. Sci. 59 (4), 659–674.
Wiedermann, J., van Leeuwen, J., 2002. Relativistic computers and non-uniform complexity theory. In: Calude, C.S., Dinneen, M.J., Peper, F. (eds.), Unconventional Models of Computation (UMC'2002), Lecture Notes in Computer Science, Vol. 2509, Springer, Berlin, pp. 287–299.

Meurig Beynon puts an empirical slant on —

TURING'S APPROACH TO MODELLING STATES OF MIND

In discussing Turing's seminal 1936 paper, the *Stanford Encyclopedia of Philosophy,* http://plato.stanford.edu/entries/turing/, highlights the way in which his conception of the 'Turing Machine (TM)' was guided by the idea of *modelling states of mind*: '… in a bolder argument, the one he placed first, he considered an "intuitive" argument appealing to the states of mind of a human computer (Turing, 1936, p. 249). The entry of "mind" into his argument was highly significant, but at this stage it was only a mind following a rule.'

There are two respects in which the subsequent treatment of TMs in mainstream computer science has sidelined 'intuitive' elements:

- The TM is viewed as the fundamental mathematical abstraction in a theory of computer science that is based on 'computational thinking'.
- Computer science has promoted the computational theory of mind, which proposes that everything the human mind does is attributable to *following rules*.

In combination, these two viewpoints promote a narrow conception of computer science; they respectively root computing in an abstract machine model with a strict formal semantics and set out to show that such a model is enough to give a good account of its core applications.

Throughout the short history of computer science, it has suited the political purposes of an emerging discipline to emphasise its connections with Turing's work – one of the supreme intellectual achievements of the twentieth century. But perhaps it is not so clear that computer science as currently conceived truly fulfils Turing's aspirations for a science of computing.

As Hodges (2004) points out, Turing's vision was broader than that of a mathematical logician: 'The essence of Turing's achievement was the discovery of a concept *with an application to* logic, rooted in ideas which lay outside mathematics' and, unlike 'Church's Thesis', 'Turing's definition

[of computability] was modeled on what human beings could actually do'. Also, as Hodges (2004) goes on to observe: '[After 1948] Turing did … surprisingly little … to build up the modern science of computation'.

As for Turing's stance on the contribution that computational abstractions make to our understanding of mind, there are many clues in his discussion of 'The Imitation Game' in Turing (1950). It is clear that Turing sees the isssue of whether an interrogator might be deceived into attributing a machine's responses to a human as a *limited* question, and not one that decisively illuminates the nature of mind. For instance, he acknowledges that other concerns about the mind might be beyond the scope of his enquiry: 'I don't wish to give the impression that I think there is no mystery about consciousness … [b]ut I do not think these mysteries necessarily need be solved before we can answer the question with which we are concerned in this paper'. Even in relation to his contention that a computer could succeed in 'The Imitation Game', Turing (1950) is careful to point out that – despite the rebuttals he gives in response to objections to this possibility: 'The reader will have anticipated that I have no very convincing arguments of a positive nature to support my views'.

Hodges (1988) refers elsewhere to the 'trenchant materialist and atheist Turing who emerged after 1936', and it is perhaps these characteristics that have been most emphasised in subsequent building upon his legacy. In that context, it may seem surprising that, in rebutting counter-arguments to the idea that a computer might successfully play 'The Imitation Game', Turing (1950) remarks: 'The Argument from Extrasensory Perception] is to my mind quite a strong one'. But this is more explicable if, as Hodges (1988) advocates, 'we recognize [in Turing] the common thread – a great seriousness about the sheer mystery of mental phenomena, and an equally serious conviction that they must be reconciled with a scientific world view'.

As the *Stanford Encyclopedia of Philosophy* quotation above emphasises, Turing's treatment of states of mind was conceived with a view to modelling 'a mind following rules', as was appropriate for addressing the *Entscheidungsproblem*: a Turing machine should model procedures 'definite in the sense that at every stage a completely explicit "note of instructions" could be written down explaining what was to be done in such a way that another person could take up the work' (Hodges, 1988). Beyond question, Turing's insight informed the practical contributions he made to the development of computer programming. For instance, it gives a deep meaning – deeper than a programmer needed to appreciate – to the statement with which he prefaced his *General Remarks on Electronic Computers* in his Manual for the Ferranti Mk. I computer Turing (1951): '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'. It also led him to identify the importance in programming of more disciplined use of mathematical notation (Turing, 1944–5).

But – to take nothing away from the extraordinary fertile nature of Turing's insight, and what has been, and can yet be, achieved within the framework of computational models in many disciplines – it seems likely that Turing himself did not consider the broader issue of the nature of mind to be closed. To quote Hodges (1988) once again: '… we cannot feel that Turing had arrived at a complete theory of what he meant by modelling the mental functions of the brain by a logical machine structure'. Significantly, in the spirit of an applied mathematician, Turing gave high priority to squaring mathematical theory with empirical evidence and practical applicability. In this connection, Hodges (2004) contrasts Turing's approach to enhancing discrete computational models as models of physical systems through introducing randomness with 'that of some modern theorists, who seek to outdo discrete computation by exploiting the very elements that Turing made little of' and whose approaches lead to difficulties that render them 'meaningless without stability and robustness in the face of infinitesimal phenomena'. As further testimony to Turing's practical orientation, we need only consider the degree of direct involvement he wanted in building

early computers, his contributions to code-breaking and speech recognition, and his concern to take account of the inevitability of operational error in theorising about computers (Hodges, 2004).

It may be that, by imputing greater authority to mechanical models of mind than Turing himself would have decisively endorsed, today's students of computer science are in danger of identifying mathematics with a blind process of inference such as can be carried out by a machine. Such a concern is raised by Byers (2007), for instance, in motivating his account of mathematical practice. Turing's biographers describe him as a mathematician '[whose] native style was rough-and-ready and prone to minor errors' (Feferman, 2006) for whom the notion of 'intuition' had a particular fascination (cf. his mathematical work on the Riemann hypothesis (Hodges, 1988)). If this seems to be at odds with the demystification of effective procedures that he achieved in his own work, an instructive parallel may be drawn with his older contemporary Emil Post, who concluded his paper 'Absolutely unsolvable problems and relatively undecidable propositions' (written in 1941, but only published posthumously (Post, 1965)) by expressing his amazement at the reaction to Godel's undecidability results in the following terms:

> "... mathematical thinking is, and must be, essentially creative. It is to the writer's continuing amazement that ten years after Gödel's remarkable achievement current views on the nature of mathematics are thereby affected only to the point of seeing the need of many formal systems, instead of a universal one. Rather has it seemed to us to be inevitable that these developments will result in a reversal of the entire axiomatic trend of the late nineteenth and early twentieth centuries, with a return to meaning and truth."

To the end of his life – in the spirit of Post's injunction – Turing seems to have been motivated to seek significance beyond an abstract logical interpretation for his TM concept. In realising his vision for computation, he felt the essential need to establish the connection with physical reality. Hodges (1988) cites Penrose's summary of Turing's position in 1950: 'It seems likely that he viewed physical action in general – which would include action of a human brain – to be always reducible to some kind of Turing-machine action'. As Hodges (1988) later goes on to relate, this was to be an unresolved problem for Turing, who recognised the challenge presented by the indeterminacy principle in quantum mechanics.

It is hard to imagine how the academic discipline of computer science would have emerged without Turing's contribution. In his work, Turing showed extraordinary prescience in relation to many aspects of the 'computer programming' related activity that has been the central focus of academic computer science throughout its history. But, at the time of his death, the transformative impact of computers and programming could hardly have been predicted. And in the same way that mathematics demands a broader account than formal systems can supply, so too does contemporary computing. In concluding this brief review, it is appropriate to look at ways in which modern computing and the science of computing to which we must now aspire is influenced by other perspectives on modelling human states of mind. This conclusion reflects the author's own research interest, under the auspices of the Empirical Modelling (EM) project http://www.dcs.warwick.ac.uk/modelling, in seeking a broader alternative conceptual framework for computing.

Turing (1950) asserts that the problem of developing a digital computer that can succeed at The Imitation Game 'is mainly one of programming'. In the context of modern software development, it has become apparent that 'the problem of programming' cannot be understood in a narrow sense. One of the most critical aspects of software development is that of binding meanings to artefacts that ostensibly are – or are to be – specified purely in computational terms. The idea that such semantic considerations can be comprehensively addressed by formal computational semantics has been

criticised by reviewers representing many different perspectives. They include the expert software consultant Michael Jackson (2006), the distinguished computer scientist Peter Naur (1985) and the philosopher Cantwell-Smith (2002). The common theme in these, and other critiques, is that formal semantics can only go so far in mediating meanings in the software development process, especially where the activity involves 'radical design' (cf. Jackson (2006); Vincenti (1993)).

Whereas it suited Turing's (1936) purpose in addressing his mathematical objective to consider human states of mind associated with carrying out a calculation, quite different kinds of states of mind feature in modern software development involving radical design. Such a development process has to take account of the perspectives of many human participants whose understanding is mediated in quite different ways from those of the traditional 'human computer': they cannot be expected to appreciate the full purposes or context for actions, to be able to interpret formal notations reliably, or to be able to communicate their wishes abstractly without reference to actual experience that can only be had and skills that can only be developed for instance by interacting with a prototype system. And even though the functional goal and the process itself may be clearly specified, the practical situated knowledge needed to enact the process may itself be difficult to access – as when we try to make a pot of tea in a neighbour's house, and have to contend with finding the ingredients ('where are the tea bags?'), identifying the utensils we need ('is that a teapot?'), and determining how to configure these ('where do I plug this in?'). In developing software for reactive systems, this exploratory activity may take yet more extreme forms: in configuring devices and tuning their responses, it as if we are investigating the feasibility of constructing the very hardware on which our programs are to be executed (Beynon et al., 2006).

The duality that separates 'the given already engineered computing device' from 'the to-be-specified abstract sequence of instructions to be performed on the device' is characteristic of the computational framework within which Turing was reasoning. Turing (1950) exploits this characteristic when he identifies The Imitation Game as 'drawing a fairly sharp line between the physical and the intellectual capabilities of a man', and stipulates that 'the interrogator cannot demand practical demonstration'.

An instructive comparison can be made between Turing's approach to modelling the mind of a human computer and that conceived by David Gooding (1990) in his account of Faraday's seminal experimental work on electromagnetic phenomena. In interpreting the way in which this activity was conducted, Gooding (2001) introduced the notion of 'construals' as 'proto-interpretative representations which combine images and words as provisional or tentative interpretations of novel experience … [that is] being created … through the interaction of visual, tactile, sensorimotor and auditory modes of perception together with existing interpretative concepts including mental images'. Such construals can be regarded as a means to knowledge representation in spirit similar to that advocated by Rodney Brooks (cf. Brooks (1991a) and Brooks (1991b)). Gooding (2001) invokes his research into Faraday's use of construals in his critique of the 'profoundly mistaken' notion 'that systematic, rational thought is or can be separate from the world that it seeks to understand, manipulate or control'. In collaboration with Addis et al. (2008), Gooding builds on this work to propose a broader notion of computer science embracing 'irrational sets' that 'require the use of an abductive inference system'.

Of the computer-based innovations that have been developed post-Turing, the spreadsheet is perhaps the one that is most directly connected with the 'modelling of states of mind' that Turing (1936) discussed. For instance, a spreadsheet can be viewed as a particularly effective way to represent human states of mind at the interface between the user and the computational process. Several of the characteristic themes that Hodges (1988) identifies in Turing's vision of the TM also seem to be relevant to the spreadsheet concept. The spreadsheet exemplifies 'the blending of the

References

Addis, T., Addis, J.T., Billinge, D., Gooding, D., Visscher, B.-F., 2008. The abductive loop: tracking irrational sets. Found. Sci. 13(5), 5–16.

Beynon, W.M., Boyatt, R.C., Russ, S.B., 2006. Rethinking programming. In: Latifi, S. (ed.), Proceedings of the IEEE Third International Conference on Information Technology: New Generations (ITNG 2006), April 10–12, 2006, Las Vegas, Nevada, USA 2006, pp. 149–154.

Brooks, R.A., 1991a. Intelligence without representation. Artif. Intell. 47, pp. 139–159.

Brooks, R.A., 1991b. Intelligence without reason. Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI-91, Morgan Kaufmann Pub. Inc., San Francisco, CA, pp. 569–595.

Byers, W., 2007. How Mathematicians Think: Using Ambiguity, Contradiction, and Paradox to Create Mathematics. Princeton University Press.

Cantwell-Smith, B., 2002. The foundations of computing. In: Scheutz, M. (Ed.), Computationalism: New Directions. MIT Press, Cambridge, MA, pp. 23–58.

Feferman, S., 2006. Turing's thesis. Notices Am. Math. Sci. 53(10), 1200–1206.

Gooding, D., 1990. Experiment and the Making of Meaning: Human Agency in Scientific Observation and Experiment. Kluwer, Dordrecht.

Gooding, D., 2001. Experiment as an instrument of innovation: Experience and embodied thought. In: Beynon, Nehaniv, and Dautenhahn (Eds.), Proceedings of the 4th International Conference on Cognitive Technology: Instruments of Mind. Springer LNCS, Vol. 2117, pp. 130–140.

Hodges, A., 1988. Alan Turing and the Turing machine. In: Rolf Herken (Ed.), The Universal Turing Machine. A Half-Century Survey, Oxford University Press, Oxford.

Hodges, A., 2004. Alan Turing: the logical and physical basis of computing. In: Proceedings of Alan Mathison Turing 2004: A Celebration of His Life and Achievements, Manchester University, 5 June, 2004. BCS eWiC Series, http://www.bcs.org/content/conWebDoc/17127

Jackson, M.A., 2006. What can we expect from program verification? IEEE Comput., 39(10), 53–59.

James, W., 1912/1996. Essays in Radical Empiricism (Reprinted from the original 1912 edition by Longmans, Green and Co., New York), London: Bison Books.

Naur, P., 1985. Intuition in software development. In: Ehrig, H., Floyd, C., Nivat, M., Thatcher, J.W. (Eds.), Mathematical Foundations of Software Development, Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT), Berlin, Germany, March 25-29, 1985, Volume 2. Lecture Notes in Computer Science 186, Springer, pp. 60–79.

Post, E., 1965. Absolutely unsolvable problems and relatively undecidable propositions. In: M. Davis (Ed.), The Undecidable – Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions, Raven Books, New York.

Steinle, F., 1997. Entering new fields: exploratory uses of experimentation. Phil. Sci. 64 (Proc.), pp. S65–S74.

Turing, A.M.. 1936. On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. 42(2), 230–265.

Turing, A.M., 1944–45. The Reform of Mathematical Notation (unpublished, in Collected Works).

Turing, A.M., 1950. Computing machinery and intelligence, Mind 49, 433–460.

Turing, A.M., 1951. Programmers' Handbook for the Manchester Electronic Computer Mark II (1st ed.), Computing Machine Laboratory, Manchester University, *c.* March 1951. Digital facsimile in The Turing Archive for the History of Computing at http://www.AlanTuring.net/programmers_handbook.

Vincenti, W.C.. 1993. What Engineers Know and How They Know It: Analytical Studies from Aeronautical History. The Johns Hopkins University Press, Baltimore.

Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/entries/turing/

The EM website at url: http://www.dcs.warwick.ac.uk/modelling

Henk Barendregt and Antonio Raffone explore —

Conscious Cognition as a Discrete, Deterministic and Universal Turing Machine Process[1]

1. Systems with states

It is often maintained that the brain-as-computer metaphor is ill taken. Nevertheless one can view conscious cognition as a Turing machine process, with its discrete, deterministic and universal aspects. Not being familiar to the language of science one may object to the claim that computation plays an important role in the life of humans (and in fact all animals). Nevertheless, for goal-directed movements fast and accurate (unconscious) computations are necessary. Sensory input has to be transformed to output in the form of adequate action. Cognitive scientists, who are aware of the need for computation, still may object to the computer metaphor. Indeed, our brain is not a network of Boolean switches and it does neither have numerical input nor output. Our claim is that it is nevertheless useful to interpret cognition as a hybrid Turing machine process.

Modelling systems (machines or living organisms) the notion of 'state' is important. Only considering stimulus-reaction (Input, Action) transitions, we get

$$I \mapsto A. \tag{1.1}$$

This 'behaviouristic' view has limited possibilities. Actual systems can react differently to the same input. To model this difference, inspired by Turing machines, one introduces states, modifying (1.1) to

$$I \times S \mapsto A \times S. \tag{1.2}$$

Now the output may depend also on the state. This will be elaborated below.

2. The Turing machine: processes and computation

A Turing machine is a theoretical model of ad hoc computing devices, including the universal Turing machine,[2] after which the modern digital computers are built. It consists of a potentially two-sided infinite tape[3] with memory cells, a movable reading/writing head placed on one of the cells, and a finite set S of states. The cells each contain a symbol from a finite input alphabet I (set of symbols). Each specific Turing machine is determined by:

$$t_1, \ldots, t_m : I \times S \mapsto A \times S, \tag{2.1}$$

[1] Added in print. After acceptance for publication of this commentary we found out that in Zylberberg et al. (2011) overlapping ideas have been presented.

[2] The universality means that just one machine can simulate the behaviour of all other ones by giving it various *programs*.

[3] In modern computers a disc or flash memory is used instead of a tape. The infinity of the tape was proposed by Turing in order to be technology independent. But each computation on a Turing machine uses only a finite amount of memory.

where we have the following

$$\begin{aligned} I &= \text{set of possible inputs (symbols)} \\ &\quad \text{the head may read on the tape at its location,} \\ S &= \text{set of possible states,} \\ A &= \{L, R, W(a)\}, \text{ the set of possible actions:} \\ L &= \text{moving head left (or the tape moves right),} \\ R &= \text{moving head right (or the tape moves left),} \\ W(a) &= \text{overwriting present location with symbol } a \in I. \end{aligned}$$

For example a machine M can have a, b in I and s_1, s_2 in S, and transition rules

$$\begin{aligned} t_1: &\quad \langle a, s_1 \rangle \mapsto \langle R, s_2 \rangle \\ t_2: &\quad \langle b, s_1 \rangle \mapsto \langle W(a), s_2 \rangle \end{aligned}$$

with the following meanings.

t_1: if M reads an a in state s_1, then
the reading head moves one cell to the right and M enters state s_2;

t_2: if M reads a b in state s_1, then
it (over)writes (this b with) an a and M enters state s_2.

With a Turing machine one can run *processes* and perform *computations*.

A *computation* starts with an input. In the Turing machine this is represented as a finite sequence of data, elements of I, written on consecutive cells of the tape. The other cells are blank (also considered as an element of the alphabet I). The read/write head is located at a particular cell of the tape and the machine is in an initial state q_0. The machine performs the actions according to its transition rules, until no more rule applies and the machine 'halts'. The resulting contents on the tape is considered as the output of the computation.

Turing made it plausible that any kind of mechanical computation can be performed in such a way. Moreover, he constructed a single Turing machine UM, the *Universal Machine*, that can simulate an arbitrary Turing machine M. Wanting to simulate the computation of M on input i, notation $M(i)$, one can construct a program p_M for M such that for all input i one has

$$UM(p_M, i) = M(i).$$

This means that UM requires an extra argument, the program code p_M, next to the given argument i. Turing used it to define a problem that cannot be answered by the computation of any Turing machine and hence not by any computation.

A *process* is like a computation, but without the requirement that there is a final state in which the machine comes to a halt. So computations are special processes focused on *termination*; processes in general are focused on *continuation*. The usefulness of processes can be seen by giving some of the cells on the tape a special status: for input ('sensors') and for output ('actuators') from and to the outside world. A factory involving heating devices, thermometers, and safety valves, may be controlled in this way by a Turing machine acting as process.

The process (or computation) taking place in a Turing machine is discrete and deterministic: it consists of a stream of distinct steps, only depending on the input.

3. The neural Turing machine

From the description of a process it is clear that life (humans, animals and even plants) can be thought of as processes. In artificial intelligence (AI) one tries to emulate these processes. There are

the two views in AI, one the symbolic rule based Simon and Newell (1958), and the connectionist one related to Turing (1948) and Hillis (1989). Simon and Newell state that intelligence works in a discrete serial way following specific rules. The connectionists state that cognition uses the parallelism of 'neural nets' and not a sequential system. In the hybrid version of Turing machines presented below, the sequential machine will get transition rules programmed by a parallel neural net, providing a useful unification for understanding human cognition.

Let us review the model of the Turing machine. A particular such machine is determined by a finitely specified transition map (2.1). Now we slightly change the interpretation of this notation.

$$\begin{aligned} I &= \text{now stands for sensory input,} \\ S &= \text{set of possible states,} \\ A &= \text{now stands for actions, including neural excitation for} \\ &\quad \text{movement and focussing attention,} \\ \mapsto &= \text{the transition determined by a neural net.} \end{aligned}$$

We do have a non-essential extension. No longer is I a finite alphabet, but a virtually unbounded set of inputs from the world. It still is essentially finite by the limitations of our senses. In a Turing machine the set I is typically of size 2^n, with $n < 10$; in human cognition it is orders of magnitude bigger. The same applies to the set A. This set directs bodily movements, speech or mental action.

Another feature that happens in the brain is that whilst we are processing, our processor does change. This includes development and is essential for homo sapiens. This seems like a proper extension of the notion of a Turing machine. But thanks to the existence of a universal Turing machine this is not so. Instead of (N stands for the neural net determining the transitions and A can act on N)

$$I \times S \xrightarrow[N]{} A \times S \tag{3.1}$$

one can employ the universal machine and write the equivalent

$$I \times p_N \times S \xrightarrow[\text{UM}]{} A \times S.$$

Now it becomes possible that the A act on the program p_N. In ordinary computing this is not advisable, as it is difficult to reason about the resulting effects. But in the neural evolution it fits perfectly well.

In the resulting model of cognition the set of states S plays an important role. Rather than seeing human cognition in a stimulus response fashion like in (1.1), as was fashionable in the behaviourist days of last century, the cognitive model (3.1) shows the essence of states. A 'state' is a mathematical concept: giving the same input–output relation. We know empirically that attention and emotions greatly influence these states. Under the same circumstances these inner state can make of a human being a saint, a scientist, a Scrooge or worse. It should be noted that the model (3.1) is discrete. Conscious cognition is a stream of separate phenomena, taking place in time. We will come back to this in the next section.

4. Conscious cognition: discrete temporal frames

A currently influential model of human conscious cognition is the global workspace (GW) theory (Baars, 1998; Baars et al., 2003). In this model, conscious cognition enables an access to a varying subset of brain sources.

