Foreword

Alan M. Turing is arguably the most influential person who shaped the domains of computing, artificial intelligence and digital forecast. The computing community worldwide celebrated 2012 as Alan Turing year to honour his contributions and celebrate his lasting scientific influence on computing and the impact of computing on science and society. Indian Academy of Sciences held a special session on 3 November 2012 during its 78th Annual Meeting in Dehradun. The session had the following presentations: R. K. Shyamasundar (TIFR): Computing legacy of Alan Turing; Manindra Agrawal (IIT, Kanpur): Turing machines and the development of complexity theory; Ramesh Hariharan (Strand Genomics Ltd): Patterns in biology and the program of life; V. Arvind (IMSc): Algorithmic randomness, real numbers and computability.

Current Science decided to publish a special section in honour of Alan Turing to provide his scientific influence to its readers. The section consists of the following papers: The computing legacy of Alan M. Turing by R. K. Shyamasundar; Turing and animal coat patterns by Ramesh Hariharan; Normal numbers and algorithmic randomness: a historical sketch by V. Arvind; A brief history of polynomial identity testing by Manindra Agarwal. Even though this paper was not presented at the symposium, it has been included as it traces the link of the important problem with the complexity theory that was highly influenced by Turing's work. The last paper by Jaikumar Radhakrishnan is an excerpt of his interview with people that included Stephen A. Cook (a distinguished computer scientist and a Turing Laureate, University of Toronto), Rohit Parikh (a distinguished logician, CUNY) and Manoj Gopalakrishnan (TIFR) that highlights impact of Turing's work from different perspectives.

R. K. Shyamasundar

The computing legacy of Alan M. Turing (1912–1954)

R. K. Shyamasundar*

School of Technology and Computer Science, Tata Institute of Fundamental Research, Homi Bhabha Road, Mumbai 400 005, India

Alan Turing is considered one among the 20th century's 100 greatest minds. The invention of stored-program universal computer by him, is arguably the most influential mathematical abstraction of the 20th century that changed the whole world for good. While this invention became one of the cornerstones of computer science, Turing was best known during his time as the genius who broke some of Germany's most secret codes during the war of 1939–45. His inventions and discoveries covered a wide spectrum of areas like logic and computability, cryptology, computer architecture, artificial intelligence, digital forecast, chemical morphogenesis, algorithm randomness. While he was a theoretician's theoretician, he had an immense practical outlook with a deep understanding of computing, including its impact on science and society. In this article, some of his pioneering contributions shall be highlighted in an accessible way.

Keywords: Artificial intelligence, automatic computing engine, computing legacy, universal computers.

Of the finest types of intelligence – human, artificial, and military – Turing is perhaps the only person to have made a world-changing contribution to all three.

Nature, Editorial, 2012

*e-mail: shyam@tifr.res.in

Introduction

IN 1999, *Time* magazine named Alan M. Turing among the 20th century's 100 greatest minds that included Albert Einstein, Wright brothers, Crick and Watson, as well as Alexander Fleming. Alan Turing a true pioneer, laid the foundations for digital computer, artificial intelligence, and made an astonishing forecast for the information age. He made pioneering contributions with respect to

a. Universal model of computation
b. Alan Turing's system of logic
c. Automatic computing engine (ACE)
d. Artificial intelligence
e. Forecast for the information age
f. Turing numbers – an instance of a fundamental entity in the theory of algorithmic randomness
g. A chemical basis for morphogenesis.

In this article, we shall cover (a) in some detail, followed by highlights from (b) to (e). We will not be touching upon (f)–(g) further in this article. Turing's legacy extends to other areas like biological pattern formation, limits of computation in physics, etc. All these achievements happened in his short lifespan of 42 years (Figure 1).

During his lifetime, Turing was known more as a genius who broke some of Germany's secret codes created by encrypting machines called ENIGMA that was responsible