A neuronal underpinning for the GW model has been developed in Dehaene and Naccache (2001). It is characterised by a winner-take-all dynamics, forming a 'neural processing bottleneck', involving 'broadcasting' activity from prefrontal cortex to neurons on a global scale in the brain.

Only one large-scale reverberating neural assembly is assumed to be active at any given moment. This crucially involves the thalamocortical pulse and imposes a temporal resolution for the stream of conscious cognition, needing at least 100 ms for a perceptual awareness moment.

Independently, based on psychophysical, neurophysiological and electrophysiological findings, Varela et al. (2001) postulate that a specific large-scale neural assembly underlies the emergence and operation of each conscious cognitive act. Such assemblies occur in the thalamocortical system, using closed-loop signalling with periods of 100–300 ms, see Tononi and Edelman (1998). This is consistent with the earlier behavioural evidence of the psychological refractory period, based on minimal temporal resolutions (Welford, 1952), about 150 ms.

On the other hand, Efron (1973) suggested, based on psychophysical evidence, that conscious cognition is temporally discrete and parsed into sensory sampling intervals or 'perceptual frames', estimated to be about 70–100 ms in average duration. More recently, based on psychophysical and electrophysiological evidence, the range 70–100 ms has been interpreted as an attentional object-based sampling rate for visual motion (van Rullen and Koch, 2006). This rate could be related to a sequence of shorter temporal processes, needed for unconscious treatment of sensory and other input, see van Rullen and Koch (2003) for a review. It may provide an estimate of the rate at which temporal representations at an unconscious level can be accessed (van Wassenhove, 2009).

To reconcile the framing of conscious cognition with the apparent continuity of perceptual experience, John (1990) suggested the following mechanism. A cortical convergence of a cascade of momentary perceptual frames establishes a steady-state perturbation from baseline brain activity. This idea has received substantial support from electroencephalographic (EEG) studies. The dynamics of the EEG field is represented by intervals of quasi-stability or 'microstates', with sudden transitions between them (Strik and Lehmann, 1993).

5. Conscious cognition: mind states

According to Baars' GW theory (Baars et al., 2003), sensory cognition works as follows. Input as signals from the sensory cortex are amplified by attention and become 'contents' of consciousness. After this amplification, feed back to the sensory cortex takes place to enable conscious access to the contents themselves, in a recurrent GW process. See Dehaene and Naccache (2001) and Lamme (2003).

In this process 'contextual' brain systems play a role in shaping conscious events. These include the 'where' and 'what' pathways in the parietal cortex for visual processing, see Milner and Goodale (2008). Regions of prefrontal cortex appear to do the same for other aspects of experience, including emotional, goal-related and self-representation aspects (Baars et al., 2003). Also the insula appears to play a crucial role as body- and feeling-related contextual system for awareness (Craig, 2009). More in general, as shown by behavioural research, affective states, including moods and emotions, provide a inner context guiding different forms of human judgment and cognitive processing, see Clore and Huntsinger (2007) for a review. These contexts can be considered as mind states, not only determining actions, but also the next input via selective attention. Selectivity in turn stems from current goals represented in prefrontal cortex (Duncan, 2001) and can ultimately be related to the current mind state. In a synthetic view, apart from inputs from sensory fields, inputs to the GW come from the GW output itself, see also Maia and Cleeremans (2005), depending on a given mind state.

In a TM controlling an industrial process the input is determined solely by the world. This is not so in human emotional cognition, where attention plays an input selecting role. Therefore mind states are themselves the ground for conscious cognition, not just a context. By their broadcasting, 'speaking to the audience' in Baars' theatre metaphor, they have the greatest influence on the brain state as a whole, and on (intentions for) action and thinking.

The brain substrates for mind states are potentially wider than those for the GW, with an overlap with the latter, and with the inclusion of various kinds of unconscious contextual systems supporting conscious cognition. The neural substrates for longer lasting emotional mind states plausibly also include the cerebrospinal fluid, as discussed in the paper by Veening and Barendregt (2010).

and computer science and for some of the practicalities of using one computer for multiple purposes, including time-sharing. One of the consequences is that a Turing machine implementing another Turing machine can also be a virtual machine implemented in a UTM: so that layered implementations are possible.

In the decades following publication of Turing's paper, engineering developments emerged in parallel with mathematical developments, with some consequences that have not received much attention, but are of great philosophical interest and potentially also biological import. I will suggest in Part 2 that biological evolution 'discovered' many of the uses of virtual machinery long before we did. Unfortunately, the word 'virtual' suggests something 'unreal' or 'non-existent', whereas virtual machines can make things happen: they can be causes, with many effects, including physical effects. To that extent they, and the objects and processes that occur in them, are *real* not *virtual*!

A possible source of misunderstanding is the fact that among a subset of computer scientists the label 'virtual machine' refers to software implementations of 'real', 'physical' machines which they accurately simulate (Popek and Goldberg, 1974). The notion of 'virtual machine' used in this paper includes machines whose operations cannot all be defined in terms of physical properties, although they are all *implemented* in physical machinery, and can interact with and control physical machinery. These virtual machines should not be regarded as surrogates for 'real' physical machines. They are real enough, in their causal powers, despite being virtual.

3. Causation and computation

Causation is a crucial aspect of the engineering developments in computing, as I'll now try to explain. It is possible to take any finite collection of Turing machines and emulate them running in parallel, in synchrony, on a UTM. This demonstrates that *synchronised* parallelism does not produce any qualitatively new form of computation. The proofs are theorems about relationships between abstract mathematical structures including sequences of states of Turing machines – and do not mention physical causation. A running physical machine can be an instance of such an abstract mathematical structure. However, being physical it can be acted on by physical causes, e.g. causes that alter its speed. Moreover, as remarked in Sloman (1996), standard computability theorems do not apply to physical Turing machines that are not synchronised. For example, if TM T1 repeatedly outputs '0', and T2 repeatedly outputs '1', and the outputs are merged to form a binary sequence, then if something (e.g. a device controlled by a geiger counter) causes the speeds of T1 and T2 to vary randomly and they run forever, the result could (and most probably would) be a non-computable infinite binary sequence, even though each of T1 and T2 conforms to theorems about Turing machines. (This claim will be refuted if it ever turns out that the whole physical universe can be modelled on a single Turing machine. I know of no evidence that such a model is possible.)

Likewise, if a machine has physical sensors and some of its operations depend on the sensor readings, then the sequence of states generated may not be specifiable by any TM, if the environment is not equivalent to a TM. So the mathematical 'limit' theorems do not apply to all physically implemented information-processing systems. In fact a machine with sensors and effectors connected to physical objects in the environment is fundamentally different from a Turing machine running its 'closed' world consisting only of its (infinite tape) and controlling transition table.

Mathematical entities, such as numbers, functions, proofs and abstract models of computation, do not have spatio-temporal locations, whereas running instances of computations do, some of them distributed across networks. Likewise, there are no causal connections, only logical connections, between the TM states that form the subject matter of the mathematical theory of computation, whereas there *are* causal connections in the running instances, depending on the physical machinery used and the physical environment. So, notions like 'reliability' are relevant to the physical

instances, but not the mathematical abstractions. From a mathematical point of view there is no difference between three separate computers running the same program, and a single computer simulating the three computers running the program. However, an engineer aiming for reliability would choose three physically separate computers with a voting mechanism as part of a flight control system, rather than a mathematically equivalent, equally fast, implementation in a single computer (Sloman, 1996), if all the computers use equally reliable components.

Physical details of time-sharing of the machines have other consequences. When the three separate machines running in synchrony switch states in unison, nothing happens between the states, whereas in the time-shared implementation on one computer, the underlying machine has to go through operations to switch from one virtual machine to another. Such 'context switching' processes have intermediate sub-states that do not occur in the parallel implementation. A detailed mathematical model of one machine running three virtual machines will need to include the intermediate states that occur during switching, but a model of three separate concurrently active machines will not. A malicious intruder, or a non-malicious operating system, will have opportunities to interfere with the time-shared systems during a context-switching process, e.g. modifying the emulated processes, interrupting them, or copying or modifying their internal data.

Such opportunities for intervention (e.g. checking that a sub-process does not violate access restrictions or transferring information between devices) are often used both within individual computers and in networked computers causally linked to external environments, e.g. sensing or controlling physical devices, chemical plants, air-liners, commercial customers, social or economic systems, and many more. In some cases, analog-to-digital digital-to-analog converters, and direct memory access mechanisms now allow constant interaction between processes. See also Dyson (1997).

The technology supporting the causal interactions includes (in no significant order): *memory management, paging, cacheing, interfaces of many kinds, interfacing protocols, protocol converters, device drivers, interrupt handlers, schedulers, privilege mechanisms, resource control mechanisms, file-management systems, interpreters, compilers, 'run-time systems' for various languages, garbage collectors, mechanisms supporting abstract data types, inheritance mechanisms, debugging tools, pipes, sockets, shared memory systems, firewalls, virus checkers, security systems, operating systems, application development systems, name-servers*, and more. All of these can be seen as contributing to intricate webs of causal connections in running systems, including *preventing* things from happening, *enabling* certain things to happen in certain conditions, *ensuring* that if certain things happen then other things happen, and in some cases *maintaining mappings* between physical and virtual processes, e.g. in device drivers. Philosophers who think that different causal webs at different levels of abstraction cannot coexist need to learn more engineering, unfortunately not a standard component of a philosophy degree.

4. Causation in RVMs

A running virtual machine can have many effects, including causing its own structure to change. Understanding how virtual machines can cause anything to happen requires a three-way distinction, between: (a) *Mathematical Models* (MMs), e.g. numbers, sets, grammars, proofs, etc., (b) *Physical Machines* (PMs), including atoms, voltages, chemical processes, electronic switches, etc., and (c) *Running Virtual Machines* (RVMs), e.g. calculators, games, formatters, provers, spelling checkers, email handlers, operating systems, etc., running in general-purpose computers.

MMs are static abstract structures, like proofs and axiom systems. Like numbers, they cannot *do* anything. They include Turing machine executions whose properties are the subject of mathematical proofs. Unfortunately some uses of 'virtual machine' refer to MMs, e.g. 'the Java virtual machine'. These are abstract, inactive, mathematical entities, not RVMs, whereas PMs and RVMs are active and cause things to happen.

Physical machines on our desks can now support varying collections of virtual machinery with various kinds of concurrently interacting components whose causal powers operate in parallel with the causal powers of underlying virtual or physical machines, and help to control those physical machines. Some of them are *application* RVMs that perform specific functions, e.g. playing chess, correcting spelling, handling email. Others are *platform* RVMs, like operating systems, or run-time systems of programming languages, which are capable of supporting many different higher level RVMs. Different RVMs have *different levels of granularity* and *different kinds of functionality*. They all differ from the granularity and functionality of the physical machinery. Relatively simple transitions in a RVM can use a very much larger collection of changes at the machine code level and an even larger collection of physical changes in the underlying PM – far more than any human can think about. Apart from the simplest programs even machine code specifications are unmanageable by human programmers. Automatic mechanisms (including compilers and interpreters) are used to ensure that machine-level processes support the intended RVMs.

Interpreted and compiled programming languages have important differences in this context. An interpreter ensures *dynamically* that the causal connections specified in the program are maintained. If the program is changed while running, the interpreter's behaviour will change. In contrast, a compiler *statically* creates machine code instructions to ensure that the specifications in the program are subsequently adhered to, and the original program plays no role thereafter. Changing it has no effect, unless it is recompiled (e.g. if an *incremental* compiler is used). In principle the machine code instructions can be altered directly by a running program (e.g. using the 'poke' command in Basic) but this is usually feasible only for relatively simple changes and would probably not be suitable for altering a complex plan after new obstacles are detected, and modifying the physical wiring would be out of the question. So some kinds of self-monitoring and self-modification are simplest if done using process descriptions corresponding to a high level virtual machine specified in an interpreted formalism and least feasible if done at the level of physical structure. Compiled machine code instructions are an intermediate case.

There are two different benefits of using a suitable RVM, namely (a) the already mentioned coarser granularity of events and states compared with a PM or low level RVM, and (a) the use of an ontology related to the application domain (e.g. playing chess, making airline reservations). Both of these are indispensable for processes of design, testing, debugging, extending, and also for run-time self-monitoring and control, which would be impossible to specify at the level of physical atoms, molecules or even transistors (partly because of explosive combinatorics, especially on time-sharing, multi-processing systems where the mappings between virtual and physical machinery keep changing). The coarser grain, and application-centred ontology makes self-monitoring (like human debugging of the system) more practical when high-level interpreted programs are run than when machine code compiled programs are run. This relates to the third aspect of some virtual machinery: ontological irreducibility.

5. Implementable but irreducible

The two main ideas presented so far are fairly familiar, namely (a) a VM can run on another (physical or virtual) machine, and (b) RVMs running in parallel can interact causally with one another and with things in the environment. A third consequence of 20th century technology is not so obvious, namely: *some VMs include states, processes and causal interactions whose descriptions require concepts that cannot be defined in terms of the language of the physical sciences: they are non-physically describable machines (NPDMs).* Virtual machinery can extend our ontology of types of causal interaction beyond physical interactions.

This is not a form of mysticism. It is related to the fact that a scientific theory can use concepts (e.g. 'gene', 'valence') that are *not definable* in terms of the actions and observations that scientists can perform. This contradicts both the 'concept empiricism' of philosophers like Berkeley

and Hume, originally demolished in Kant (1781), and also its modern reincarnation, the 'symbol grounding' thesis popularised by Harnad (1990), which also claims that all concepts have to be derived from experience of instances. The alternative 'theory tethering' thesis, explained in Sloman (2007), is based on the conclusion in twentieth century philosophy of science that undefined symbols used in deep scientific theories get their meanings primarily, though not exclusively, from the structure of the theory, though a formalisation of such a theory need not fully determine what exactly it applies to in the world. The remaining indeterminacy of meaning is partly reduced by specifying forms of observation and experiment (e.g. 'meaning postulates' in Carnap (1947)) that are used in testing and applying the theory, 'tethering' the semantics of the theory. The meanings are never uniquely determined, since it is always possible for new observations and measurements (e.g. of charge on an electron) to be adopted as our knowledge and technology advance.

Ontologies used in specifying VMs, e.g. concepts like 'pawn', 'threat', 'capture' etc. used in specifying a chess VM, are also mainly defined by their role in the VM, whose specification expresses an explanatory theory about chess. Without making use of such concepts, which are not part of the ontology of physics, designers cannot develop implementations and users cannot understand what the program is for, or make use of it. So, when the VM runs, there is a physical implementation that is also running, but the two are not identical: there is an asymmetric relation between them. The PM is an *implementation* of the VM, but the VM is not an implementation of the VM, and there are many other statements that are true of one and false of the other. The RVM, but not the PM, may include threats, and defensive moves. And neither 'threat' nor 'defence' can be defined in the language of physics. Not all the concepts used to describe objects, events and processes in a RVM are *definable* in terms of concepts of physics even though the RVM is *implemented* in a physical machine. The physical machine could include some of the environment with which the RVM interacts. The detailed description of the PM is not a specification of the VM, since the VM could be the same even if it were implemented on a very different physical machine with different physical processes occurring during the execution even of a particular sequence of chess moves. The VM description is also not equivalent to any fixed *disjunction of descriptions* since the VM specification determines which PMs are adequate implementations. Programmers can make mistakes, and bugs in the virtual machinery are detected and removed, usually by altering a textual specification of the abstract virtual machinery not the physical machinery. When a bug in the program is fixed it does not have to be fixed differently for each physical implementation – a compiler or interpreter for the language handles the mapping between virtual machine and physical processes and those details are not part of the specification of the common virtual machine.

Neither can the VM machine states and processes be defined in terms of physical input-output specifications, since very different technologies can be used to implement interfaces for the same virtual machine, e.g. using mouse, keyboard, microphone or remote email for input. Moreover, some VMs perform much richer tasks than can be fully expressed in input–output relations, e.g. the visual system of a human (or future robot!) watching turbulent rapids in a river. (Compare the critique of Skinner in Chomksy (1959).)

The indefinability of VM ontologies in terms of PM ontologies does not imply that RVMs include some kind of 'spiritual stuff' that can exist independently of the physical implementation machinery, as assumed by those who believe in immortal minds, or souls. Despite the indefinability there are close causal connections between VM and PM states, but that includes things like detection of a threat causing a choice of defensive move, which is a VM process that can cause changes in the physical display and the physical memory contents. We thus have what is sometimes referred to as 'downwards causation', in addition to 'upwards causation' and 'sideways causation' (within the RVM).

6. Implications

The complex collection of hardware, firmware, and software technologies, developed since Turing's time has made possible information-processing systems of enormous complexity and sophistication performing many tasks that were previously performed only by humans and some that not even

humans can perform. This has required new ways of thinking about *non-physically describable* virtual machinery (NPDVM) with causal powers. The new conceptual tools are relevant not only to engineering tasks but also to understanding what self-monitoring, self-controlling systems can do. Philosophy now has the task of working out in detail metaphysical implications of multiple coexisting causal webs with causation going sideways, upwards and downwards. Implications for evolution of mind are discussed in Part 2 of this paper, included in Part III of this volume. Finally, Part 3 of this paper, presenting the concept of meta-morphogenesis (the processes by which the processes of change and development change) will be included in Part IV of this volume.

References

Carnap, R., 1947. Meaning and Necessity: A Study in Semantics and Modal Logic. Chicago University Press, Chicago.

Chomksy, N., 1959. Review of skinner's Verbal Behaviour. Language, 35, 26–58.

George, B.D., 1997. Darwin Among The Machines: The Evolution Of Global Intelligence. Addison-Wesley, Reading, MA.

Harnad, S., 1990. The symbol grounding problem. Physica D, 42, 335–346.

Kant, I., 1781. Critique of Pure Reason. Macmillan, London. Translated (1929) by Norman Kemp Smith.

Popek, G.J., Goldberg, R.P., 1974. Formal requirements for virtualizable third generation architectures. Commun. ACM 17 (7).

Sloman, A., 1996. Beyond turing equivalence. In: Millican, P.J.R., Clark, A. (Eds.), Machines and Thought: The Legacy of Alan Turing (vol I), The Clarendon Press, Oxford, pp. 179–219. URL http://www.cs.bham.ac.uk/research/projects/cogaff/96-99.html#1. (Presented at Turing90 Colloquium, Sussex University, April 1990.

Sloman, A., 2007. Why symbol-grounding is both impossible and unnecessary, and why theory-tethering is more powerful anyway. http://www.cs.bham.ac.uk/research/projects/cogaff/talks/#models.

Turing, A.M., 1936. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. 42 (2), 230–265. URL http://www.abelard.org/turpap2/tp2-ie.asp.

Artur Ekert on the physical reality of —

$$\sqrt{NOT}$$

One of many remarkable traits of Alan Turing was his ability to bridge the gap between the abstract and the physical. His background in physics is clearly seen in his approach to the definition of computability. Turing's machines (Turing, 1936) captured the notion of effective computation in a much more tangible and convincing way than, for example, the lambda calculus proposed by Alonzo Church (this was generously acknowledged by Church (1937) himself). Although Turing's machines were abstract constructs of his mathematical imagination there was nothing unphysical about them. Indeed, Turing's machines (with arbitrarily long tapes) can be built, but no one would ever do so except for fun, as they would be extremely slow and cumbersome. The computer I am working on at the moment is much faster and more reliable.

But wait a minute! Where does this reliability come from? My computer is a physical object, made out of a vast number of electronic components. How do I know that the computer generates the same outputs as the appropriate abstract Turing machine? How do I know that the machinery

Quantum theory explains the behaviour of $\sqrt{\textbf{not}}$ and correctly predicts the probabilities of all the possible outputs no matter how we concatenate the machines. This knowledge was created as the result of conjectures, experimentation, and refutations. Hence, reassured by the physical experiments that corroborate this theory, logicians are now entitled to propose a new logical operation $\sqrt{\textbf{not}}$. Why? Because a faithful physical model for it exists in nature!

The story of the $\sqrt{\textbf{not}}$ is just one example which illustrates the main point: whenever we improve our knowledge about physical reality, we may also gain new means of improving our knowledge of logic, mathematics and formal constructs. It seems that we have no choice but to recognise the dependence of our mathematical *knowledge* (though not of mathematical truth itself) on physics, and that being so, it is time to abandon the classical view of computation as a purely logical notion independent of that of computation as a physical process (Deutsch, 1997; Deutsch et al., 2000).

References

Church, A., 1937. Review of Turing 1936. J. Symb. Log. 2, 42–43.

Deutsch, D., 1985. Quantum theory, the Church-Turing principle and the universal quantum computer. Proc. R. Soc. A: Math. Phys. Eng. Sci. 400, 97–117.

Deutsch, D., 1997. The Fabric of Reality. Allen Lane, The Penguin Press, New York, London.

Deutsch, D., Ekert, A., Lupacchini, R., 2000. Machines, logic and quantum physics. Bull. Symb. Log. 6, 265–283.

Ekert, A., 2006. Quanta, ciphers and computers. In: Fraser, G. (Ed.), The New Physics for the Twenty-First Century. Cambridge University Press, Cambridge, New York, pp. 268–283.

Nielsen, M. A., Chuang, I. L., 2000. Quantum Computation and Quantum Information. Cambridge University Press, Cambridge, New York.

Turing, A. M., 1936. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. 42, 230–265.

Cristian Calude, Ludwig Staiger and Michael Stay on —

HALTING AND NON-HALTING TURING COMPUTATIONS

1. Introduction

Turing's famous paper (Turing, 1936) proved that the *halting problem* – the problem of deciding whether a given Turing machine ever reaches the halting state when provided with a given tape – was undecidable. Turing machines give us a convenient way to talk about the time and space necessary to carry out computations, and play a significant role in both classical recursion theory and the theory of computational complexity (Cooper, 2004; Balcázar et al., 1995; Sipser, 2006; Wagner and Wechsung, 1986).

Nowadays, the undecidability discovered by Turing need not be quite the fearsome phenomenon it at first appears. To understand this we look in more detail at the time and space of Turing computations.

First, any Turing machine having an undecidable halting problem uses an infinite number of cells on its working tape (Calude and Staiger, 2010). Accordingly, the halting behaviour of a Turing machine M on input x can be divided into three categories:

(1) The machine M halts on x, in which case the number of cells used is necessarily finite.
(2) The machine M does not halt on x, but uses only finitely many distinct cells on its tapes.
(3) The machine M does not halt on x and uses infinitely many distinct cells on its tapes.

In the second case above, the halting problem for M on x is decidable, so Turing's undecidability result relies on the fact that machines with an undecidable halting problem necessarily use infinite space.

Secondly, the critical time (Chaitin, 1987) can be used to yield a classification of Turing computations into three categories:

- The machine M halts on x in time bounded by the critical number of steps.
- The machine M halts on x in time not bounded by the critical number of steps.
- The machine M does not halt on x.

The last case, that is when M does not halt on x, can be refined in terms of space complexity. Finally, the significance of these mathematical facts for hypercomputation and formal proofs in mathematics will be briefly discussed.

2. Turing machines

A Turing machine is a formalisation of a mechanical device. The device has a long tape on which a finite alphabet of symbols are read or written, and the tape can be shifted back and forth through the machine, which can read symbols from and write symbols to the tape. The machine itself has a finite set of internal states, and updates those states depending on what it finds on the tape. In the formalisation, the tape is infinite and there are never any errors. Turing's own description is wonderfully lucid, and we refer the reader to his account for details.

The set of pairs (M,x), where M is a Turing machine and x is an input, can be computably enumerated. We fix such an enumeration and we denote by $code(M,x)$ the code, or description, of the pair (M,x) in this enumeration.

3. Resources

Let M be a Turing machine and x an input word.

The function $time_M(x)$ denotes the number of steps executed by M on input x (see Balcázar et al. (1995)). By $M(x) < \infty$ we denote the fact that M stops on x. The *halting problem for a particular Turing machine* M is the problem of deciding, given x, whether $M(x) < \infty$. The *halting set* or the *domain* of M is the set $halt_M = \{x \mid M(x) < \infty\}$. It is well known that the halting problem for most Turing machines M is undecidable; more precisely, the halting set of M is computably enumerable but not computable. A Turing machine whose domain is prefix-free is called *self-delimiting* or *prefix-free* (Calude, 2002). Although many results presented below hold for any Turing machine, for uniformity we study only prefix-free Turing machines, which from now on we will simply call *machines*.

The computational space, or *space function*, $space_M(x)$ used by M on x is defined to be the number – finite or infinite – of cells used by M during its computation with the input x; a cell used

at least once is counted as used.[4] Obviously, if $space_M(x)$ is finite, then the computation process as described above can have only a finite number of different configurations.[5]

Clearly, $space_M(x) < \infty$ whenever $M(x) < \infty$, and $M(x) = \infty$ if and only if $time_M(x) = \infty$.

Given a machine M, we can therefore classify input strings x according to $time_M$ and $space_M$ and get the following three sets:

$$halt_M = \{x \mid time_M(x) < \infty\},$$
$$\{x \mid time_M(x) = \infty,\ space_M(x) < \infty\},$$
$$\{x \mid space_M(x) = \infty\}.$$

Calude and Staiger (2010) showed that if for every x, $space_M(x) < \infty$, then the halting problem for M is decidable. This result does not contradict Turing's undecidability of the *halting problem* because the set of descriptions $code(M,x)$ – where M is a machine, x is a string – for which $space_M(x) < \infty$, is computably enumerable but not computable.

4. Halting time

Let bin be the computable bijection that associates to every integer $n \geq 1$ its binary expansion without the leading 1: $\text{bin}(1)$ is the empty string, $\text{bin}(2) = 0$, $\text{bin}(3) = 1$, $\text{bin}(4) = 00$ etc. The *natural complexity* of the string y (with respect to the machine M) is $\nabla_M(y) = \min\{n \geq 1 \mid M(\text{bin}(n)) = y\}$ (see Calude and Stay (2006)); ∇ is a relative of Kolmogorov complexity for partially computable functions used in Manin and Zilber (2010).

The invariance theorem says that one can effectively construct a 'universal machine' that can simulate any other machine. A machine U is universal if for every machine M there is a constant $\varepsilon > 0$ (depending upon U and M) such that $\nabla_U(x) \leq \varepsilon \cdot \nabla_M(x)$, for all strings x. We fix a universal machine U and define $\nabla = \nabla_U$.

Say a machine M gets an input x and runs for exactly t steps before halting. Chaitin (1987) showed that there is a program y for the universal machine not much longer than $code(M,x)$ such that $U(y) = t$ – or more formally, that there is a constant c such that if $M(x)$ halts exactly in time t, then $\nabla(\text{bin}(t)) \leq 2^{|code(M,x)|+c}$.

A binary string x is algorithmically random if $\nabla(x) \geq 2^{|x|}/|x|$. A time t is called algorithmically random if $\text{bin}(t)$ is algorithmically random. Assume that $M(x)$ has not stopped in time $2^{2N+2c+1}$, where $N = |code(M,x)|$ and c comes from Chaitin's statement above; then Calude and Stay (2008) proved that $M(x)$ cannot stop at any algorithmically random time $t \geq 2^{2N+2c+1}$.

Therefore, if one runs a program for long enough (where 'long enough' depends on c above), then either the program halts at a non-algorithmically random time or it does not halt at all. The density of non-algorithmically random numbers near n is $1/n$. Hence, *most times are not halting times for any machine and input.*

Consider the set $S = \{0,1\}^R \times \mathbb{N}$ whose elements are pairs of an input string of length R and a potential runtime. Let $q(x)$ be the uniform probability distribution on strings of length R and $p(n)$ be any computable probability distribution on natural numbers. Given M,x and a positive integer m, we can effectively compute a critical value $T_{critical}(|code(M,x)|,m)$ such that either $M(x)$ stops in time less than $T_{critical}(|code(M,x)|,m)$, or the probability given by $q \times p$ that $M(x)$ eventually stops is smaller than 2^{-m}.

[4] This definition differs slightly from the space complexity usually employed in computational complexity theory (Balcázar et al., 1995), which treats the space as infinite if the time is infinite.

[5] A configuration records the current state, tape contents and head location (Sipser, 2006, p. 140).

Manin (2010) proved a general result of this kind valid for many complexity measures, including time.

Given a probability bound 2^{-m}, the halting behaviour of a machine M can be described by the following three sets:

$$\{x \mid time_M(x) < T_{critical}(|code(M,x)|,m)\},$$
$$\{x \mid T_{critical}(|code(M,x)|,m) \leq time_M(x) < \infty\},$$
$$\{x \mid time_M(x) = \infty\}.$$

The last case can be refined using the computational space as follows:

$$\{x \mid time_M(x) = \infty, space_M(x) < \infty\},$$
$$\{x \mid time_M(x) = \infty, \text{ ZFC proves } space_M(x) = \infty\},$$
$$\{x \mid space_M(x) = \infty, \text{ but ZFC cannot prove } space_M(x) = \infty\}.$$

5. Final comments

What is the 'real-world' significance of this commentary?[6]

The bad news is that even an accelerated Turing machine (Copeland, 2002) needs infinite space to solve the halting problem. This begs for more insight into the mysterious usefulness of analogue computation which can bypass this limit (as Kreisel (1970) anticipated). The good news is that the halting problem can be probabilistically solved with any probability less than one.

Hilbert's formal proofs have been apparently killed for the practice of mathematics by Gödel's Incompleteness Theorem, so ultimately by the undecidability of the halting problem. Unexpectedly – at least from the theoretical view point – in the last decade, enormous progress has been made on automating the production of formal proofs, with tools like Isabelle, Coq and others (Hales, 2008). In part, this successful story – which in our humble opinion will change the way mathematics is done – is due to the practical possibility of working with meaningful, computational resource defined, approximations of the halting problem.

Acknowledgements

We thank Alastair Abbott, Barry Cooper and Jan van Leeuwen for comments that improved the presentation.

References

Balcázar, J.L., Díaz, J., Gabarró, L., 1995. Structural Complexity I, second ed. Springer-Verlag, Berlin.

Barrow, J., 2005. The Infinite Book. A Short Guide to the Boundless, Timeless and the Endless, Jonathan Cape, London.

Boolos, G., Jeffrey, R.C., 1980. Computability and Logic, Cambridge University Press, Cambridge.

Calude, C.S., 2002. Information and Randomness: An Algorithmic Perspective, second ed. Springer-Verlag, Berlin.

Calude, C.S., Staiger, L., 2010. A note on accelerated Turing machines. Math. Struct. Comput. Sci. 20, 1011–1017.

[6] ... was the question Barry Cooper asked us at the 9th iteration of this commentary.

Calude, C.S., Stay, M.A., 2006. Natural halting probabilities, partial randomness, and Zeta functions. Inf. Comput. 204, 1718–1739.
Calude, C.S., Stay, M.A., 2008. Most programs stop quickly or never halt. Adv. Appl. Math. 40, 295–308.
Chaitin, G.J., 1987. Computing the busy beaver function. In: Cover, T.M., Gopinath, B. (Eds.), Open Problems in Communication and Computation, Springer-Verlag, Heidelberg, pp. 108–112.
Cooper, S.B., 2004. Computability Theory, Chapman & Hall/CRC, Boca Raton, New York, London.
Copeland, B., 2002. Accelerating Turing machines. Minds Mach. 12 (2), 281–300.
Hales, T.C., 2008. Formal proof. Notices of the AMS 11, 1370–1380.
Kreisel, G., 1970. Church's thesis: a kind of reducibility axiom for constructive mathematics. In: Kino, A., Myhill, J., Vesley, R.E. (Eds.), Intuitionism and Proof Theory, North-Holland, Amsterdam, 121–150.
Manin, Y., 2010. Infinities in quantum field theory and in classical computing: renormalization program. In Ferreira, F., Löwe, B., Majordomo, E., Gomes, L.M. (Eds.), Programs, Proofs, Process. Proceedings CiE 2010, LNCS 6158, Springer, Heidelberg, pp. 307–316. Full paper in Renormalization and Computation II: Time Cutoff and the Halting Problem, http:// arxiv.org/abs/0908.3430, 24 August 2009.
Manin, Y., Zilber, B., 2010. A Course in Mathematical Logic for Mathematicians, second ed. Springer, Heidelberg.
Sipser, M., 2006. Introduction to the Theory of Computation, second ed. PWS, Boston.
Turing, A.M., 1936. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. 2 42, 230–265.
Turing, A.M., 1937. On computable numbers, with an application to the Entscheidungsproblem: A correction. Proc. Lond. Math. Soc. 2, 43, 544–546.
Wagner, K., Wechsung, G., 1986. Computational Complexity, Deutscher Verlag der Wissenschaften, Berlin.

Philip Welch leads us —

TOWARD THE UNKNOWN REGION: ON COMPUTING INFINITE NUMBERS

Darest thou now, O Soul,
Walk out with me toward the
Unknown Region,
Where neither ground is for the feet,
nor any path to follow?

Whitman, Leaves of Grass

The story is told that Church, after presenting the λ-calculus as a means of addressing the *Entscheidungsproblem*, was told by Gödel, in effect, to go away and try again. Gödel in conversation with Church said that he found the suggestion that effectively calculable be identified with the λ-calculus as 'thoroughly unsatisfactory'.[1] However, Gödel immediately recognised Turing's model as *the* model for this problem. Why was this? Presumably the criteria, somewhat implicit in Hilbert's question, as to whether there is an algorithm, or 'effective process' to determine from an 'effectively' given set of axioms in a deductive system, for any sentence φ of the system's language, whether or not φ was deducible from those axioms, would hinge crucially on the notion of algorithm or 'effectivity'. As Gödel had already shown the incompleteness of formal systems (satisfying a modicum of some basic requirements and thereby introducing the primitive recursive functions),

[1] This is recalled in a letter from Church to Kleene (Kleene, 1959). The conversation took place in early 1934 (Rosser, 1984).

were domains inside which certain natural Σ_1-*set theoretic recursions* or constructions (rather than number theoretic) could be effected.

Much of this may seem a million miles from the original conceptions of recursive set of integers and universality of the computation/recursion procedure that allowed for the Enumeration and Recursion Theorems *etc.* mentioned above. However these founding theorems return, almost partly as definitions, in the notion that Moschovakis singled out as encompassing a generalised theory of inductive definability, that of the *Spector class*. We give the definition (with some details swept under the rug) as it pertains to inductive definitions at the lowest type, those of sets of integers.

Definition 2. *(Spector Class) (Moschovakis) A class of sets of integers* $\Gamma \subseteq \mathcal{P}(\omega)$ *(or* $\mathcal{P}(\omega^\omega)$*, or direct products of such) is a* Spector Class *if the following are satisfied:*

(i) *(Closure)* Γ *is closed under trivial substitutions and those by functions themselves in* Γ*, and universal quantification over numbers:* $\forall^\omega$*;*

(ii) *(Universality) There is a relation* $R \in \Gamma \cap \omega \times \omega$*, which is* universal *for all relations in* Γ *on* ω*: if* $P \in \mathcal{P}(\omega) \cap \Gamma$ *then* $\exists e \in \omega\ \forall k (k \in P \leftrightarrow (e,k) \in R)$ *(and similarly for other products of* ω *and* ω^ω*).*

(iii) *(Norm Property) For any* $P \in \Gamma$ *there is a function* φ *with* $\varphi : P \longrightarrow On$ *satisfying certain properties close to saying that the prewellordering induced on* P *by* φ *has both its graph and its complement in* Γ*.*

The last property here is somewhat distractingly technical, and so has been left vague, but the point is that the function into the ordinals, gives us a nice prewellordering of the set P and in a very loose sense, we can think of it as saying that an n gets into P before m does if $\varphi(n) < \varphi(m)$. Property (ii) is recognisable as an *Enumeration* property; (i) hints at some initial basic closure properties.

There are countless examples of Spector classes, but the original basic such *ur*-class is obtained by taking the class of Π^1_1 sets of integers (and this is the *least* such class). Such classes relate to notions of definability over admissible sets, since to any Spector class Γ of sets of integers we may find an admissible structure M_Γ over which the sets in Γ correspond to sets inductively definable over M_Γ via some operator Ψ as above.

So, on the one hand, the increasing generalisations of recursion theory to higher types, and especially those including infinite ordinals in their domain, were mostly *mathematical* generalisations rather than machine model generalisations. This might lead one to think that with the increasing sophistication of the approach, that the original intuition of machine or computer had been left far, far behind. However running through this development was always a thread of machine-mindedness. For Kleene Recursion this was for Rogers (1967) the '$\aleph_0$-mind': one could think of Kleene Recursion as the notion of computability that might arise by taking hold of the TM model where a mind could survey the whole tape, as one step; or rewrite an ω-sequence of bits, as one operation, and further consult an oracle A (consisting of a *set* of subsets of $\mathbb{N}$) and receive a 0/1 answer as to whether the *whole set* of integers coded on a tape was or was not in A. In short anything that a mind capable of comprehending, and acting on, an $\aleph_0$=sized amount of information could do. The class of wellfounded 'computation trees' is then a Π^1_1 subset of $\mathcal{P}(\mathbb{N})$ with the latter identified as $\mathbb{R}$. A computational process that outputs, e.g., answers to membership questions about whether the real x belongs to a certain 'generalised r.e' set of real numbers, becomes then the problem of determining the wellfoundedness of a certain tree within any transitive admissible set containing that real x. In a realistic sense (*pace* the countably infinite processes involved) this can still be construed as a machine model.[2]

[2] This would take us too far off course, but even in metarecursion theory, Platek said that he also thought of a formalism involving a Shepherdson–Sturgess–Minsky like register machine containing countable ordinals in its registers, and performing the appropriate machine like instructions (Platek, Private communication). This could have been extended to α-recursion theory.

The recent two decades however has seen an expansion of interest in generalising models of computation, and this has led to mathematical investigations of models of computation where one or more space or time parameters are relaxed: the model of the *Infinite Time Turing Machine* of Hamkins and Lewis (2000) allows time to become transfinite. Time is still considered to tick away in discrete steps for $t = 0, 1, 2, \ldots$, but there are also now limit stages $t = \omega$, then $\omega + 1, \ldots, \omega + \omega, \ldots$ One simply has to specify a behaviour at limit stages of time: thinking of the ordinary Turing machine model, as the program is finite, if the machine does not halt but runs for an ω-sequence of steps, then it is in a program loop; so at time ω put it at the beginning of the outermost loop or subroutine in which it was involved (in other words at the least instruction number it visited infinitely often below ω). What is in the cells? We may specify, given an alphabet of 0's, 1's and B (for blanks) that the i-th cell C_i contains the alphabet symbol j at time ω, if there was a time $t = n < \omega$ so that for all later finite times $m \geq n$ the cell constantly contained a j; however if the cell value changed unboundedly often before ω then let it have value a B for ambiguity at time ω. Where is the R/W head? It at time t cell $C_{k(t)}$ is being read, we may place it at cell $C_{k(\omega)}$ where $k(\omega)$ is defined to be the liminf of the $k(t)$'s for $y < \omega$, *unless* this value is ω itself (because the head has wandered off to infinity). In the latter case we define $k(\omega)$ to be 0. (This accords with our idea of putting the machine at the start of the outermost loop entered into unboundedly often before time ω where possible. Hamkins and Lewis (2000) does this differently, by having three tapes, a 0/1 alphabet only and by taking limsups at limit ordinals, and placing the head back to cell C_0 at all limit times. However mathematically the functions produced are the same.) The same considerations are used at any limit ordinal λ. We may now amuse ourselves by asking any of the myriad questions that have been asked for the standard model TM. What are the 'ITTM'-computable functions produced by such machines? What are the decidable sets of integers? What is the halting set $H = \{P_e(e) \downarrow \mid e \in \mathbb{N}\}$? (Notice that we barely have to change notation to formulate the question.) Now computations may halt *after* stage ω; but at what stages? Since an ITTM can now receive an infinite stream of input it essentially can also compute on reals as well as integers. We can devise oracle machines that, like Kleene Recursion, can quiz a set of reals. What can we do now? Lest one think that this is merely an occupation for an idle hour on a rainy Sunday afternoon, one can show that the classes of ITTM 'semi-decidable' sets of numbers and reals produced form a Spector class. They are thus a particular instance of a higher type recursion theory.

Consider another machine: if we are relaxing time, why not go the whole hog and relax space considerations too? Let us consider an ITTM machine model with an *uncountably* long tape? Or even a tape as long as the class of ordinal numbers. What then can such a machine create? Remarkably there is an *ordinary standard Turing program* that can in effect compute, given some finite number of ordinals (input as 1's at the appropriate ordinal places on the tape) the truth set in the Gödel constructible universe L of the constructible set L-coded by those ordinals. In short, following such considerations, we have another presentation, now a *machine theoretic presentation*, of the set-theoretic Gödel L-hierarchy to set aside those alternatives of Jensen (1972) and Deutsch (1985) Sec. 9 (See Dawson (2009) and Koepke (2005).)

One may wonder at the apparent strength of these machine models, but a moment's reflection shows it to be in the limit rules themselves. We may have ordinary Turing style-action at successor stages, but the limit rule is a kind of infinitary logical rule integrating over this time dimension. It may look innocuous to put B-blanks on tapes at infinite stages, or take a liminf of previous cell values, but it is in these actions that the whole essence of the process inheres.

So what of these 'liminf' processes themselves? Consider then generalisations of the inductive operators defined above. We now define for such a Ψ (no longer required to be monotone):

$$\Psi_0(X) = X; \quad \Psi_{\alpha+1}(X) = \Psi(\Psi_\alpha(X));$$
$$\Psi_\lambda(X) = \liminf_{\alpha \to \lambda} \Psi_\alpha(X) =_{df} \cup_{\alpha<\lambda} \cap_{\lambda>\beta>\alpha} \Psi_\beta(X) \text{ for limit } \lambda.$$

What kind of operators are these? Such do not necessarily reach a fixed point but instead (after countably many iterations) reach a *stability point*: a least stage $\zeta = \zeta(X)$ so that $\Psi_\zeta(X)$ periodically returns for ever after as α runs through all the ordinals. Elementary arguments show that ζ exists and is countable. We thus may develop a theory of such *quasi-inductive definitions*. Starting with the natural numbers the quasi-inductive sets defined by arithmetic (or hyperarithmetic) operators Ψ again form a natural Spector class. Indeed this is to be expected: we can program an ITTM to calculate them.

Other examples may occur to the reader: consider now *Infinite Time Register Machines* (Koepke, 2006; Koepke and Miller, 2008). We allow transfinite time on a standard register machine containing integers. If co-finally at a limit stage a register has become unbounded we by *fiat* reset it to zero, and otherwise register contents contain the liminf of the previous values; the instruction number about to be performed at a limit stage is again the beginning of the outermost routine called unboundedly before the limit time. Now ask the same questions as for ITTM's. There is however a fundamental difference between the Register machine model and the TM model, which does not show up at the finite level. Universality fails, as there is no universal ITRM: as the numbers of registers increase, their strength increases. (In fact the class considered as a whole is weaker by far than the ITTM class.)

Instead of simply asking what the computational power of a transfinite computational model is, one can approach the machine from another direction: that of reverse mathematics (Simpson, 1999). Even the assertion that every ITTM on zero input either halts or loops requires a proof that can be effected only in a substantial fragment of second order number theory: Π^1_2-Comprehension is insufficient, although Π^1_3-Comprehension suffices. For ITRMs the analogous assertion turns out to be equivalent to Π^1_1-CA_0, see Koepke and Welch (2011).

At the risk of a truism, the thing one must always be aware of in considering these generalisations is the infinitary nature of the generalisation: in analysing ITTM's Hamkins and Lewis (2000) stick closely to questions and analogies with Turing jump and degree. However it is the analysis of the new concept in terms of set theory, or already extant higher recursion theory, that renders a full characterisation and delivers the deeper theorems. In the study by Hamkins and Lewis (2000) the analogy with TM's machines was thoroughly pursued, but ultimately the analogy of ITTM-*degree* is closer to that of hyperdegree, or rather Δ^1_2-degree, and the behaviour of such machines is closely tied to that of low levels of the Gödel constructible hierarchy; it is this realisation that enables one to actually answer some of the questions they asked: what are the semi-decidable sets of reals? How long do computations really take? Indeed an analysis of the latter is necessary in order to prove the analogue of Kleene's Normal Form Theorem:

Theorem 3. *(ITTM Normal Form Theorem) (Welch, 2009)*

(a) *For any program index e we may effectively (in the usual sense) find an index e' so that:*

$$\forall x \in 2^{\mathbb{N}} P_e(x) \downarrow \Rightarrow \exists y \in 2^{\mathbb{N}} P_{e'}(x) \downarrow y$$

where y is a code for the whole computation sequence for $P_e(x)$.

(b) *There is an ITTM-decidable universal predicate $\mathfrak{T}_1$, and an arithmetical function U satisfying $\forall e \forall x$:*

$$P_e(x) \downarrow z \leftrightarrow \exists y \in 2^{\mathbb{N}} [\mathfrak{T}_1(e,x,y) \wedge U(y) = z].$$

Note that a y in part (a) has to be a real coding at the very least the ordinal length of the computation $P_e(x)$; in order for there to be any hope of such an e' existing, we need to know that (a code for) the *length* of any computation of the form $P_e(x)$ can itself be computed from something decidable in x. This length is of course a transfinite ordinal, and so we are in a very different ballpark from the original Normal Form Theorem. Fortunately we can, and this length has a characterisation in terms of the L-hierarchy; further, because of this analysis we can determine the decidable, semi-decidable

sets etc. for this model. As evidence that we are doing the right kind of thing we see that we can also obtain a Spector Criterion for the notion of ITTM-reducibility on sets of integers, $x \leq_\infty y$; using ITTM-degrees, and ITTM-jump (denoted x^∇), just as Spector had for hyperdegree:

$$x \leq_\infty y \to (x^\nabla \leq_\infty y \leftrightarrow \zeta^x < \zeta^y)$$

where now ζ^x is the least ordinal after which the universal x-ITTM machine starts repeating.

We mention this here, as it typifies analyses of such proposed machine-theoretically inspired models, that once they are allowed into the transfinite realm, then one uses set-theoretic, or analytical methods to resolve such questions. Once one has started this kind of freeing oneself from the finite realm, one sees all sorts of possibilities: for example consider the Blum-Shub-Smale machine – again something certainly inspired by the Turing model. If we let this run transfinitely what may that compute? Can we think of more general transfinite machines with different and perhaps more complex limit rules? Friedman and Welch (2011) is one attempt to define such machines that run through all the reals of the least β-model of analysis. Can one make sense in general of (some variant of) dynamical systems allowed to run beyond ω? Usually such systems are restricted to continuous functions on some interval or manifold – but what if we consider more general functions in some higher Baire class?

It seems to me that there is little difference in the end between a system of equations recursively applied (in some general sense), and a general machine. However, within the theory of Spector classes and higher type recursion theory, these machine-inspired models occur sporadically as points of illumination, as concrete, and so readily graspable, examples of that rather abstract theory.

And all of this ultimately we have Turing to thank for.

References

Dawson, B., 2009. Ordinal time Turing computation. Ph.D. thesis, Bristol.

Devlin, K., 1984. Constructibility. Perspectives in Mathematical Logic. Springer Verlag, Berlin, Heidelberg.

Friedman, S.D., Welch, P.D., 2011. Hypermachines. J. Symbol. Log. 76 (2), 620–636.

Gandy, R.O., 1988. The confluence of ideas in 1936. In: Herken, (Ed.), The Universal Turing Machine: A Half Century Survey. Oxford University Press, Oxford, pp. 55–111.

Hamkins, J.D., Lewis, A., 2000. Infinite time Turing machines. J. Symb. Log. 65 (2), 567–604.

Jensen, R.B., 1972. The fine structure of the constructible hierarchy. Ann. Math. Log. 4, 229–308.

Kleene, S.C., 1959. Origins of recursive function theory. Ann. Hist. Comput. 3, 52–67.

Koepke, P., 2005. Turing computation on ordinals. Bull. Symb. Log. 11, 377–397.

Koepke, P., 2006. Infinite time register machines. In: Beckmann, A., et al. (Eds.), Logical Approaches to Computational Barriers, vol. 3988 of Springer Lecture Notes Computer Science, Springer, Swansea, pp. 257–266.

Koepke, P., Miller, P., 2008. An enhanced theory of infinite time register machines. In: Beckmann, A., et al. (Ed.), Logic and the Theory of Algorithms, vol. 5028 of Springer Lecture Notes Computer Science, Springer, Swansea, pp. 306–315.

Koepke, P., Welch, P.D., 2011. A generalised dynamical system, infinite time register machines, and Π^1_1-CA_0. In Proceedings of CiE 2011, Sofia, Lecture Notes in Computer Science, Springer.

Odifreddi, P.-G., 1989. Classical Recursion Theory: the theory of functions and sets of natural numbers. Studies in Logic. North-Holland, Amsterdam.

Rogers, H., 1967. Recursive Function Theory. Higher Mathematics. McGraw, 1967.

Rosser, H.B., 1984. Highlights of the history of the λ-calculus. Ann. Hist. Comput., 6 (4), 337–349.

Sieg, W., 1994. Mechanical procedures and mathematical experience. In: George, A. (Ed.), Mathematics and Mind. Oxford University Press, Oxford, New York, Toronto, Tokyo.

Simpson, S., 1999. Subsystems of second order arithmetic. Perspectives in Mathematical Logic. Springer.

Welch, P.D., 2009. Characteristics of discrete transfinite Turing machine models: halting times, stabilization times, and normal form theorems. Theor. Comput. Sci. 410, 426–442.

This page intentionally left blank

A. M. TURING. *On computable numbers, with an application to the Entscheidungsproblem.* ***Proceedings of the London Mathematical Society,*** series 2, vol. 42 (1936–7), pp. 230–265.

The author proposes as a criterion that an infinite sequence of digits 0 and 1 be "computable" that it shall be possible to devise a computing machine, occupying a finite space and with working parts of finite size, which will write down the sequence to any desired number of terms if allowed to run for a sufficiently long time. As a matter of convenience, certain further restrictions are imposed on the character of the machine, but these are of such a nature as obviously to cause no loss of generality – in particular, a human calculator, provided with pencil and paper and explicit instructions can be regarded as a kind of Turing machine. Thus, it is immediately clear that computability, so defined, can be identified with (especially, is no less general than) the notion of effectiveness as it appears in certain mathematical problems (various forms of the Entscheidungsproblem, various problems to find complete sets of invariants in topology, group theory, etc., and in general any problem which concerns the discovery of an algorithm).

The principal result is that there exist sequences (well defined on classical grounds), which are not computable. In particular, the *deducibility problem* of the functional calculus of first order (Hilbert and Ackermann's engere Funktionenkalkül) is unsolvable in the sense that, if the formulas of this calculus are enumerated in a straightforward manner, the sequence whose nth term is 0 or 1, according as the nth formula in the enumeration is or is not deducible, is not computable. (The proof here requires some correction in matters of detail.)

In an appendix, the author sketches a proof of equivalence of "computability" in his sense and "effective calculability" in the sense of the present reviewer (*American Journal of Mathematics*, vol. 58 (1936), pp. 345–363, see review in this Journal, vol. 1, pp. 73–74). The author's result concerning the existence of uncomputable sequences was also anticipated, in terms of effective calculability, in the cited paper. His work was, however, done independently, being nearly complete and known in substance to a number of persons at the time the paper appeared.

As a matter of fact, there is involved here the equivalence of three different notions: computability by a Turing machine, general recursiveness in the sense of Herbrand-Gödel-Kleene and λ-definability in the sense of Kleene and the present reviewer. Of these, the first has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately – i.e., without the necessity of proving preliminary theorems. The second and third have the advantage of suitability for embodiment in a system of symbolic logic.

ALONZO CHURCH

This page intentionally left blank

Computability and λ-Definability

(J. Symbolic Logic, vol. 2 (1937), p. 153–163)

Henk Barendregt, Giulio Manzonetto and Rinus Plasmeijer trace through to today —

THE IMPERATIVE AND FUNCTIONAL PROGRAMMING PARADIGM

1. Models of computation

In Turing (1936) a characterisation is given of those functions that can be computed using a mechanical device. Moreover it was shown that some precisely stated problems cannot be decided by such functions. In order to give evidence for the power of this model of computation, Turing (1937) showed that machine computability has the same strength as definability via λ-calculus, introduced in Church (1936). This model of computation was also introduced with the aim of showing that undecidable problems exist.

Turing machine computability forms a very simple model that is easy to mechanise. Lambda calculus computability, on the other hand, is *a priori* more powerful. Therefore, it is not obvious that it can be executed by a machine. In showing the equivalence of both models, Turing shows us that λ-calculus computations are performable by a machine, so demonstrating the power of Turing machine computations. This gave rise to the combined

Church-Turing Thesis *The notion of intuitive computability is exactly captured by λ-definability or by Turing computability.*

Computability via Turing machines gave rise to imperative programming. Computability described via λ-calculus gave rise to functional programming. As imperative programmes are more easy to run on hardware, this style of software became predominant. We present major advantages of the functional programming paradigm over the imperative one, that are applicable, provided one is willing to explicitly deal with simple abstractions.

2. Functional programming

2.1. *Features from lambda calculus*

Rewriting. Lambda terms form a set of formal expressions subjected to possible *rewriting* (or *reduction*) steps. For each term, there are in general several parts that can be rewritten. However, if there is an eventual outcome, in which there is no more possibility to rewrite, it is necessarily unique.

Application. An important feature of the syntax of λ-terms is *application*. Two expressions can be applied to each other: if F and A are λ-terms, then so is FA, with as intended meaning the function F

Supported in part by NWO Project 612.000.936 CALMOC.

Alan Turing: His Work and Impact. DOI: 10.1016/B978-0-12-386980-7.00004-7

applied to the argument A. Both the function and the argument are given the same status as λ-terms. This implies that functions can be applied to functions, obtaining higher-order functions.

Abstraction. Next to the application there is *abstraction*. This feature allows to create complex functions. For example, given terms F and G intended as functions, then one may form $F \circ G$ and $G \circ F \circ G$ with the rewriting rules

$$(F \circ G)a \;\rightarrow\; F(Ga);$$
$$(G \circ F \circ G)a \;\rightarrow\; G(F(Ga)).$$

It is interesting to note that there is one single mechanism, λ-abstraction, that can capture both examples and much more. Given a λ-term M in which the variable x may occur, one can form the abstraction $\lambda x.M$. It has as intended meaning the function that assigns to x the value M. More generally, $\lambda x.M$ assigns to N the value $M[x{:=}N]$, where the latter denotes the expression obtained by substituting N for x in M. Then one has

$$F \circ G \triangleq \lambda x.F(Gx);$$
$$G \circ F \circ G \triangleq \lambda x.G(F(Gx)).$$

β-reduction. Corresponding to this abstraction with its intended meaning, there is a single rewriting mechanism. It is called β-reduction and is

$$(\lambda x.M)N \rightarrow M[x{:=}N],$$

giving the two rewrite examples mentioned above from the definition of $F \circ G$ and $G \circ F \circ G$. One can iterate the procedure and introduce the higher-order function C 'composition' as follows.

$$C \triangleq \lambda f \lambda g \lambda x.f(gx).$$

Having C one can write $F \circ G \triangleq C\ F\ G$, where in the absence of parentheses, one should read this $C\ F\ G$ as $(C\ F)\ G$. Dually the iterated abstraction $\lambda f \lambda g \lambda x.f(gx)$ should be read as $\lambda f(\lambda g(\lambda x.f(gx)))$.

Instead of λ-abstraction, it is convenient to define functions by their applicative behaviour[1]. One then writes `comp f g x = f (g x)`, obtaining 'composition' `comp f g = f ∘ g`. One can even give definitions that are 'looping', like `L x = (x, L(x + 1))`, so that `L 0 = (0,(1,(2,(3,...))))`. Similarly, one can construct the list of all prime numbers. Infinite lists are easier to describe than a list of say 28 elements.

Lazy evaluation. Expressions are evaluated as little as possible. Consider the program `f (L 4)`: the expression `L 4` is computed only when `f` tries to read some input and is just evaluated for long enough to return a value to `f`. This enables dealing with 'infinite objects', mentioned above.

2.2. *Features beyond lambda calculus*

For the pragmatics of functional programming, several features are added to the basic system of λ-calculus.

Data. Although integers and 'scientific' real numbers can be represented as λ-terms, for efficiency reasons they are given by special constants, together with the primitives for standard operations.

[1] This method is called as *heuristic application principle* by Böhm.

Names. The original λ-calculus formalism does not have names *by design*, as arbitrary λ-abstractions can be made. However, for the pragmatics of using and reusing software components, it is useful to introduce a naming construction `let`. For example, `letcomp` $= \lambda f \lambda g \lambda x.f(gx)$ means that composition $C \triangleq \lambda f \lambda g \lambda x.f(gx)$ is now called `comp` and can be used later to define

$$F \circ G \triangleq \mathtt{comp}\, F\, G.$$

3. Types

In physics, constants have a 'dimension', e.g., speed is measured in kilometre per hour. When we bike at $\mathtt{v} = 12\,\mathtt{km/h}$ and we do this for $\mathtt{t} = 3\,\mathtt{h}$, we have gone $\mathtt{vt} = 12 \cdot 3 = 36\,\mathtt{km}$. Dimensions prevent that we want to consider, e.g., $\mathtt{vt}^2$ to compute the distance.

Similarly, functional programming languages come with a type system helping to ensure correctness. A program expecting a number (`: Int`) should not receive a judgement (`: Bool`). Giving module `F` a type `A` is denoted by `F : A` (read 'F in A'). One starts typing the data, e.g., `3 : Int`, `True : Bool`. Functions with behaviour `G x = y` get as type `A` → `B`, where `x : A` and `y : B`. An application `F a` is only allowed if the types match, i.e., `F : A` → `B` and `a : A`.

The functional programmer indicates the types of the data structures and basic functions, and the machine performs *type inferencing* at compile time. This is a major help for combining software modules in a correct way: many bugs are caught as the result will be an untypable program.

Algebraic Data Types. Next to basic data types, like `Int`, `Bool`, one likes to use common data structures, like lists and trees of elements of type `A`. Such structures start small and grow. There is the empty list `Nil` and one can extend a list `tl` ('tail') by chaining an element `hd` ('head') of type `A`, obtaining `Cons hd tl`. The function that counts the number of elements in a list (of arbitrary type) can be defined by specifying that on the empty list it is 0 and on an enlarged list it is the length of the previous list plus 1.

```
count : List A → Int
count Nil = 0
count (Cons hd tl) = 1 + count tl
```

Using so-called Generalised Abstract Data Types, one introduces several such types simultaneously, mutually depending on each other, keeping type inferencing possible, see Schrijvers et al. (2009).

Generic types. One can 'code' algebraic types enabling to write uniformly functions on these. For example, there is a map on lists and on trees of data (hanging at the leaves), whereby a given function `f` is applied to each element. The two functions can be obtained by specializing one program with a generic type, including possible exceptions. With this feature, one can embed Domain Specific Languages within a functional language, see Plasmeijer et al. (2007).

Dynamic types. A functional program `P` having type `A` is compiled to machine code and evaluated. During this process, the original term, its type and context of definitions are forgotten. Using 'dynamic types', one may keep track of this syntactic data. This enables dynamic code, e.g., for writing typed Operating System in a pure functional language and dealing with unknown plug-ins or database specifications.

Efficiency. Considerable effort has been put into the compiler technologies for functional languages. The code generated by the state of art compilers for Haskell, OCaML and Clean is so good, that efficiency is no longer an issue, see the URL <`shootout.alioth.debian.org`>.

An example. Using abstraction, application, algebraic data types and functions as arguments, one can write compact software that can easily be modified. One can construct `foldr` with the following specification.

```
foldr (•) [a1,...,an] start = (a1 • (a2 • (... (an • start)...)))
```

Here, '•' is a binary operation used in infix position, and '(•)' stands for • as argument. The program

References

Bove, A., Dybjer, P., Norell, U., 2009. A brief overview of Agda – a functional language with dependent types. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M., (Eds.), Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17–20, 2009. Proceedings, volume 5674 of Lecture Notes in Computer Science, Springer, pp. 73–78.

Church, A., 1936. An unsolvable problem of elementary number theory. Am. J. Math. 58 (2), pp. 345–363.

Coq Development Team, 2010. The Coq Proof Assistant Reference Manual – Version V8.3. URL: <http://coq.inria.fr>.

Hammond, K., Michaelson, G., 1999. Research Directions in Parallel Functional Programming, Springer.

Jones, S.P., (Eds.), 2003. Haskell 98 Language and Libraries: The Revised Report, Cambridge University Press.

Leroy, X., 2009. A formally verified compiler back-end. J. Autom. Reason. 43 (4), 363–446.

Marlow, S., Jones, S.L.P. Singh, S., 2009. Runtime support for multicore Haskell. In: Hutton, G., Tolmach, A.P., (Eds.), Proceeding of the 14th ACM SIGPLAN International Conference on Functional Programming, ICFP 2009, August 31–September 2, ACM, Edinburgh, Scotland, UK, pp. 65–78.

McCarthy, J., Abrahams, P.W., Edwards, D.J., Hart, T.P., Levin, M.I., 1962. LISP 1.5 Programmer's Manual, MIT Press.

Milner, R., Tofte, M., Harper, R., McQueen, D., 1997. The Definition of Standard ML, The MIT Press.

Plasmeijer, R.M., Achten, P., Koopman, P.W.M., 2007. iTasks: executable specifications of interactive work flow systems for the web. In: Hinze, R., Ramsey, N., (Eds.), Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007, October 1–3, 2007, ACM, Freiburg, Germany, pp. 141–152.

Plasmeijer, R.M., van Eekelen, M.C.J.D., 2002. Clean Language Report, University of Nijmegen, Software Technology Group.

Schrijvers, T., Jones, S.L.P., Sulzmann, M., Vytiniotis, D., 2009. Complete and decidable type inference for GADTs. In: Hutton, G., Tolmach, A.P., (Eds.), Proceeding of the 14th ACM SIGPLAN International Conference on Functional Programming, ICFP 2009, August 31–September 2, 2009, ACM, Edinburgh, Scotland, UK, pp. 341–352.

Syme, D., Granicz, A., Cisternino, A., 2007. Expert F#, Reactive, Asynchronous and Concurrent Programming. Apress, 2007.

Turing, A.M., 1936. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. ser. 2. 24, 230–265.

Turing, A.M., 1937. Computability and lambda-definability. J. Symbol. Log. 2 (4),153–163.

Turner, D.A., 1985. Miranda: A non-strict functional language with polymorphic types. Proceedings IFIP Conference on Functional Programming Languages and Computer Architecture, Nancy, France, September 1985. Springer Lecture Notes in Computer Science 201, pp 1-16.

COMPUTABILITY AND λ-DEFINABILITY

A. M. TURING

Several definitions have been given to express an exact meaning corresponding to the intuitive idea of 'effective calculability' as applied for instance to functions of positive integers. The purpose of the present paper is to show that the computable[1] functions introduced by the author are identical with the λ-definable[2] functions of Church and the general recursive[3] functions due to Herbrand and Gödel and developed by Kleene. It is shown that every λ-definable function is computable and that every computable function is general recursive. There is a modified form of λ-definability, known as λ-*K*-definability, and it turns out to be natural to put the proof that every λ-definable function is computable in the form of a proof that every λ-*K*-definable function is computable; that every λ-definable function is λ-*K*-definable is trivial. If these results are taken in conjunction with an already available[4] proof that every general recursive function is λ-definable we shall have the required equivalence of computability with λ-definability and incidentally a new proof of the equivalence of λ-definability and λ-*K*-definability.

A definition of what is meant by a computable function cannot be given satisfactorily in a short space. I therefore refer the reader to *Computable* pp. 230–235 and p. 254. The proof that computability implies recursiveness requires no more knowledge of computable functions than the ideas underlying the definition: the technical details are recalled in §5. On the other hand in proving that the λ-*K*-definable functions are computable it is assumed that the reader is familiar with the methods of *Computable* pp. 235–239.

The identification of 'effectively calculable' functions with computable functions is possibly more convincing than an identification with the λ-definable or general recursive functions. For those who take this view the formal proof of equivalence provides a justification for Church's calculus, and allows the 'machines' which generate computable functions to be replaced by the more convenient λ-definitions

1. Definition of λ-*K*-definability

In this section the notion of λ-*K*-definability is introduced in a form suitable for handling with machines. There will be three differences from the normal, in addition to that which distinguishes λ-*K*-definability from λ-definability. One consists in using only one kind [ ] of bracket instead of three, {}, (), [ ]; another is that $x, x', x'', \cdots$ are used as variables instead of an indefinite infinite

Received September 11, 1937.

[1] A. M. Turing, *On computable numbers with an application to the Entscheidungsproblem,* ***Proceedings of the London Mathematical Society*****, ser. 2,** vol. 42 (1936–7), pp. 230–265, quoted here as *Computable*. A similar definition was given by E. L. Post, *Finite combinatory processes—formulation* 1, this JOURNAL, vol. 1 (1936), pp. 103–105.

[2] Alonzo Church *An unsolvable problem of elementary number theory*, ***American journal of mathematics***, vol. 58 (1936), pp. 345–363, quoted here as *Unsolvable*.

[3] S. C. Kleene, *General recursive functions of natural numbers*, ***Mathematische Annalen***, vol. 112 (1935–6), pp. 727–742. A definition of general recursiveness is also to be found in *Unsolvable* pp. 350–351.

[4] S. C. Kleene. *λ-definability and recursiveness,* ***Duke mathematical journal***, vol. 2 (1936), pp. 340–353.

list of the single symbols, and the third is a change in the form of condition (ii) of immediate transformability, not affecting the definition of convertibility except in form.

There are five symbols which occur in the formulae of the conversion calculus. They are λ, x, |, [and]. A sequence of symbols consisting of x followed by | repeated any number (possibly 0) of times is called a *variable. Properly- formed formulae* are a class of sequences of symbols which includes all variables. Also if ***M*** and ***N*** are[5] properly-formed formula, then [***M***][***N***] (i.e. the sequence consisting of [followed by ***M*** then by], [and the sequence ***N***, and finally by]) is a properly-formed formula. If ***M*** is a properly-formed formula and ***V*** is a variable, then λ***V***[***M***] is a properly-formed formula. If any sequence is a properly-formed formula it must follow that it is so from what has already been said.

A properly-formed formula ***M*** will be said to be *immediately transformable* into ***N*** if either:

(i) ***M*** is of the form λ***V***[***X***] and ***N*** is λ***U***[***Y***] where ***Y*** is obtained from ***X*** by replacing the variable ***V*** by the variable ***U*** throughout, provided that ***U*** does not occur in ***X***.

(ii) ***M*** is of the form [λ***V***[***X***]][***Y***] where ***V*** is a variable and ***N*** is obtained by substituting ***Y*** for ***V*** throughout ***X***. This is to be subject to the restriction that if ***W*** be either ***V*** or a variable occurring in ***Y***, then λ***W*** must not occur in ***X***.

(iii) ***N*** is immediately transformable into ***M*** by (ii).

A will be said to be immediately convertible into ***B*** if ***A*** is immediately transformable into ***B*** or if ***A*** is of the form ***X***[***L***]***Y*** and ***B*** is ***X***[***M***]***Y*** where ***L*** is immediately transformable into ***M***. Either ***X*** or ***Y*** may be void. ***A*** is *convertible* to ***B*** (***A*** conv ***B***) if there is a finite sequence of properly-formed formulae, beginning with ***A*** and terminating with ***B***, each immediately convertible into the preceding.

The formulae,

λx[λx$^{\|}$ [x$^{\|}$]]	(abbreviated to 0),
λx[λx$^{\|}$ [[x][x$^{\|}$]]]	(abbreviated to 1),
λx[λx$^{\|}$ [[x][[x][x$^{\|}$]]]]	(abbreviated to 2), etc.,

represent the natural numbers. If ***n*** represents a natural number then the next natural number is represented by a formula which is convertible to [***S***][***n***] where ***S*** is

$$\lambda x^{||}[\lambda x\ [\lambda x^{|}[[x][[[x^{||}][x]][x^{|}]]]]].$$

A function $f(n)$ of the natural numbers, taking natural numbers as values will be said to be λ-*K*-definable if there is a formula ***F*** such that [***F***][***n***] is convertible to the formula representing $f(\boldsymbol{n})$ if ***n*** is the formula representing n. The formula [***F***][***n***] can never be convertible to two formulae representing different natural numbers, for it has been shown[6] that if two properly-formed formulae are in normal form (i.e., have no parts of the form [λ***V***[***M***]][***N***]) and are convertible into one another, then the conversion can be carried out by the use of (i) only. The formulae representing the natural numbers are in normal form and the formulae representing two different natural numbers are certainly not convertible into one another by the use of (i) alone.

---

[5] Heavy type capitals are used to stand for variable or undetermined sequences of symbols. In expressions involving brackets and heavy type letters it is to be understood that the possible substitutions of sequences of symbols for these letters is to be subject to the restriction that the pairing of the explicitly shown brackets is unaltered by the substitution; thus in ***X***[***L***]***Y*** the number of occurrences of [in ***L*** must equal the number of occurrences of].

[6] Alonzo Church and J. B. Rosser, *Some properties of conversion*, ***Transactions of the American Mathematical Society***, vol. 39 (1936), pp. 472–482. The result used here is Theorem 1 Corollary 2 as extended to the modified conversion on p. 482.

2. Abbreviations

A number of abbreviations of the same character as those in *Computable* (pp. 235–239) are introduced here. They will be applied in connection with the calculus of conversion, but are necessary for other purposes, e.g. for carrying out the processes of any ordinary formal logic with machines. The abbreviations in *Computable* are taken as known.

'The sequence of symbols marked with α (followed by α)' will be abbreviated to S(α) in the explanations. Sequences are normally identified by the way they are marked, and are as it were lost when their marks are erased.

In the tables B will be used as a name for the symbol 'blank.'

$\mathfrak{pem}(\mathfrak{A}, \alpha, \beta)$		$\mathfrak{pe}(\mathfrak{pem}_1, \alpha)$
$\mathfrak{pem}_1$	$R, P\beta$	$\mathfrak{A}$

$\mathfrak{pem}_1$ here stands for $\mathfrak{pem}_1(\mathfrak{A}, \alpha, \beta)$ and similar abbreviations must be understood throughout.

$\mathfrak{pem}(\mathfrak{A}, \alpha, \beta)$. The machine prints α at the end of the sequence of symbols on F-squares and marks it with β. $\rightarrow \mathfrak{A}$.

The tables for $\mathfrak{crm}$ $(\mathfrak{B}, \gamma, \beta)$ and $\mathfrak{cem}(\mathfrak{B}, \gamma, \beta)$ are to be obtained from those for $\mathfrak{cr}(\mathfrak{B}, \gamma)$ and $\mathfrak{ce}(\mathfrak{B}, \gamma)$ by replacing $\mathfrak{pe}(\mathfrak{A}, \alpha)$ by $\mathfrak{pem}(\mathfrak{A}, \alpha, \beta)$ throughout.

$\mathfrak{cpr}(\mathfrak{A}, \mathfrak{C}, \alpha, \beta)$	$\mathfrak{cp}(\mathfrak{cpr}_1, \mathfrak{cpr}_2, \mathfrak{cpr}_3, \alpha, \beta)$
$\mathfrak{cpr}_1$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{cpr}, \mathfrak{b}, \beta, b), \mathfrak{b}, \alpha, a)$
$\mathfrak{cpr}_2$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{C}, b, \beta), a, \alpha)$
$\mathfrak{cpr}_3$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{A}, b, \beta), a, \alpha)$

$\mathfrak{cpr}(\mathfrak{A}, \mathfrak{C}, \alpha, \beta)$. The machine compares S(α) with S(β). $\rightarrow \mathfrak{A}$ if they are alike; $\rightarrow \mathfrak{C}$ otherwise. No erasures are made.

The letters a, b occurring in the table for $\mathfrak{cpr}$ should not be used elsewhere in any machine whose table involves $\mathfrak{cpr}$. This can be made automatic by using $a_{\mathfrak{cpr}}$ and $b_{\mathfrak{cpr}}$ say, instead of a and b. We shall however write a and b and understand them to mean $a_{\mathfrak{cpr}}$ and $b_{\mathfrak{cpr}}$. The same applies for the letters $a, \cdots, z$ in all such tables.

$\mathfrak{f}(\mathfrak{A}, \gamma)$	γ	L	$\mathfrak{A}$
	not γ	R, R	$\mathfrak{f}$
$\mathfrak{bf}(\mathfrak{A}, \alpha, \beta, \gamma, \delta)$		E, Pa	$\mathfrak{f}(\mathfrak{bf}_1, \gamma)$
$\mathfrak{bf}_1$	α	R, E, Pb	$\mathfrak{f}(\mathfrak{bf}_2, \gamma)$
	β	L	$\mathfrak{bf}_3$
	others	$R, R, R,$	$\mathfrak{f}(\mathfrak{bf}_1, \mathfrak{b}, \gamma)$
$\mathfrak{bf}_2$	β	R, E, Pb	$\mathfrak{f}(\mathfrak{bf}, \mathfrak{b}, a)$
	not β	R, R, R	$\mathfrak{f}(\mathfrak{bf}_2, \gamma)$
$\mathfrak{bf}_3$	γ or b	$E, P\delta, L, L$	$\mathfrak{bf}_3$
	α	$E, P\gamma$	$\mathfrak{A}$
	others	L, L	$\mathfrak{bf}_3$

$\mathfrak{bf}(\mathfrak{A}, \alpha, \beta, \gamma, \delta)$. This describes the process of finding the partner of a bracket. If α and β are regarded as left and right brackets, then if the machine takes up the internal configuration $\mathfrak{bf}$ when scanning a square next on the right of an α it will find the partner of this α in the sequence $S(\gamma)$, and will mark the part of $S(\gamma)$ which is between the brackets with δ (instead of γ). The final internal configuration is $\mathfrak{A}$ and the scanned square is that which was scanned when the internal configuration $\mathfrak{bf}$ was first taken up.

$\mathfrak{sb}(\mathfrak{A},\alpha,\beta,\gamma,\delta,\epsilon)$			$f'(\mathfrak{sb}_1,\mathfrak{crm}(\mathfrak{re}(\mathfrak{re}(\mathfrak{sb},a,j)b,\beta)\gamma,\epsilon),\beta)$
$\mathfrak{sb}_1$	σ	$R,\ E,\ Pb$	$\mathfrak{sb}_2(\mathfrak{A},\ \alpha,\ \beta,\ \gamma,\ \delta,\ \epsilon,\ \sigma)$
$\mathfrak{sb}_2$			$\mathfrak{f}'(\mathfrak{sb}_3,\mathfrak{crm}(\mathfrak{re}(\mathfrak{re}(\mathfrak{re}(\mathfrak{A},\mathfrak{b},\beta),j,\alpha),a,\alpha),\ a,\delta),\ \alpha)$
$\mathfrak{sb}_3$	σ	R,E,Pa	$\mathfrak{sb}$
	not σ	R,E,Pa	$\mathfrak{re}\ (\mathfrak{f}'(\mathfrak{sb}_4,\ \mathfrak{b},\ a)_{\mathfrak{b}},\beta)$
$\mathfrak{sb}_4$	τ	R,E,Pj	$\mathfrak{re}(\mathfrak{pem}(\mathfrak{sb},\tau,\delta),\ a,\alpha)$
$\mathfrak{sub}(\mathfrak{A},\alpha,\beta,\gamma,\delta)$			$\mathfrak{sb}(\mathfrak{A},\alpha,\beta,\gamma,\ \delta,\delta)$
$\mathfrak{dt}(\mathfrak{A},\ \mathfrak{B},\alpha,\beta)$			$\mathfrak{pem}(\mathfrak{sb}(\mathfrak{f}(\mathfrak{e}(\mathfrak{A},d),\mathfrak{B},\ d),\ \alpha,\beta,p,r,\ d),\ r,p)$

$\mathfrak{sub}(\mathfrak{A},\ \alpha,\beta,\ \gamma,\ \delta)$. $S(\gamma)$ is substituted for $S(\beta)$ throughout $S(\alpha)$. The result is copied down and marked with δ. $\rightarrow \mathfrak{A}$.

$\mathfrak{dt}(\mathfrak{A},\ \mathfrak{B},\ \alpha,\beta)$. It is determined whether the sequence $S(\beta)$ occurs in $S(\alpha)$. $\rightarrow \mathfrak{A}$ if it does; $\rightarrow \mathfrak{B}$ otherwise.

The tables which follow are particularly important in all cases where an enumeration of all possible results of operations of given types is required. The enumeration may be carried out by regarding the operation as determined by a number of choices, each between two possibilities, **L** and **M** say. Each possible sequence of operations is then associated with a finite sequence of letters **L** and **M**. These sequences can easily be enumerated. The method used here is to replace **L** by 0, each **M** by 1, follow the whole by 1, reverse the order and regard the result as the binary Arabic numeral corresponding to the given sequence. Thus the first few sequences (beginning with the one associated with 1) are: the null sequence, **L**, **M**, **LL**, **ML**, **LM**, **MM**, **LLL**, **MLL**, **LML**, **MML**. In the general table below ζ and η are used instead of **L** and **M**.

$\mathfrak{add}(\mathfrak{A},\alpha,\zeta,\eta)$			$\mathfrak{f}'(\mathfrak{add}_1,\mathfrak{pem}(\mathfrak{add}_2,\zeta,a),\alpha)$
$\mathfrak{add}_1$	η	R,E	$\mathfrak{pem}(\mathfrak{add},\ \zeta,a)$
	ζ	R,E	$\mathfrak{pem}(\mathfrak{add}_2,\ \eta,a)$
$\mathfrak{add}_2$			$\mathfrak{cem}(\mathfrak{re}(\mathfrak{A},\ a,\ \alpha),\ \alpha,a)$

$\mathfrak{add}(\mathfrak{A},\ \alpha,\zeta,\ \eta)$. The sequence $S(\alpha)$ consisting of letters ζ and η only is transformed into the next sequence. $\rightarrow \mathfrak{A}$.

$\mathfrak{ch}(\mathfrak{A},\ \mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta)$			$\mathfrak{f}'(\mathfrak{ch}_1,\mathfrak{re}(\mathfrak{C},b,\alpha),\alpha)$
$\mathfrak{ch}_1$	ζ	R,E,Pb	$\mathfrak{A}$
	η	R,E,Pb	$\mathfrak{B}$

$\mathfrak{ch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta)$ is an internal configuration which is taken up when a choice has to be made. $S(\alpha)$ is the sequence of letters ζ and η determining the choices. $\rightarrow \mathfrak{A}$ if the first unused letter is ζ; $\rightarrow \mathfrak{B}$

determined whether the immediate transformation (ii) is permissible: if it is then $\mathfrak{red}_{10}$ is taken up and the substitution carried out.

$\mathfrak{imc}(\mathfrak{A},\mathfrak{C},\alpha,\beta,\theta)$			$\mathfrak{ch}(\mathfrak{f}'(\mathfrak{imc}_1,\ \mathfrak{imc}_2,\ \alpha),\mathfrak{re}(\mathfrak{imc}_1,\alpha,a),\ \mathfrak{C},\ \theta)$
$\mathfrak{imc}_1$,	[	R,E,Pc	$\mathfrak{ch}(\mathfrak{imc},\ \mathfrak{imc}_3,\ \mathfrak{C},\theta)$
	not [	R,E,Pc	$\mathfrak{imc}$
$\mathfrak{imc}_2$			$\mathfrak{re}(\mathfrak{crm}(\mathfrak{A},\alpha,\beta),c,\alpha)$
$\mathfrak{imc}_3$			$\mathfrak{q}(\mathfrak{bf}(\mathfrak{imc}_4,\ [,],\ \alpha,a),\ c)$
$\mathfrak{imc}_4$			$\mathfrak{ch}(\mathfrak{vc},\mathfrak{ch}(\mathfrak{rc},\mathfrak{ex},\ \mathfrak{C},\theta),\ \mathfrak{C},\ \theta)$
$\mathfrak{vc}$			$\mathfrak{va}(\mathfrak{imc}_5,\mathfrak{C},a,b,\theta)$
$\mathfrak{rc}$			$\mathfrak{red}(\mathfrak{imc}_5,a,b)$
$\mathfrak{er}$			$\mathfrak{pff}(\mathfrak{red}(\mathfrak{er}_1,b,d),\mathfrak{G},b,\theta)$
$\mathfrak{er}_1$			$\mathfrak{cpr}(\mathfrak{e}(\mathfrak{imc}_5,d),\mathfrak{e}(\mathfrak{e}(\mathfrak{crm}(\mathfrak{imc}_5,a,b),d),b),d,a)$
$\mathfrak{imc}_5$			$\mathfrak{crm}(\mathfrak{crm}(\mathfrak{crm}(\mathfrak{re}(\mathfrak{re}(\mathfrak{e}(\mathfrak{A},b),c,\alpha),a,\alpha),\alpha,\beta),b,\beta),c,\beta)$

$\mathfrak{imc}(\mathfrak{A},\ \mathfrak{C},\alpha,\beta,\ \theta)$. An immediate conversion is chosen and performed on $\mathrm{S}(\alpha)$. The result is marked with $\beta.\ \rightarrow \mathfrak{A}$.

$\mathfrak{conv}(\mathfrak{A},\alpha,\beta,\theta)$			$\mathfrak{pe}(\mathfrak{crm}(\mathfrak{conv}_1,\alpha,d),.)$
$\mathfrak{conv}_1$			$\mathfrak{ch}(\mathfrak{imc}(\mathfrak{conv}_2,\mathfrak{au},d,f,\theta),\mathfrak{re}(\mathfrak{A},d,\beta),\mathfrak{au},\theta$
$\mathfrak{conv}_2$			$\mathfrak{e}(\mathfrak{re}(\mathfrak{conv}_1,f,d),d)$
$\mathfrak{au}$			$\mathfrak{q}(\mathfrak{au}_1,.)$
$\mathfrak{au}_1$	$\mathfrak{B}$		$\mathfrak{crm}(\mathfrak{A},\alpha,\beta)$
	not $\mathfrak{B}$	R,E,R	$\mathfrak{au}_1$

$\mathfrak{conv}(\mathfrak{A},\alpha,\beta,\theta)$. A conversion is chosen and performed on $\mathrm{S}(\alpha)$. The result is marked with $\beta. \rightarrow \mathfrak{A}$. The sequence determining the choices is $\mathrm{S}(\theta)$. If it should happen that this sequence is exhausted before the conversion is completed then the final formula is the same as the original, i.e. $\mathrm{S}(\alpha)$. The half finished conversion work is effectively removed from the tape by erasing the marks.

4. Computability of λ-*K*-definable functions

It is now comparatively simple to show that a λ-*K*-definable function is computable, i.e., that[7] if $f(n)$ is λ-K-definable then the sequence γ_f in which there are $f(n)$ figures 1 between the nth and the $(n+1)$th 0, and $f(0)$ figures before the first 0, is computable.

To simplify the table for the machine which computes γ_f we use the abbreviation $\mathfrak{Wr}(\mathfrak{A},\ \boldsymbol{M},\ \alpha)$ for an internal configuration starting from which the machine writes the sequence $\boldsymbol{M}$ of symbols at the end, marking it with α and finishing in the internal configuration $\mathfrak{A}$. Thus the table for $\mathfrak{Wr}(\mathfrak{A},\lambda\mathrm{x}^{|},\ \alpha)$ would be:

$\mathfrak{Wr}(\mathfrak{A},\ \lambda\mathrm{x}\ ^{|},\ \alpha)$		$\mathfrak{pe}(\mathfrak{Wr}_1,\ \mathfrak{B})$
$\mathfrak{Wr}_1$,	$P\lambda,\ R,P\alpha,R,\ P\mathrm{x},R,P\alpha,R,\ P^{|},\ R,P\alpha$	$\mathfrak{A}$

[7] Computable p. 254.

2.3. *Strong normalisation*

Actually the simply typed lambda calculus enjoys strong normalisation, which means that all β-reductions are terminating regardless of the strategy that is chosen. The classic proof of strong normalisation by using the reducibility technique is due to the study by Tait (1967), already obtained in 1963 and used by many authors. The proof of strong normalisation by Tait does not use a complexity measure assigned to terms. In the study by de Vrijer (1987), it is shown that it is possible to do this, assigning to a term M an ordinal $|M|$ (in fact a natural number), in such a way that $M \to_\beta N$ entails $|M| > |N|$, regardless what redex is reduced. It is an open problem whether such ordinals can be assigned in a natural and simple way.

3. Postscript

Lambda calculus was more often on Turing's mind. The logician Robin Gandy, who had been a student and associate of Turing, mentioned in 1986 at a conference for his retirement that in the early 1950s, Turing had told him ideas to implement lambda reduction using graphs. This is now commonly done when designing compilers for functional programming languages. Thereby, Turing was not careful about the distinction between free and bound variables and Gandy could correct him. Then Turing said: "That remark is worth 10 pounds a week!", in those days enough for a decent living.

References

Barendregt, H.P., 1984. The lambda calculus, its syntax and semantics, 2nd ed. No. 103 in Studies in Logic and the Foundations of Mathematics, North-Holland.

Barendregt, H.P., Manzonetto, G., Plasmeijer, M.J., 2012. The imperative and functional programming paradigm. In: Cooper, B., van Leeuwen, J. (Eds.), This volume, Elsevier, pp. 121–126.

de Vrijer, R.C., 1987. Exactly estimating functionals and strong normalization. Indagat. Math. 49, 479–493.

Endrullis, J., Hendriks, D., Klop, J.W., 2010. Modular construction of fixed point combinators and clocked Böhm trees. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11–14 July 2010, IEEE Computer Society, Edinburgh, United Kingdom, pp. 111–119.

Gandy, R.O., 1980. An early proof of normalization by A. M. Turing. In: Seldin, J.P., Hindley, J.R. (Eds.), To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press Limited, pp. 453–455.

Lévy, J.-J., 1978. Réductions correctes et optimales dans le lambda-calcul. Ph.D. thesis, Université Paris 7.

Tait, W.W., 1967. Intentional interpretation of functionals of finite type I. J. Symbol. Log. 32 (2), 198–212.

Turing, A.M., 1937. Computability and λ-definability. J. Symbolic Logic 2 (4), 1937, pp. 153–163.

Turing, A.M., 1937. The ƥ-function in lambda-K-conversion. J. Symbol. Log. 2 (4), 164.

von Plato, J., 2008. Gentzen's proof of normalization for natural deduction. Bull. Symbol. Log. 14 (2), 240–257.

Systems of Logic Based on Ordinals

(Proc. Lond. Math. Soc., series 2 vol. 45 (1939), pp. 161–228)

Solomon Feferman returns to —

TURING'S THESIS: ORDINAL LOGICS AND ORACLE COMPUTABILITY

In the sole extended break from his life and varied career in England, Alan Turing spent the years 1936–8 doing graduate work at Princeton University under the direction of Alonzo Church, the doyen of American logicians. Those two years sufficed for him to complete a thesis and obtain the Ph.D. The results of the thesis were published in 1939 under the title, 'Systems of logic based on ordinals' (Turing, 1939). That was the first systematic attempt to deal with the natural idea of overcoming the Gödelian incompleteness of formal systems by iterating the adjunction of statements – such as the consistency of the system – that 'ought to' have been accepted but were not derivable; in fact, these kinds of iterations can be extended into the transfinite. As Turing put it beautifully in his introduction (Turing, 1939):

> The well-known theorem of Gödel (1931) shows that every system of logic is in a certain sense incomplete, but at the same time it indicates means whereby from a system L of logic a more complete system L' may be obtained. By repeating the process we get a sequence L, $L_1 = L'$, $L_2 = L'_1, \ldots$ each more complete than the preceding. A logic L_ω may then be constructed in which the provable theorems are the totality of theorems provable with the help of the logics L, L_1, $L_2, \ldots$ Proceeding in this way we can associate a system of logic with any constructive ordinal. It may be asked whether such a sequence of logics of this kind is complete in the sense that to any problem A there corresponds an ordinal α such that A is solvable by means of the logic L_α.

Using an ingenious argument in pursuit of this aim, Turing obtained a striking yet equivocal partial completeness result that clearly called for further investigation. But he did not continue that himself, and it would be some twenty years before the line of research he inaugurated would be renewed by others. The paper itself received little attention in the interim, though it contained a number of original and stimulating ideas, and though Turing's name had by then been well established through his earlier work on the concept of effective computability. One of those ideas is that of oracle computability, addressed elsewhere in this volume.

Here, in brief, is the story of what led Turing to Church, what was in his thesis, and what came after, both for him and for the subject.[1]

[1] Much of this note is adapted directly from my paper (Feferman, 2006) for the *Notices* of the American Mathematical Society. Prior to that I had written about this material at somewhat greater length in my work (Feferman, 1988), and that in turn was incorporated as an introductory note to Turing's 1939 paper in the volume, *Mathematical Logic* (Turing, 2001) of his collected works. In its biographical part, I have drawn to a considerable extent on Andrew Hodges' superb biography, *Alan Turing: The Enigma* (Hodges, 1994).

Alan Turing: His Work and Impact. DOI: 10.1016/B978-0-12-386980-7.00006-0

misguided. His aim was to produce an arithmetical problem which is not number-theoretical in his sense, i.e., not in $\forall\exists$-form. This is trivial by a cardinality argument, since there are only countably many effective relations $R(x,y)$ of which we could say that $\forall x \exists y R(x, y)$ holds. Turing's way of dealing with this, instead, is through the new notion of computation relative to an *oracle*. As he puts it:

> Let us suppose that we are supplied with some unspecified means of solving number-theoretical [i.e., $\forall\exists$] problems; a kind of oracle as it were. ... 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.

He then showed that the problem of determining whether an o-machine terminates on any given input is an arithmetical problem not computable by any o-machine, and hence not solvable by the oracle itself. Turing did nothing further with the idea of o-machines, either in this paper or afterward. Post (1944) took it as his basic notion for a theory of *degrees of unsolvability*, crediting Turing with the result that for any set of natural numbers there is another of higher degree of unsolvability. This transformed the notion of computability from an absolute notion into a relative notion that would lead to entirely new developments and eventually to vastly generalised forms of recursion theory.[3]

4. Ordinal logics redux

The problems left open in Turing's thesis were attacked in my 1962 paper, 'Transfinite recursive progressions of axiomatic theories' (Feferman, 1962). The title contains my rechristening of 'ordinal logics' in order to give a more precise sense of the subject matter. I showed there that Turing's progression based on iteration of consistency statements is not complete for true $\forall\exists$ statements, contrary to his hope. In fact, the same holds for the even stronger progression obtained by iterating adjunction to a system S of the *local reflection principle for S*. This is a scheme that formalises, for each arithmetical sentence A, that if A is provable in S then A holds. The *uniform reflection principle* is a generalisation of the local principle to arbitrary formulas. Then, I showed that a progression based on the iteration of that is complete for all true arithmetical sentences. One can also find a path P through O along which every true arithmetical sentence is provable in that progression. On the other hand, invariance fails badly in the sense that there are paths P' through O for which there are true sentences in $\forall$-form not provable along that path, as shown in my paper with Clifford Spector (Feferman and Spector, 1962). The book *Inexhaustibility* (Franzén, 2004a) by Torkel Franzén contains an accessible introduction to Feferman (1962), and his paper Franzén (2004b) gives an interesting explanation of what makes Turing's and my completeness results work.

The problem raised by Turing of recognising which expressions (or numbers) are actually notations for ordinals is dealt with in part through the concept of *autonomous progressions of theories*, obtained by imposing a boot strapping process. That allows one to go to a system S_a only if one already has a proof in a previously accepted system S_b that $a \in O$ (or that a recursive ordering of order type corresponding to a is a well-ordering). Such progressions are not complete but have been used to propose characterisations of certain informal concepts of proof, such as that of finitist proof (Kreisel, 1960, 1970) and predicative proof (Feferman, 1964). For more recent progress that replaces the use of transfinite progressions via a concept of the *unfolding* of formal systems based on suitable *axiom schemata*, see my article (Feferman, 1996), and (Feferman and Strahm, 2000).

[3] See the relevant pieces in other parts of this volume. I have written at greater length of the significance of oracle computability from several perspectives in my paper (Feferman, 1992).

References

Church, A., 1936a. A note on the Entscheidungsproblem, J. Symbolic Logic 1, 40–41; correction, ibid., 101–102.

Church, A., 1936b. An unsolvable problem of elementary number theory. Amer. J. Math. 58, 345–363.

Church, A., Kleene, S.C., 1936. Formal definitions in the theory of ordinal numbers. Fundamenta Mathematicae 28, 11–21.

Feferman, S., 1962. Transfinite recursive progressions of axiomatic theories. J. Symbolic Logic 27, 259–316.

Feferman, S., 1964. Systems of predicative analysis. J. Symbolic Logic 29, 1–30.

Feferman, S., 1988. Turing in the land of O(z). In: Herken, R. (Ed.), The Universal Turing Machine. A Half-Century Survey, Oxford University Press, Oxford, pp. 113–147.

Feferman, S., 1992. Turing's 'oracle': From absolute to relative computability?—and back. In: Echeverria, J., et al. (Eds.), The Space of Mathematics, Walter de Gruyter, Berlin, pp. 314–348

Feferman, S., 1996. Gödel's program for new axioms: Why, where, how and what?. In: Hajek, P. (Ed.). Gödel '96, Lecture Notes in Logic 6, 3–22.

Feferman, S., 2006. Turing's thesis. Notices Amer. Math. Soc. 53 (10) 1200–1205.

Feferman, S., Spector, C., 1962. Incompleteness along paths in recursive progressions of theories. J. Symbolic Logic 27, 383–390.

Feferman, S., Strahm, T., 2000. The unfolding of non-finitist arithmetic. Ann. Pure and Applied Logic 104, 75–96.

Franzén, T., 2004a. Inexhaustibility. A Non-Exhaustive Treatment, Lecture Notes in Logic 28. Assoc. for Symbolic Logic, A. K. Peters, Ltd., Wellesley (distribs.).

Franzén, T., 2004b. Transfinite progressions: a second look at completeness. Bull. Symbolic Logic 10, 367–389.

Hodges, A., 1994. Alan Turing: The Enigma, Simon and Schuster, New York 1983, rev. ed. Springer-Verlag, New York.

Kreisel, G., 1960. Ordinal logics and the characterization of informal concepts of proof. In: Proc. International Congress of Mathematicians at Edinburgh 1958, Cambridge Univ. Press, New York, pp. 289–299.

Kreisel, G., 1970. Principles of proof and ordinals implicit in given concepts. In: Myhill, J., et al. (Eds.), Intuitionism and Proof Theory, North-Holland, Amsterdam, pp. 489–516.

Post, E., 1944. Recursively enumerable sets and their decision problems. Bull. Amer. Math. Soc. 50, 284–316.

Turing, A.M., 1936–7. On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. 42 (2), 230–265; A correction, ibid. 43, 544–546. Reprinted in Turing (2001).

Turing, A.M., 1939. Systems of logic based on ordinals. Proc. London Math. Soc. (2), 161–228. Reprinted in Turing (2001).

Turing, A.M., 2001. Mathematical Logic. In: Gandy, R.O., Yates, C.E.M. (Eds.), Collected Works of A.M. Turing, Elsevier Science Publishers, Amsterdam.

1. The calculus of conversion. Gödel representations.

It will be convenient to be able to use the "conversion calculus" of Church for the description of functions and for some other purposes. This will make greater clarity and simplicity of expression possible. I give a short account of this calculus. For detailed descriptions see Church [3], [2], Kleene [1], Church and Rosser [1].

The formulae of the calculus are formed from the symbols {, }, (,), [,], λ, δ, and an infinite list of others called variables; we shall take for our infinite list a, b, ..., z, x', x'', ... Certain finite sequences of such symbols are called *well-formed formulae* (abbreviated to W.F.F.); we define this class inductively, and define simultaneously the free and the bound variables of a W.F.F. Any variable is a W.F.F. ; it is its only free variable, and it has no bound variables. δ is a W.F.F. and has no free or bound variables. If **M** and **N** are W.F.F. then {**M**}(**N**) is a W.F.F., whose free variables are the free variables of **M** together with the free variables of **N**, and whose bound variables are the bound variables of **M** together with those of **N**. If **M** is a W.F.F. and **V** is one of its free variables, then λ**V**[**M**] is a W.F.F. whose free variables are those of **M** with the exception of **V**, and whose bound variables are those of **M** together with **V**. No sequence of symbols is a W.F.F. except in consequence of these three statements.

In metamathematical statements we use heavy type letters to stand for variable or undetermined formulae, as was done in the last paragraph, and in future such letters will stand for well-formed formulae unless otherwise stated. Small letters in heavy type will stand for formulae representing undetermined positive integers (see below).

A W.F.F. is said to be in normal form if it has no parts of the form {λ**V**[**M**]}(**N**) and none of the form {{δ}(**M**)}(**N**), where **M** and **N** have no free variables.

We say that one W.F.F. is *immediately convertible* into another if it is obtained from it either by:

(i) Replacing one occurrence of a well-formed part λ**V**[**M**] by λ**U**[**N**], where the variable **U** does not occur in **M**, and **N** is obtained from **M** by replacing the variable **V** by **U** throughout.

(ii) Replacing a well-formed part {λ**V**[**M**]}(**N**) by the formula which is obtained from **M** by replacing **V** by **N** throughout, provided that the bound variables of **M** are distinct both from **V** and from the free variables of **N**.

(iii) The process inverse to (ii).

(iv) Replacing a well-formed part {{δ}(**M**)}(**M**) by

$$\lambda f\,[\lambda x\ [\{f\}(\{f\}(x))]]$$

if **M** is in normal form and has no free variables.

(v) Replacing a well-formed part {{δ}(**M**)}(**N**) by

$$\lambda f\,[\lambda x[\{f\}(x)]]$$

if **M** and **N** are in normal form, are not transformable into one another by repeated application of (i), and have no free variables.

(vi) The process inverse to (iv).

(vii) The process inverse to (v).

These rules could have been expressed in such a way that in no case could there be any doubt about the admissibility or the result of the transformation [in particular this can be done in the case of process (v)].

A formula **A** is said to be *convertible* into another **B** (abbreviated to "**A** conv **B**") if there is a finite chain of immediate conversions leading from one formula to the other. It is easily seen that the relation of convertibility is an equivalence relation, *i.e.* it is symmetric, transitive, and reflexive.

the property to hold for a given argument if and only if it is possible to deduce that it holds from the axioms.

Axiomatic properties may also be characterized in this way. A property ψ of positive integers is axiomatic if and only if there is a computable property ϕ of two positive integers, such that $\psi(x)$ is true if and only if there is a positive integer y such that $\phi(x, y)$ is true. Or again ψ is axiomatic if and only if there is a W.F.F. **F** such that $\psi(n)$ is true if and only if $\mathbf{F}(\mathbf{n})$ conv 2.

3. Number-theoretic theorems.

By a *number-theoretic theorem*[†] we shall mean a theorem of the form "$\theta(x)$ vanishes for infinitely many natural numbers x", where $\theta(x)$ is a primitive recursive[‡] function.

We shall say that a problem is number-theoretic if it has been shown that any solution of the problem may be put in the form of a proof of one or more number-theoretic theorems. More accurately we may say that a class of problems is number-theoretic if the solution of any one of them can be transformed (by a uniform process) into the form of proofs of number-theoretic theorems.

I shall now draw a few consequences from the definition of "number theoretic theorems" and in section 5. I shall try to justify confining our consideration to this type of problem.

An alternative form for number-theoretic theorems is "for each natural number x there exists a natural number y such that $\phi(x,y)$ vanishes", where $\phi(x, y)$ is primitive recursive. In other words, there is a rule whereby, given the function $\theta(x)$, we can find a function $\phi(x,y)$, or given $\phi(x,y)$, we can find a function $\theta(x)$, such that "$\theta(x)$ vanishes infinitely often" is a necessary and sufficient condition for "for each x there is a y such that $\phi(x,y) = 0$". In fact, given $\theta(x)$, we define

$$\phi(x,y) = \theta(x) + \alpha(x,y),$$

where $\alpha(x,y)$ is the (primitive recursive) function with the properties

$$\begin{aligned} \alpha(x,y) &= 1 \ (y \leq x), \\ &= 0 \ (y > x). \end{aligned}$$

If on the other-hand we are given $\phi(x,y)$ we define $\theta(x)$ by the equations

$$\theta_1(0) = 3,$$

$$\theta_1(x+1) = 2^{(1+\varpi_2(\theta_1(x)))\sigma(\phi(\varpi_3(\theta_1(x))-1,\ \varpi_2(\theta_1(x))))}3^{\varpi_3(\theta_1(x))+1-\sigma(\phi(\varpi_3(\theta_1(x))-1,\ \varpi_2(\theta_1(x))))},$$

$$\theta(x) = \phi\ (\varpi_3(\theta_1(x)) - 1, \quad \varpi_2(\theta_1(x))),$$

[†] I believe that there is no generally accepted meaning for this term, but it should be noticed that we are using it in a rather restricted sense. Tile most generally accepted meaning is probably this: suppose that we take an arbitrary formula of the functional calculus of the first order and replace the function variables by primitive recursive relations. The resulting formula represents a typical number-theoretic theorem in this (more general) sense.

[‡] Primitive recursive functions of natural numbers arc defined inductively as follows. Suppose that $f(x_1,\ldots,x_{n-1})$, $g(x_1,\ldots,x_n)$, $h(x_1,\ldots,x_{x+1})$ are primitive recursive, then $\varphi(x_1, \ldots, x_n)$ is primitive recursive if it is defined by one of the sets of equations (*a*) to (*e*).

(a) $\phi(x_1, \ldots, x_n) = h((x_1, \ldots, x_{m-1}), g(x_1,\ldots,x_n), x_{m+1},\ldots,x_{n-1},x_n)\ (1 \leq m \leq n)$;
(b) $\phi(x_1, \ldots, x_n) = f(x_2, \ldots, x_n)$;
(c) $\phi(x_1) = a$, where $n = 1$ and α is some particular natural number;
(d) $\phi\ (x_1) = x_1 + 1\ (n = 1)$;
(e) $\phi(x_1,\ldots, x_{n-1}, 0) = f(x_1,\ldots, x_{n-1})$; $\quad \phi(x_1,\ldots, x_{n-1}, x_{n+1}) = h(x_1, \ldots, x_n, \phi(x_1,\ldots, x_n))$.

The class primitive recursive functions is more restricted than the class of computable functions, but it has the advantage that there is a process whereby it can be said of a set of equations whether it defines a primitive recursive function in the manner described above.

where $\varpi_r(x)$ is defined so as to mean "the largest s for which r^s divides x". The function $\sigma(x)$ is defined by the equations $\sigma(0) = 0,\ \sigma(x+1) = 1$. It is easily verified that the functions so defined have the desired properties.

We shall now show that questions about the truth of the statements of the form "does $f(x)$ vanish identically", where $f(x)$ is a computable function, can be reduced to questions about the truth of number-theoretic theorems. It is understood that in each case the rule for the calculation of $f(x)$ is given and that we are satisfied that this rule is valid, *i.e.* that the machine which should calculate $f(x)$ is circle free (Turing [1], 233). The function $f(x)$, being computable, is general recursive in the Herbrand-Gödel sense, and therefore, by a general theorem due to Kleene†, is expressible in the form

$$\psi(\epsilon y[\phi(x,y) = 0]), \tag{3.2}$$

where $\epsilon y[\mathfrak{A}(y)]$ means "the least y for which $\mathfrak{A}(y)$ is true" and $\psi(y)$ and $\phi(x,y)$ are primitive recursive functions. Without loss of generality, we may suppose that the functions $\phi,\ \psi$ take only the values $0,\ 1$. Then, if we define $\rho(x)$ by the equations (3.1) and

$$\rho(0) = \psi(0)(1 - \theta(0)),$$
$$\rho(x+1) = 1 - (1 - \rho(x))\sigma[1 + \theta(x) - \psi\{\varpi_2(\theta_1(x))\}]$$

it will be seen that $f(x)$ vanishes identically if and only if $\rho(x)$ vanishes for infinitely many values of x.

The converse of this result is not quite true. We cannot say that the question about the truth of any number-theoretic theorem is reducible to a question about whether a corresponding computable function vanishes identically; we should have rather to say that it is reducible to the problem of whether a certain machine is circle free and calculates an identically vanishing function. But more is true: every number-theoretic theorem is equivalent to the statement that a corresponding machine is circle free. The behaviour of the machine may be described roughly as follows: the machine is one for the calculation of the primitive recursive function $\theta(x)$ of the number-theoretic problem, except that the results of the calculation are first arranged in a form in which the figures 0 and 1 do not occur, and the machine is then modified so that, whenever it has been found that the function vanishes for some value of the argument, then 0 is printed. The machine is circle free if and only if an infinity of these figures are printed, *i.e.* if and only if $\theta(x)$ vanishes for infinitely many values of the argument. That, on the other hand, questions of circle freedom may be reduced to questions of the truth of number-theoretic theorems follows from the fact that $\theta(x)$ is primitive recursive when it is defined to have the value 0 if a certain machine $\mathcal{M}$ prints 0 or 1 in its $(x+1)$-th complete configuration, and to have the value 1 otherwise.

The conversion calculus provides another normal form for the number theoretic theorems, and the one which we shall find the most convenient to use. Every number-theoretic theorem is equivalent to a statement of the form "$\mathbf{A}(\mathbf{n})$ is convertible to 2 for every W.F.F. $\mathbf{n}$ representing a positive integer", $\mathbf{A}$ being a W.F.F. determined by the theorem; the property of $\mathbf{A}$ here asserted will be described briefly as "$\mathbf{A}$ is dual". Conversely such statements are reducible to number theoretic theorems. The first half of this assertion follows from our results for computable functions, or directly in this way. Since $\theta(x-1)+2$ is primitive recursive, it is formally definable, say, by means of a formula $\mathbf{G}$. Now there is (Kleene [1], 232) a W.F.F. $\mathcal{P}$ with the property that, if $\mathbf{T}(\mathbf{r})$ is convertible to a formula representing a positive integer for each positive integer r, then $\mathcal{P}(\mathbf{T}, \mathbf{n})$ is convertible to s, where s is the n-th positive integer t (if there is one) for which $\mathbf{T}(\mathbf{t})$ conv 2; if $\mathbf{T}(\mathbf{t})$ conv 2

† Kleene [3], 727. This result is really superfluous for our purpose, since the proof that every computable function is general recursive proceeds by showing that these functions are of the form (3.2). (Turing [2], 161).

for less than n values of t then $\mathcal{P}(\mathbf{T}, \mathbf{n})$ has no normal form. The formula $\mathbf{G}(\mathcal{P}(\mathbf{G}, \mathbf{n}))$ is therefore convertible to 2 if and only if $\theta(x)$ vanishes for at least n values of x, and is convertible to 2 for every positive integer x if and only if $\theta(x)$ vanishes infinitely often. To prove the second half of the assertion, we take Gödel representations for the formulae of the conversion calculus. Let $c(x)$ be 0 if x is the G.R. of 2 (*i.e.* if x is $2^3.3^{10}.5.7^3.11^{28}.13.17.19^{10}.23^2.29.31.\ 37^{10}.41^2.43.47^{28}.53^2.59^2.61^2.67^2$) and let $c(x)$ be 1 otherwise. Take an enumeration of the G.R. of the formulae into which **A(m)** is convertible: let $a(m, n)$ be the n-th number in the enumeration. We can arrange the enumeration so that $a(m, n)$ is primitive recursive. Now the statement that **A(m)** is convertible to 2 for every positive integer m is equivalent to the statement that, corresponding to each positive integer m, there is a positive integer n such that $c(a(m, n)) = 0$; and this is number-theoretic.

It is easy to show that a number of unsolved problems, such as the problem of the truth of Fermat's last theorem, are number-theoretic. There are, however, also problems of analysis which are number-theoretic. The Riemann hypothesis gives us an example of this. We denote by $\zeta(s)$ the function defined for $\Re s = \sigma > 1$ by the series $\sum_{n=1}^{\infty} n^{-s}$ and over the rest of the complex plane with the exception of the point $s = 1$ by analytic continuation. The Riemann hypothesis asserts that this function does not vanish in the domain $\sigma > \frac{1}{2}$. It is easily shown that this is equivalent to saying that it does not vanish for $2 > \sigma > \frac{1}{2}\Im s = t > 2$, *i.e.* that it does not vanish inside any rectangle $2 > \sigma > \frac{1}{2} + 1/T,\ T > t > 2$, where T is an integer greater than 2. Now the function satisfies the inequalities

$$\left.\begin{aligned} \left|\zeta(s) - \sum_{1}^{N} n^{-s} - \frac{N^{1-s}}{s-1}\right| &< 2t(N-2)^{-\frac{1}{2}},\ 2 < \sigma < \tfrac{1}{2},\ t \geq 2, \\ |\zeta(s) - \zeta(s')| &< 60t|s - s'|,\ 2 < \sigma' < \tfrac{1}{2},\ t' \geq 2, \end{aligned}\right\}$$

and we can define a primitive recursive function $\xi(l, l', m, m', N, M)$ such that

$$\left|\xi(l,l',m,m',N,M) - M\left|\sum_{1}^{N} n^{-s} + \frac{N^{1-s}}{s-1}\right|\right| < 2, \quad \left(s = \frac{l}{l'} + i\frac{m}{m'}\right),$$

and therefore, if we put

$$\xi(l,\ M,\ m,\ M, M^2 + 2,\ M) = X(l,\ m, M),$$

we have

$$\left|\zeta\left(\frac{l+\vartheta}{M} + i\frac{m+\vartheta}{M}\right)\right| \geq \frac{X(l,m,M) - 122T}{M},$$

provided that

$$\frac{1}{2} + \frac{1}{T} \leq \frac{l-1}{M} < \frac{l+1}{M} < 2 - \frac{1}{M}, \qquad 2 < \frac{m-1}{M} < \frac{m+1}{M} < T$$
$$(-1 < \theta < 1,\ -1 < \theta' < 1).$$

If we define $B(M, T)$ to be the smallest value of $X(l, m, M)$ for which

$$\frac{1}{2} + \frac{1}{T} + \frac{1}{M} \leq \frac{l}{M} < 2 - \frac{1}{M}, \qquad 2 + \frac{1}{M} < \frac{m}{M} < T - \frac{1}{M},$$

then the Riemann hypothesis is true if for each T there is an M satisfying

$$B(M,T) > 122T.$$

formulae are dual. Then we can find a logic formula whose extent consists of just those formulae which can be proved to be dual by the rules; that is to say, there is a rule for obtaining the logic formula from the system of symbolic logic. In fact the system of symbolic logic enables us to obtain[†] a computable function of positive integers whose values run through the Gödel representations of the formulae provable by means of the given rules. By the theorem of equivalence of computable and λ-definable functions, there is a formula $\mathbf{J}$ such that $\mathbf{J}(1)$, $\mathbf{J}(2)$, ... are the G.R. of these formulae. Now let

$$W \rightarrow \lambda jv.\mathcal{P}\left(\lambda u.\delta\left(j(u),v\right),1,I,\ 2\right).$$

Then I assert that $W(\mathbf{J})$ is a logic with the required properties. The properties of $\mathcal{P}$ imply that $\mathcal{P}(\mathbf{C},\ 1)$ is convertible to the least positive integer $\mathbf{n}$ for which $\mathbf{C}(\mathbf{n})$ conv 2, and has no normal form if there is no such integer. Consequently $\mathcal{P}(\mathbf{C},\ 1,\ I,\ 2)$ is convertible to 2 if $\mathbf{C}(\mathbf{n})$ conv 2 for some positive integer n, and it has no normal form otherwise. That is to say that $W(\mathbf{J},\ \mathbf{A})$ conv 2 if and only if $\delta\left(\mathbf{J}(\mathbf{n}),\ \mathbf{A}\right)$ conv 2, some n, *i.e.* if $\mathbf{J}(\mathbf{n})$ conv $\mathbf{A}$ some n.

There is conversely a formula W' such that, if $\mathbf{L}$ is a logic, then $W'(\mathbf{L})$ enumerates the extent of $\mathbf{L}$. For there is a formula Q such that $Q(\mathbf{L},\ \mathbf{A},\ \mathbf{n})$ conv 2 if and only if $\mathbf{L}(\mathbf{A})$ is convertible to 2 in less than n steps. We then put

$$W' \rightarrow \lambda ln.\ \mathrm{form}\left(\varpi\left(2,\mathcal{P}\left(\lambda x.Q\left(l,\mathrm{form}\left(\varpi\left(2,x\right)\right),\varpi\left(3,x\right)\right),n\right)\right)\right).$$

Of course, $W'\left(W(\mathbf{J})\right)$ normally entirely different from $\mathbf{J}$ and $W\left(W'(\mathbf{L})\right)$ from $\mathbf{L}$.

In the case where we have a symbolic logic whose propositions can be interpreted as number-theoretic theorems, but are not expressed in the form of the duality of formulae, we shall again have a corresponding logic formula, but its relation to the symbolic logic is not so simple. As an example let us take the case where the symbolic logic proves that certain primitive recursive functions vanish infinitely often. As was shown in §3, we can associate with each such proposition a W.F.F. which is dual if and only if the proposition is true. When we replace the propositions of the symbolic logic by theorems on the duality of formulae in this way, our previous argument applies and we obtain a certain logic formula $\mathbf{L}$. However, $\mathbf{L}$ does not determine uniquely which are the propositions provable in the symbolic logic; for it is possible that "$\theta_1(x)$ vanishes infinitely often" and "$\theta_2(x)$ vanishes infinitely often" are both associated with "A is dual", and that the first of these propositions is provable in the system, but the second not. However, if we suppose that the system of symbolic logic is sufficiently powerful to be able to carry out the argument on pp. 157–158 then this difficulty cannot arise. There is also the possibility that there may be formulae in the extent of $\mathbf{L}$ with no propositions of the form "$\theta(x)$ vanishes infinitely often" corresponding to them. But to each such formula we can assign (by a different argument) a proposition p of the symbolic logic which is a necessary and sufficient condition for $\mathbf{A}$ to be dual. With p is associated (in the first way) a formula $\mathbf{A}'$. Now $\mathbf{L}$ can always be modified so that its extent contains $\mathbf{A}'$ whenever it contains $\mathbf{A}$.

We shall be interested principally in questions of completeness. Let us suppose that we have a class of systems of symbolic logic, the propositions of these systems being expressed in a uniform notation and interpretable as number-theoretic theorems; suppose also that there is a rule by which we can assign to each proposition p of the notation a W.F.F. $\mathbf{A}_p$, which is dual if and only if p is true, and that to each W.F.F. $\mathbf{A}$ we can assign a proposition $p_{\mathbf{A}}$ which is a necessary and sufficient condition for $\mathbf{A}$ to be dual. $p_{\mathbf{A}_p}$, is to be expected to differ from p. To each symbolic logic C we can assign two logic formulae $\mathbf{L}_c$ and $\mathbf{L}'_c$. A formula $\mathbf{A}$ belongs to the extent of $\mathbf{L}_c$ if $p_{\mathbf{A}}$ is provable in C, while the extent of $\mathbf{L}'_C$ consists of all $\mathbf{A}p$, where p is provable in C. Let us say that the class of symbolic logics is complete if each true proposition is provable in one of them: let us also say that

[†] Compare Turing [1], 252, second footnote, [2], 156.

The class defined by $D(x)$ is then called a *series* with the ordering relation $G(x,y)$. The series is said to be *well ordered* and the ordering relation is called an *ordinal* if every sub-series which is not void has a first term, *i.e.* if

$$(D')\{(\exists x)(D'(x))\&(x)(D'(x) \supset D(x)) \supset (\exists z)(y)[D'(z)\&(D'(y) \supset G(z,y) \vee z = y)]\}. \tag{7.2}$$

The condition (7.2) is equivalent to another, more suitable for our purposes, namely the condition that every descending subsequence must terminate formally

$$(x)\left\{D'(x) \supset D(x)\&(\exists y)(D'(y)\&G(y,x))\right\} \supset (x)(\sim D'(x)). \tag{7. 3}$$

The ordering relation $G(x,y)$ is said to be similar to $G'(x,y)$ if there is a one-one correspondence between the series transforming the one relation into the other. This is best expressed formally, thus

$$\begin{aligned}(\exists M)[(x)\{D(x) \supset (\exists x')\mathrm{M}(\mathrm{x},\, x')\}\&(x')\{D'(x') \supset (\exists x)M(x,\, x')\} \\ \&\{(M(x,\, x')\&M\,(x,\, x'')) \vee (M(x',\, x)\&M\,(x'',\, x)) \supset x' = x''\} \\ \&\{M(x,\, x')\&M(y,\, y') \supset (G(x,\, y) = G(x',\, y'))\}].\end{aligned} \tag{7.4}$$

Ordering relations are regarded as belonging to the same ordinal if and only if they are similar.

We wish to give names to all the ordinals, but this will not be possible until they have been restricted in some way; the class of ordinals, as at present defined, is more than enumerable. The restrictions that we actually impose are these: $D(x)$ is to imply that x is a positive integer; $D(x)$ and $G(x,\, y)$ are to be computable properties. Both of the propositional functions $D(x)$, $G(x,\, y)$ can then be described by means of a single W.F.F. $\mathbf{\Omega}$ with the properties:

$\mathbf{\Omega}(\mathbf{m},\, \mathbf{n})$ conv 4 unless both $D(m)$ and $D(n)$ are true,

$\mathbf{\Omega}(\mathbf{m},\, \mathbf{m})$ conv 3 if $D(m)$ is true,

$\mathbf{\Omega}(\mathbf{m},\, \mathbf{n})$ conv 2 if $D(m)$, $D(n)$, $G(m,\, n), \sim (m = n)$ are true,

$\mathbf{\Omega}(\mathbf{m},\, \mathbf{n})$ conv 1 if $D(m)$, $D(n), \sim G(m,\, n), \sim (m = n)$ are true.

In consequence of the conditions to which $D(x)$, $G(x,y)$ are subjected, $\mathbf{\Omega}$ must further satisfy:

(a) if $\mathbf{\Omega}(\mathbf{m},\, \mathbf{n})$ is convertible to 1 or 2, then $\mathbf{\Omega}(\mathbf{m},\, \mathbf{m})$ and $\mathbf{\Omega}(\mathbf{n},\, \mathbf{n})$ arc convertible to 3,

(b) if $\mathbf{\Omega}(\mathbf{m},\, \mathbf{m})$ and $\mathbf{\Omega}(\mathbf{n},\, \mathbf{n})$ are convertible to 3, then $\mathbf{\Omega}(\mathbf{m},\mathbf{n})$ is convertible to 1, 2, or 3,

(c) if $\mathbf{\Omega}(\mathbf{m},\, \mathbf{n})$ is convertible to 1, then $\mathbf{\Omega}(\mathbf{n},\, \mathbf{m})$ is convertible to 2 and conversely,

(d) if $\mathbf{\Omega}(\mathbf{m},\, \mathbf{n})$ and $\mathbf{\Omega}(\mathbf{n},\, \mathbf{p})$ are convertible to 1, then $\mathbf{\Omega}(\mathbf{m},\, \mathbf{p})$ is also,

(e) there is no sequence $m_1,\, m_2,\, \ldots$ such that $\mathbf{\Omega}(\mathbf{m}_{i+1},\, \mathbf{m}_i)$ conv 2 for each positive integer i,

(f) $\mathbf{\Omega}(\mathbf{m},\mathbf{n})$ is always convertible to 1, 2, 3, or 4.

If a formula $\mathbf{\Omega}$ satisfies these conditions then there are corresponding propositional functions $D(x)$, $G(x,y)$. We shall therefore say that $\mathbf{\Omega}$ is an *ordinal formula* if it satisfies the conditions $(a)-(f)$. It will be seen that a consequence of this definition is that Dt is an ordinal formula; it represents the ordinal ω. The definition that we have given does not pretend to have virtues such as elegance or convenience. It has been introduced rather to fix our ideas and to show how it is possible in principle to describe ordinals by means of well formed formulae. The definitions could be modified in a number of ways. Some such modifications arc quite trivial; they are typified by modifications such as changing the numbers 1, 2, 3, 4, used in the definition, to others. Two such definitions will be said to be equivalent; in general, we shall say that two definitions are equivalent if there are W.F.F. $\mathbf{T}$, $\mathbf{T}'$ such that, if $\mathbf{A}$ is an ordinal formula under one definition and represents the ordinal α, then $\mathbf{T}'(\mathbf{A})$ is an ordinal formula under the second definition and represents the same ordinal; and, conversely, if $\mathbf{A}'$ is an ordinal formula under the second definition representing α, then $\mathbf{T}(\mathbf{A}')$ represents α under the first definition. Besides definitions equivalent in this sense to our original definition, there are a number of other possibilities open. Suppose for instance that we do not

require $D(x)$ and $G(x,\ y)$ to be computable, but that we require only that $D(x)$ and $G(x,\ y)\,\&\,x < y$ are axiomatic[†]. This leads to a definition of an ordinal formula which is (presumably) not equivalent to the definition that we are using[‡]. There are numerous possibilities, and little to guide us in choosing one definition rather than another. No one of them could well be described as "wrong"; some of them may be found more valuable in applications than others, and the particular choice that we have made has been determined partly by the applications that we have in view. In the case of theorems of a negative character, one would wish to prove them for each one of the possible definitions of "ordinal formula". This programme could, I think, be carried through for the negative results of §9, 10.

Before leaving the subject of possible ways of defining ordinal formulae, I must mention another definition due to Church and Kleene (Church and Kleene [**1**]). We can make use of this definition in constructing ordinal logics, but it is more convenient to use a slightly different definition which is equivalent (in the sense just described) to the Church-Kleene definition as modified in Church [**4**].

Introduce the abbreviations

$$U \to \lambda ufx.u(\lambda y.f(y(I,x))),$$
$$Suc \to \lambda aufx.f(a(u,\ f,\ x)).$$

We define first a partial ordering relation "$<$" which holds 164 between certain pairs of W.F.F. [*conditions*$(1)-(5)$].

(1) If **A** conv **B**, then $\mathbf{A} < \mathbf{C}$ implies $\mathbf{B} < \mathbf{C}$ and $\mathbf{C} < \mathbf{A}$ implies $\mathbf{C} < \mathbf{B}$.

(2) $\mathbf{A} <$ Suc (**A**).

(3) For any positive integers m and n, $\lambda ufx.\ \mathbf{R}(\mathbf{n}) < \lambda ufx.\mathbf{R}(\mathbf{m})$ implies $\lambda ufx.\ \mathbf{R}(\mathbf{n}) < \lambda ufx.u(\mathbf{R})$.

(4) If $\mathbf{A} < \mathbf{B}$ and $\mathbf{B} < \mathbf{C}$, then $\mathbf{A} < \mathbf{C}$. $(1)-(4)$ are required for any W.F.F. **A**, **B**, **C**, $\lambda ufx.\ \mathbf{R}$.

(5) The relation $\mathbf{A} < \mathbf{B}$ holds only when compelled to do so by $(1)-(4)$.

We define C-K ordinal formulae by the conditions (6)–(10).

(6) If **A** conv **B** and **A** is a C-K ordinal formula, then **B** is a C-K ordinal formula.

(7) U is a C-K ordinal formula.

(8) If **A** is a C-K ordinal formula, then Suc (**A**) is a C-K ordinal formula.

(9) If $\lambda ufx.\mathbf{R}(\mathbf{n})$ is a C-K ordinal formula and

$$\lambda ufx.\mathbf{R}(\mathbf{n}) < \lambda ufx.\mathbf{R}(S(\mathbf{n}))$$

for each positive integer n, then $\lambda ufx.u(\mathbf{R})$ is a C-K ordinal formula[¶].

(10) A formula is a C-K ordinal formula only if compelled to be so by $(6)-(9)$.

[†] To require $G(x,y)$ to be axiomatic amounts to requiring $G(x,y)$ to be computable on account of (7.1) (ii).

[‡] On the other hand, if $D(x)$ is axiomatic and $(G(x,y)$ is computable in the modified sense that there is a rule for determining whether $G(x,y)$ is true which leads to a definite result in all cases where $D(x)$ and $D(y)$ are true, the corresponding definition of ordinal formula is equivalent to our definition. To give the proof would be too much of a digression. Probably other equivalences of this kind hold.

[¶] If we also allow $\lambda ufx.u(\mathbf{R})$ to be a C-K ordinal formula when

$$\lambda ufx.\mathbf{n}(\mathbf{R})\text{conv}\lambda ufx.S(\mathbf{n},\mathbf{R})$$

for all n, then the formulae for sum, product and exponentiation of C-K ordinal formulae can be much simplified. For instance, if **A** and **B** represent α and β, then

$$\lambda ufx.\mathbf{B}(u,f,\mathbf{A}(u,f,x))$$

represents $\alpha+\beta$. Property (6) remains true.

hypothesis that we need. There are three cases to consider:

(x) $\alpha = 0$.

(y) $\alpha = \beta + 1$.

(z) α is of neither of the forms (x), (y).

In case (x) we must have $\mathbf{A}$ conv U on account of (A). In case (y) there is a formula $\mathbf{D}$ such that $\mathbf{D} < \mathbf{A}$, and $\mathbf{B} \leq \mathbf{D}$ whenever $\mathbf{B} < \mathbf{A}$. The relation $\mathbf{D} < \mathbf{A}$ must hold in virtue of either (1), (2), (3), or (4). It cannot be in virtue of (4); for then there would be $\mathbf{B}$, $\mathbf{B} < \mathbf{A}$, $\mathbf{D} < \mathbf{B}$ contrary to (C), taken in conjunction with the definition of $\mathbf{D}$. If it is in virtue of (3), then α is the upper bound of a sequence α_1, α_2, ... of ordinals, which are increasing by reason of (iii) and the conditions $\lambda ufx.\mathbf{R}(\mathbf{n}) < \lambda ufx.\mathbf{R}(S(\mathbf{n}))$ in (B). This is inconsistent with $\alpha = \beta + 1$. This means that (2) applies [after we have eliminated (1) by suitable conversions on $\mathbf{A}$, $\mathbf{D}$] and we see that $\mathbf{A}$ conv Suc ($\mathbf{D}$); but, since $\mathbf{D} < \mathbf{A}$, $\mathbf{D}$ is a C-K ordinal formula, and $\mathbf{A}$ must therefore be a C-K ordinal formula by (8). Now take case (z). It is impossible for $\mathbf{A}$ to be of the form Suc ($\mathbf{D}$), for then we should have $\mathbf{B} < \mathbf{D}$ whenever $\mathbf{B} < \mathbf{A}$, and this would mean that we had case (y). Since $U < \mathbf{A}$, there must be an $\mathbf{F}$ such that $\mathbf{F} < \mathbf{A}$ is demonstrable either by (2) or by (3) (after a possible conversion on $\mathbf{A}$); it must of course be demonstrable by (3). Then $\mathbf{A}$ is of the form $\lambda ufx.u(\mathbf{R})$. By (3), (B) we see that $\lambda ufx.\mathbf{R}(\mathbf{n}) < \mathbf{A}$ for each positive integer n; each $\lambda ufx.\mathbf{R}(\mathbf{n})$ is therefore a C-K ordinal formula. Applying (9), ($\mathbf{B}$) we see that $\mathbf{A}$ is a C-K ordinal formula.

Proof of (vi). To prove the first half, it is sufficient to find a method whereby from a C-K ordinal formula $\mathbf{A}$ we can find the corresponding ordinal formula $\mathbf{\Omega}$. For then there is a formula H_1 such that $H_1(\mathbf{a})$ conv $\mathbf{p}$ if a is the G.R. of $\mathbf{A}$ and p is that of $\mathbf{\Omega}$. H is then to be defined by

$$H \to \lambda a.\ \text{form}\ (H_1(\text{Gr}(a))).$$

The method of finding $\mathbf{\Omega}$ may be replaced by a method of finding $\mathbf{\Omega}(\mathbf{m}, \mathbf{n})$, given $\mathbf{A}$ and any two positive integers m, n. We shall arrange the method so that, whenever $\mathbf{A}$ is not an ordinal formula, either the calculation of the values does not terminate or else the values are not consistent with $\mathbf{\Omega}$ being an ordinal formula. In this way we can prove the second half of (vi).

Let Ls be a formula such that Ls($\mathbf{A}$) enumerates the classes of formulae $\mathbf{B}$, $\mathbf{B} < \mathbf{A}$ [*i.e.* if $\mathbf{B} < \mathbf{A}$ there is one and only one positive integer n for which Ls($\mathbf{A}$, $\mathbf{n}$) conv $\mathbf{B}$]. Then the rule for finding the value of $\mathbf{\Omega}(\mathbf{m}, \mathbf{n})$ is as follows:—

First determine whether $U \leq \mathbf{A}$ and whether $\mathbf{A}$ is convertible to the form $\mathbf{r}$(Suc, U). This terminates if $\mathbf{A}$ is a C-K ordinal formula.

If $\mathbf{A}$ conv $\mathbf{r}$(Suc, U) and either $m > r + 1$ or $n > r + 1$, then the value is 4. If $m < n \leq r + 1$, the value is 2. If $n < m \leq r + 1$, the value is 1. If $m = n \leq r + 1$, the value is 3.

If $\mathbf{A}$ is not convertible to this form, we determine whether either $\mathbf{A}$ or Ls($\mathbf{A}$, $\mathbf{m}$) is convertible to the form $\lambda ufx.u(\mathbf{R})$, and if either of them is, we verify that $\lambda ufx.\mathbf{R}(\mathbf{n}) < \lambda ufx.\mathbf{R}(S(\mathbf{n}))$. We shall eventually come to an affirmative answer if $\mathbf{A}$ is a C-K ordinal formula.

Having checked this, we determine concerning m and n whether Ls($\mathbf{A}$, $\mathbf{m}$) < Ls($\mathbf{A}$, $\mathbf{n}$), Ls($\mathbf{A}$, $\mathbf{n}$) < Ls($\mathbf{A}$, $\mathbf{m}$), or $m = n$, and the value is to be accordingly 1, 2, or 3.

If $\mathbf{A}$ is a C-K ordinal formula, this process certainly terminates. To see that the values so calculated correspond to an ordinal formula, and one representing $\Xi_{\mathbf{A}}$, first observe that this is so when $\Xi_{\mathbf{A}}$ is finite. In the other case (iii) and (iv) show that $\Xi_{\mathbf{B}}$ determines a one-one correspondence between the ordinals $\beta, 1 \leq \beta \leq \Xi_{\mathbf{A}}$, and the classes of interconvertible formulae $\mathbf{B}$, $\mathbf{B} < \mathbf{A}$. If we take $G(m, n)$ to be Ls ($\mathbf{A}$, $\mathbf{m}$) < Ls($\mathbf{A}$, $\mathbf{n}$), we see that $G(m, n)$ is the ordering relation of a series of order type[†] $\Xi_{\mathbf{A}}$ and on the other hand that the values of $\mathbf{\Omega}(\mathbf{m}, \mathbf{n})$ are related to $G(m,n)$ as on p. 163.

[†] The order type is β, where $1 + \beta = \Xi_{\mathbf{A}}$; but $\beta = \Xi_{\mathbf{A}}$ since $\Xi_{\mathbf{A}}$ is infinite.

To prove this we shall show that to each C-K ordinal formula **A** there corresponds a unique system $C[\mathbf{A}]$ such that:

(i) $\mathbf{A}(\Theta, \mathbf{K}, \mathbf{m}_{C_0})$ conv $\mathbf{m}_{C[\mathbf{A}]}$,

and that it further satisfies:

(ii) $C[U]$ is a limit system of $C'_0, C'_0, \ldots$,

(iii) C [Suc (**A**)] is $(C[\mathbf{A}])'$,

(iv) $C[\lambda ufx.u(\mathbf{R})]$ is a limit system of $C[\lambda ufx.\mathbf{R}(1)], C[\lambda ufx.\mathbf{R}(2)], \ldots$,

A and $\lambda ufx.u(\mathbf{R})$ being assumed to be C-K ordinal formulae. The uniqueness of the system follows from the fact that m_C determines C completely. Let us try to prove the existence of $C[\mathbf{A}]$ for each C-K ordinal formula **A**. As we have seen (p. 165) it is sufficient to prove

(a) $C[U]$ exists,

(b) if $C[\mathbf{A}]$ exists, then $C[\mathrm{Suc}(\mathbf{A})]$ exists,

(c) if $C[\lambda ufx.\mathbf{R}(1)], C[\lambda ufx.\mathbf{R}(2)], \ldots$ exist, then $C[\lambda ufx.u(\mathbf{R})]$ exists.

Proof of (*a*).

$$\{\lambda y.\mathbf{K}(y(I, \mathbf{m}_{C_0}))\}(\mathbf{n}) \text{ conv } \mathbf{K}(\mathbf{m}_{C_0}) \text{ conv } \mathbf{m}_{C'_0}$$

for all positive integers n, and therefore, by the definition of Θ, there is a system, which we call $C[U]$ and which is a limit system of $C'_0, C'_0, \ldots$, satisfying

$$\Theta(\lambda y.\mathbf{K}(y(I, \mathbf{m}_{C_0}))) \text{ conv } \mathbf{m}_{C[U]}.$$

But, on the other hand,

$$U(\Theta, \mathbf{K}, \mathbf{m}_{C_0}) \text{ conv } (\lambda y.\mathbf{K}(y(I, \mathbf{m}_{C_0}))).$$

This proves (*a*) and incidentally (ii).

Proof of (*b*).

$$\begin{aligned} \mathrm{Suc}(\mathbf{A}, \Theta, \mathbf{K}, m_{C_0}) &\text{ conv } \mathbf{K}(\mathbf{A}(\Theta, \mathbf{K}, \mathbf{m}_{C_0})) \\ &\text{conv } \mathbf{K}(\mathbf{m}_{C[\mathbf{A}]}) \\ &\text{conv } \mathbf{m}_{(C[\mathbf{A}])'}. \end{aligned}$$

Hence $C[\mathrm{Suc}(\mathbf{A})]$ exists and is given by (iii).

Proof of (*c*).

$$\begin{aligned} \{\{\lambda ufx.\mathbf{R}\}(\Theta, \mathbf{K}, \mathbf{m}_{C_0})\}(\mathbf{n}) &\text{ conv } \{\lambda ufx.\mathbf{R}(\mathbf{n})\}(\Theta, \mathbf{K}, \mathbf{m}_{C_0}) \\ &\text{conv } \mathbf{m}_{C[\lambda ufx.\mathbf{R}(\mathbf{n})]} \end{aligned}$$

by hypothesis. Consequently, by the definition of Θ, there exists a C which is a limit system of

$$C[\lambda ufx.\mathbf{R}(1)], \ C[\lambda ufx.\mathbf{R}(2)], \ldots,$$

and satisfies

$$\Theta(\{\lambda ufx.u(\mathbf{R})\}(\Theta, \mathbf{K}, \mathbf{m}_{C_0})) \text{ conv } \mathbf{m}_C.$$

We define $C[\lambda ufx.u(\mathbf{R})]$ to be this C. We then have (iv) and

$$\begin{aligned} \{\lambda ufx.u(\mathbf{R})\}(\Theta, \mathbf{K}, \mathbf{m}_{C_0}) &\text{ conv } \Theta(\{\lambda ufx.\mathbf{R}\}(\Theta, \mathbf{K}, \mathbf{m}_{C_0})) \\ &\text{conv } \mathbf{m}_{C[\lambda ufx.u(\mathbf{R})]}. \end{aligned}$$

This completes the proof of the properties (i)−(iv). From (ii), (iii), (iv), the fact that C_0 is valid, and that C' is valid when C is valid, we infer that $C[\mathbf{A}]$ is valid for each C-K ordinal formula $\mathbf{A}$: also that there are more formulae provable in $C[\mathbf{B}]$ than in $C[\mathbf{A}]$ when $\mathbf{A} < \mathbf{B}$. The truth of our assertions regarding $\mathbf{N}$ now follows in view of (i) and the definitions of $\mathbf{N}$ and $\mathbf{G}$.

We cannot conclude that $\mathbf{N}$ is an ordinal logic, since the formulae $\mathbf{A}$ are C-K ordinal formulae; but the formula H enables us to obtain an ordinal logic from $\mathbf{N}$. By the use of the formula Gr we obtain a formula Tn such that, if $\mathbf{A}$ has a normal form, then Tn($\mathbf{A}$) enumerates the G.R.'s of the formulae into which $\mathbf{A}$ is convertible. Also there is a formula Ck such that, if h is the G.R. of a formula $H(\mathbf{B})$, then Ck($\mathbf{h}$) conv $\mathbf{B}$, but otherwise Ck ($\mathbf{h}$) conv U. Since $H(\mathbf{B})$ is an ordinal formula only if $\mathbf{B}$ is a C-K ordinal formula Ck (Tn ($\mathbf{\Omega}, \mathbf{n}$)) is a C-K ordinal formula for each ordinal formula $\mathbf{\Omega}$ and each integer n. For many ordinal formulae it will be convertible to U, but, for suitable $\mathbf{\Omega}$, it will be convertible to any given C-K ordinal formula. If we put

$$\mathbf{\Lambda} \to \lambda wa.\Gamma(\lambda n.\mathbf{N}(\mathrm{Ck}(Tn(w,n))),a),$$

$\mathbf{\Lambda}$ is the required ordinal logic. In fact, on account of the properties of $\mathbf{\Gamma}$, $\mathbf{\Lambda}(\mathbf{\Omega}, \mathbf{A})$ will be convertible to 2 if and only if there is a positive integer n such that

$$\mathbf{N}(\mathrm{Ck}\ (\mathrm{Tn}\ (\mathbf{\Omega}, \mathbf{n})), \mathbf{A})\ \mathrm{conv}\ 2.$$

If $\mathbf{\Omega}$ conv $H(\mathbf{B})$, there will be an integer n such that Ck ($\mathbf{Tn}(\mathbf{\Omega}, \mathbf{n})$) conv $\mathbf{B}$, and then

$$\mathbf{N}(\mathrm{Ck}\ (\mathrm{Tn}\ (\mathbf{\Omega}, \mathbf{n})), \mathbf{A})\ \mathrm{conv}\ \mathbf{N}(\mathbf{B}, \mathbf{A}).$$

For any n, Ck (Tn ($\mathbf{\Omega}, \mathbf{n}$)) is convertible to U or to some $\mathbf{B}$, where $\mathbf{\Omega}$ conv $H(\mathbf{B})$. Thus $\Lambda(\mathbf{\Omega}, \mathbf{A})$ conv 2 if $\mathbf{\Omega}$ conv $H(\mathbf{B})$ and $\mathbf{N}(\mathbf{B}, \mathbf{A})$ conv 2 or if $\mathbf{N}(U, \mathbf{A})$ conv 2, but not in any other case.

We may now specialize and consider particular classes W of systems. First let us try to construct the ordinal logic described roughly in the introduction. For W we take the class of systems arising from the system of *Principia Mathematica*[†] by adjoining to it axiomatic (in the sense described on p. 155) sets of axioms[‡]. Gödel has shown that primitive recursive relations[§] can be expressed by means of formulae in P. In fact, there is a rule whereby, given the recursion equations defining a primitive recursive relation, we can find a formula[¶] $\mathfrak{A}[x_0, \ldots, z_0]$ such that

$$\mathfrak{A}[f^{(m_1)}0, ..., f^{(m_r)}0]$$

is provable in P if $F(m_1, \ldots, m_r)$ is true, and its negation is provable otherwise. Further, there is a method by which we can determine about a formula $\mathfrak{A}[x_0, \ldots, z_0]$ whether it arises from a primitive recursive relation in this way, and by which we can find the equations which defined the relation. Formulae of this kind will be called *recursion formulae*. We shall make use of a property that they possess, which we cannot prove formally here without giving their definition in full, but which is

[†] Whitehead and Russell [1]. The axioms and rules of procedure of a similar system P will be found in a convenient form in Gödel [1], and I follow Gödel. The symbols for the natural numbers in P are $0, f0, ff0, \ldots, f^{(n)}0 \ldots$. Variables with the suffix "0" stand for natural numbers.

[‡] It is sometimes regarded as necessary that the set of axioms used should be computable, the intention being that it should be possible to verify of a formula reputed to be an axiom whether it really is so. We can obtain the same effect with axiomatic sets of axioms in this way. In the rules of procedure describing which are the axioms, we incorporate a method of enumerating them, and we also introduce a rule that in the main part of the deduction, whenever we write down an axiom as such, we must also write down its position in the enumeration. It is possible to verify whether this has been done correctly.

[§] A relation $F(m_1, \ldots, m_r)$ is primitive recursive if it is a necessary and sufficient condition for the vanishing of a primitive recursive function $\phi(m_1, \ldots, m_r)$.

[¶] Capital German letters will be used to stand for variable or undetermined formulae in P. An expression such as 1[,] stands for the result of substituting $\mathfrak{B}$ and $\mathfrak{E}$ for x_0 and y_0 in $\mathfrak{A}$.

enumerates the G.R.'s of the axioms of C. There is a W.F.F. E such that, if a is the G.R. of some proposition $\mathfrak{F}$, then $E(M_c, \mathbf{a})$ is convertible to the G.R. of

$$(\exists x_0)\text{Proof}_C[x_0, f^{(a)}0] \supset \mathfrak{F}.$$

If a is not the G.R. of any proposition in P, then $E(M_C, \mathbf{a})$ is to be convertible to the G.R. of $0 = 0$. From E we obtain a W.F.F. K such that $K(M_C, 2\mathbf{n}+1)$ conv $M_C(\mathbf{n}), K(M_C, 2\mathbf{n})$ conv $E(M_{C,\mathbf{n}})$. The successor system C' is defined by $K(M_C)$ conv M'_C. Let us choose a formula G such that $G(M_C, \mathbf{A})$ conv 2 if and only if the number-theoretic theorem equivalent to "**A** is dual" is provable in C. Then we define Λ_P by

$$\Lambda_P \to \lambda wa.\Gamma(\lambda y.G(\text{Ck}(\text{Tn}(w,y), \lambda mn.m(\varpi(2,n), \varpi(3,n)), K, M_P)), a).$$

This is all ordinal logic provided that P is valid.

Another ordinal logic of this type has in effect been introduced by Church[†]. Superficially this ordinal logic seems to have no more in common with Λ_P than that they both arise by the method which we have described, which uses C-K ordinal formulae. The initial systems are entirely different. However, in the relation between C and C' there is an interesting analogy. In Church's method the step from C to C' is performed by means of subsidiary axioms of which the most important (Church [2], p. 88, 1_m) is almost a direct translation into his symbolism of the rule that we may take any formula of the form (8.4) as an axiom. There are other extra axioms, however, in Church's system, and it is therefore not unlikely that it is in some respects more complete than Λ_P.

There are other types of ordinal logic, apparently quite unrelated to the type that we have so far considered. I have in mind two types of ordinal logic, both of which can be best described directly in terms of ordinal formulae without any reference to C-K ordinal formulae. I shall describe here a specimen Λ_H of one of these types of ordinal logic. Ordinal logics of this kind were first considered by Hilbert (Hilbert [1], 183ff), and have also been used by Tarski (Tarski [1], 395ff); see also Gödel [1], foot-note 48^a.

Suppose that we have selected a particular ordinal formula $\mathbf{\Omega}$. We shall construct a modification $P_{\mathbf{\Omega}}$ of the system P of Gödel (see foot-note † on p. 172. We shall say that a natural number n is a *type* if it is either even or $2p-1$, where $\mathbf{\Omega}(\mathbf{p}, \mathbf{p})$ conv 3. The definition of a variable in P is to be modified by the condition that the only admissible subscripts are to be the types in our sense. Elementary expressions are then defined as in P: in particular the definition of an elementary expression of type 0 is unchanged. An elementary formula is defined to be a sequence of symbols of the form $\mathfrak{A}_m\mathfrak{A}_n$, where $\mathfrak{A}_m$, $\mathfrak{A}_n$ are elementary expressions of types m, n satisfying one of the conditions $(a), (b), (c)$.

(a) m and n are both even and m exceeds n,

(b) m is odd and n is even,

(c) $m = 2p-1, n = 2q-1,$ and $\mathbf{\Omega}(\mathbf{p}, \mathbf{q})$ conv 2.

With these modifications the formal development of $P_{\mathbf{\Omega}}$ is the same as that of P. We want, however, to have a method of associating number-theoretic theorems with certain of the formulae of $P_{\mathbf{\Omega}}$. We cannot take over directly the association which we used in P. Suppose that G is a formula in P interpretable as a number-theoretic theorem in the way described in the course of constructing Λ_P (p. 172). Then, if every type suffix in G is doubled, we shall obtain a formula in $P_{\mathbf{\Omega}}$ which is to be interpreted as the same number-theoretic theorem. By the method of §6 we can now obtain from $P_{\mathbf{\Omega}}$ a formula $L_{\mathbf{\Omega}}$ which is a logic formula if $P_{\mathbf{\Omega}}$ is valid; in fact, given $\mathbf{\Omega}$ there is a method of obtaining $L_{\mathbf{\Omega}}$, so that there is a formula Λ_H such that $\Lambda_H(\mathbf{\Omega})$ conv $L_{\mathbf{\Omega}}$ for each ordinal formula $\mathbf{\Omega}$.

Having now familiarized ourselves with ordinal logics by means of these examples we may begin to consider general questions concerning them.

[†] In outline Church [1], 279–280. In greater detail Church [2], Chap. X.

9. Completeness questions.

The purpose of introducing ordinal logics was to avoid as far as possible the effects of Gödel's theorem. It is a consequence of this theorem, suitably modified, that it is impossible to obtain a complete logic formula, or (roughly speaking now) a complete system of logic. We were able, however, from a given system to obtain a more complete one by the adjunction as axioms of formulae, seen intuitively to be correct, but which the Gödel theorem shows are unprovable[†] in the original system; from this we obtained a yet more complete system by a repetition of the process, and so on. We found that the repetition of the process gave us a new system for each C-K ordinal formula. We should like to know whether this process suffices, or whether the system should be extended in other ways as well. If it were possible to determine about a W.F.F. in normal form whether it was an ordinal formula, we should know for certain that it was necessary to make extensions in other ways. In fact for any ordinal formula $\mathbf{\Lambda}$ it would then be possible to find a single logic formula $\mathbf{L}$ such that, if $\mathbf{\Lambda}(\mathbf{\Omega},\mathbf{A})$ conv 2 for some ordinal formula $\mathbf{\Omega}$, then $\mathbf{L}(\mathbf{A})$ conv 2. Since $\mathbf{L}$ must be incomplete, there must be formulae $\mathbf{A}$ for which $\mathbf{\Lambda}(\mathbf{\Omega},\mathbf{A})$ is not convertible to 2 for any ordinal formula $\mathbf{\Omega}$. However, in view of the fact, proved in §7, that there is no method of determining about a formula in normal form whether it is an ordinal formula, the case does not arise, and there is still a possibility that some ordinal logics may be complete in some sense. There is a quite natural way of defining completeness.

Definition of completeness of an ordinal logic

We say that an ordinal logic $\mathbf{\Lambda}$ is complete if corresponding to each dual formula $\mathbf{A}$ there is an ordinal formula $\mathbf{\Omega}_{\mathbf{A}}$ such that $\mathbf{\Lambda}(\mathbf{\Omega}_{\mathbf{A}},\mathbf{A})$ conv 2.

As has been explained in §2, the reference in the definition to the existence of $\mathbf{\Omega}_{\mathbf{A}}$ for each $\mathbf{A}$ is to be understood in the same naïve way as any reference to existence in mathematics.

There is room for modification in this definition: we might require that there is a formula $\mathbf{X}$ such that $\mathbf{X}(\mathbf{A})$ conv $\mathbf{\Omega}_{\mathbf{A}}$, $\mathbf{X}(\mathbf{A})$ being an ordinal formula whenever $\mathbf{A}$ is dual. There is no need, however, to discuss the relative merits of these two definitions, because in all cases in which we prove an ordinal logic to be complete we shall prove it to be complete even in the modified sense; but in cases in which we prove an ordinal logic to be incomplete, we use the definition as it stands.

In the terminology of §6, $\mathbf{\Lambda}$ is complete if the class of logics $\mathbf{\Lambda}(\mathbf{\Omega})$ is complete when $\mathbf{\Omega}$ runs through all ordinal formulae.

There is another completeness property which is related to this one. Let us for the moment describe an ordinal logic $\mathbf{\Lambda}$ as *all inclusive* if to each logic formula $\mathbf{L}$ there corresponds an ordinal formula $\mathbf{\Omega}_{(\mathbf{L})}$ such that $\mathbf{\Lambda}(\mathbf{\Omega}_{(\mathbf{L})})$ is as complete as $\mathbf{L}$. Clearly every all inclusive ordinal logic is complete; for, if $\mathbf{A}$ is dual, then $\delta(A)$ is a logic with $\mathbf{A}$ in its extent. But, if $\mathbf{\Lambda}$ is complete and

$$\mathrm{Ai} \to \lambda kw.\Gamma\Big(\lambda ra.\delta\Big(4,\delta\big(2,k\big(w,V(Nm(r))\big)\big)+\delta\big(2,Nm(r,a)\big)\Big)\Big),$$

then Ai$(\mathbf{\Lambda})$is an all inclusive ordinal logic. For, if $\mathbf{A}$ is in the extent of $\mathbf{\Lambda}(\mathbf{\Omega}_{\mathbf{A}})$ for each $\mathbf{A}$, and we put $\mathbf{\Omega}_{(\mathbf{L})} \to \mathbf{\Omega}_{V(\mathbf{L})}$, then I say that, if $\mathbf{B}$ is in the extent of $\mathbf{L}$, it must be in the extent of Ai $(\mathbf{\Lambda},\mathbf{\Omega}_{(\mathbf{L})})$. In fact, we see that Ai $(\mathbf{\Lambda},\mathbf{\Omega}_{V(\mathbf{L})},\mathbf{B})$ is convertible to

$$\Gamma\Big(\lambda ra.\delta\Big(4,\delta\big(2,\mathbf{\Lambda}\big(\mathbf{\Omega}_{V(\mathbf{L})},V(\mathrm{Nm}(r))\big)\big)+\delta\big(2,\mathrm{Nm}(r,a)\big)\Big),\ \mathbf{B}\Big).$$

[†] In the case of p we adjoined all of the axioms $(\exists x_0)$ Proof $[x_0,f^{(m)}0] \supset \mathfrak{F}$,where m is the G.R, of $\mathfrak{F}$; the Gödel theorem shows that *some* of them are unprovable in P.

$$\Lambda_P(H(\text{ Suc }(\lambda ufx.u(\mathbf{R}_1))))$$

is the same as the extent of

$$\Lambda_P(H(\text{Suc }(\lambda ufx.u(\mathbf{R}_2)))),$$

where $\lambda ufx.u(\mathbf{R}_1)$ and $\lambda ufx.u(\mathbf{R}_2)$ are two C-K ordinal formulae representing ω. We should have to prove that a formula interpretable as a number-theoretic theorem is provable in $C[\text{ Suc }(\lambda ufx.u(\mathbf{R}_1))]$ if, and only if, it is provable in $C[\text{ Suc }(\lambda ufx.u(\mathbf{R}_2))]$. Now $C[\text{ Suc }(\lambda ufx.u(\mathbf{R}_1))]$ is obtained from $C[\lambda ufx.u(\mathbf{R}_1)]$ by adjoining all axioms of the form

$$(\exists x_0)\ \text{Proof}_{C[\lambda ufx.u(\mathbf{R}_1)]}[x_0, f^{(m)}0) \supset \mathfrak{F}, \tag{9. 1}$$

where m is the G.R. of $\mathfrak{F}$, and C [Suc $(\lambda ufx.u(\mathbf{R}_2))$] is obtained from $C[\lambda ufx.u(\mathbf{R}_2)]$ by adjoining all axioms of the form

$$(\exists x_0)\ \text{Proof}_{C[\lambda ufx.u(\mathbf{R}_2)][x_0, f^{(m)}0]} \supset \mathfrak{F}. \tag{9. 2}$$

The axioms which must be adjoined to P to obtain $C[\lambda ufx.u(\mathbf{R}_1)]$ are essentially the same as those which must be adjoined to obtain the system $C[\lambda ufx.u(\mathbf{R}_2)]$: however the *rules of procedure which have to be applied before these axioms can be written down are in general quite different in the two cases*. Consequently (9.1) and (9.2) are quite different axioms, and there is no reason to expect their consequences to be the same. A proper understanding of this will make our treatment of question (b) much more intelligible. See also footnote ‡ on page 172.

Now let us turn to Λ_H. This ordinal logic is invariant. Suppose that $\boldsymbol{\Omega}$, $\boldsymbol{\Omega}'$ represent the same ordinal, and suppose that we have a proof of a number-theoretic theorem G in $P_{\boldsymbol{\Omega}}$. The formula expressing the number-theoretic theorem does not involve any odd types. Now there is a one-one correspondence between the odd types such that if $2m-1$ corresponds to $2m'-1$ and $2n-1$ to $2n'-1$ then $\boldsymbol{\Omega}(\mathbf{m},\mathbf{n})$ conv 2 implies $\boldsymbol{\Omega}'(\mathbf{m}',\mathbf{n}')$ conv 2. Let us modify the odd type-subscripts occurring in the proof of G, replacing each by its mate in the one-one correspondence. There results a proof in $P_{\boldsymbol{\Omega}'}$, with the same end formula G. That is to say that if G is provable in $P_{\boldsymbol{\Omega}}$ it is provable in $P_{\boldsymbol{\Omega}'}$. Λ_H is invariant.

The question (b) must be answered negatively. Much more can be proved, but we shall first prove an even weaker result which can be established very quickly, in order to illustrate the method.

I shall prove that an ordinal logic $\boldsymbol{\Lambda}$ cannot be invariant and have the property that the extent of $\boldsymbol{\Lambda}(\boldsymbol{\Omega})$ is a strictly increasing function of the ordinal represented by $\boldsymbol{\Omega}$. Suppose that $\boldsymbol{\Lambda}$ has these properties; then we shall obtain a contradiction. Let $\mathbf{A}$ be a W.F.F. in normal form and without free variables, and consider the process of carrying out conversions on $\mathbf{A}(1)$ until we have shown it convertible to 2, then converting $\mathbf{A}(2)$ to 2, then $\mathbf{A}(3)$ and so on: suppose that after r steps we are still performing the conversion on $\mathbf{A}(\mathbf{m}_r)$. There is a formula Jh such that Jh $(\mathbf{A},\mathbf{r})$ conv $\mathbf{m}_r$ for each positive integer r. Now let $\mathbf{Z}$ be a formula such that, for each positive integer n, $\mathbf{Z}(\mathbf{n})$ is an ordinal formula representing $\boldsymbol{\Omega}^{\mathbf{r}}$, and suppose $\mathbf{B}$ to be a member of the extent of $\boldsymbol{\Lambda}(\text{Suc}(\text{Lim }(\mathbf{Z})))$ but not of the extent of $\boldsymbol{\Lambda}(\text{Lim }(\mathbf{Z}))$. Put

$$\mathbf{K}^* \rightarrow \lambda a.\boldsymbol{\Lambda}(\text{Suc}(\mathbf{Lim}(\lambda r.\mathbf{Z}(\text{Jh}(a,r)))),\ \text{B});$$

then $\mathbf{K}^*$ is a complete logic. For, if $\mathbf{A}$ is dual, then

$$\text{Suc }(\text{Lim}(\lambda r.\mathbf{Z}(\text{Jh}(\mathbf{A},r))))$$

represents the ordinal $\omega^\omega + 1$, and therefore $\mathbf{K}^*(\mathbf{A})$ conv 2; but, if $\mathbf{A}$(c) is not convertible to 2, then

$$\text{Suc }(\text{Lim}(\lambda r.\mathbf{Z}(\text{Jh}(\mathbf{A},r))))$$

represents an ordinal not exceeding $\omega^c + 1$, and $\mathbf{K}^*(\mathbf{A})$ is therefore not convertible to 2. Since there are no complete logic formulae, this proves our assertion.

We may now prove more powerful results.

Index

Page numbers followed by "*f*" indicate figures, "*t*" indicate tables, and "*n*" indicate footnotes.

M

P

T