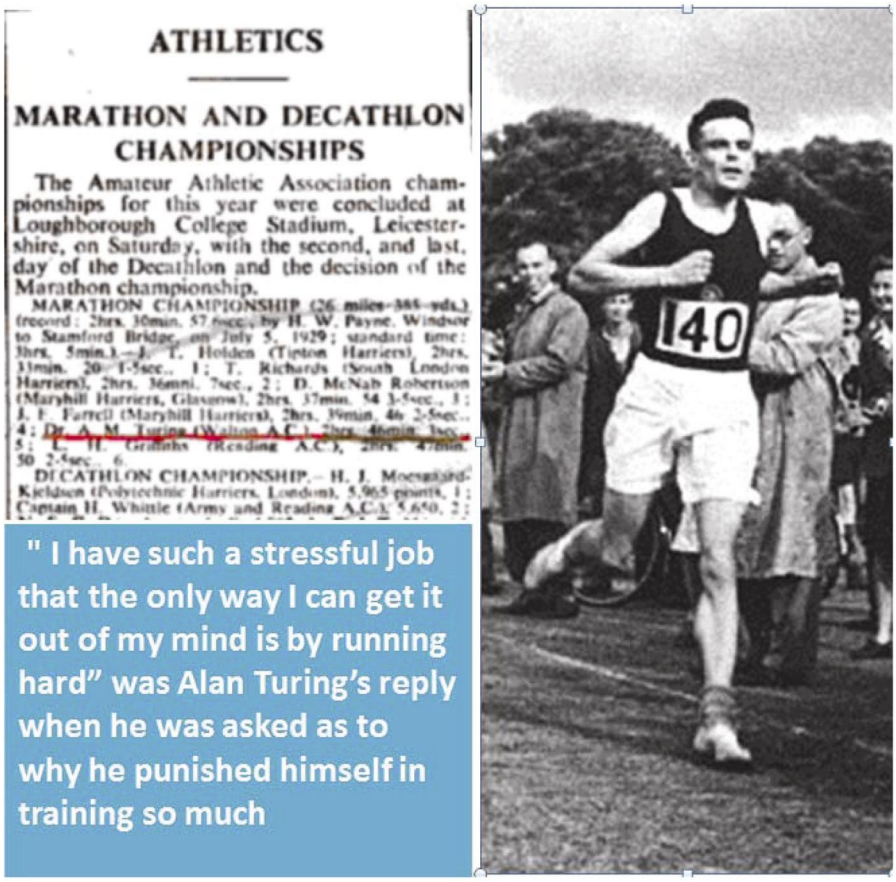

Figure 1. Alan M. Turing, the marathon runner.

for halting the war early. His overall contributions and impact can be seen aptly in the summary suggestion by the Editorial in *Nature* to declare the Alan Turing's centenary year as 'The Year of Intelligence', as the Editorial says 'Turing is perhaps the only person to have made a world-changing contribution to all the finest types of intelligence: human, military and artificial.'

Universal model of computation

In this section, we shall discuss the model of computation to convey underlying reasons as to why it has become the foundation of computer science.

Turing machine model of computation

Turing got exposed to the famous Hilbert's Entscheidungsproblem (the decision problem) in his undergraduate course at King's College, Cambridge by M. H. A. Newman. Following Leibniz's mechanical calculating engine, Hilbert dreamt that

- All mathematical formalisms should be encoded in some suitable logical formalism.
- An algorithm (or an effective procedure) needs to be arrived for determining the truth therein.

In logical terms, the question was: Whether there exists an effective method to decide, given a well-founded formula of the pure first-order predicate (note 1) calculus, whether or not its negation is satisfiable in some interpretation.

As Turing thought about the question, he was quite convinced that there cannot be any such procedure. One of the main hurdles for progress was the lack of definition of *finite procedure/effective procedure/algorithm*. He succeeded (note 2) in arriving at such a notion of effective procedure that it not only met the intuition that an effective procedure or an algorithm is nothing but a finite sequence of instructions that can be carried out mechanically, but also was mathematically precise.

He realized the above through the definition of a simple abstract mathematical machine[1,2] which has come to be known as Turing machine (TM) (Figure 2). We shall describe these in the sequel.

A TM is a primitive computation model essentially consisting of 'an unlimited memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment

there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings[1].' The graphical diagram denotes a typical TM with the finite state control along with the tape and tape head. In other words, the operations that this machine can perform be described as follows:

> Based on the state of the finite state control and the symbol under the read-scan head of the machine, it can move to another state (finite state control consists of finite nonempty set of states), change the symbol to any other symbol, and the scan head can perform any one of the following three moves: (1) move to the left by one square, (2) remain stationary, or (3) move to the right by one square.

It must be clear from the above description the movement of the machine is completely determined by the symbol under the scan head and the state of the finite state control.

Example: Consider a simple example of adding two numbers, say m and n. Given n and m on the tape, the TM machine should yield $n + m$ on the tape and halt.

Before we consider the description of the above machine, let us fix a notation required for the description of the TM.

Notation

1. Number n will be represented in unary and hence will be denoted by $n + 1$, '1's. Thus, '0' will be denoted by a single '1' and number 5 will be represented by a block of six '1's.
2. b is a separator between arguments.

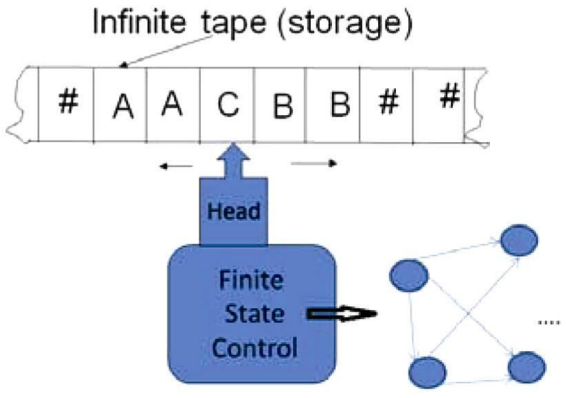

Figure 2. A Turing machine.

3. The state transition of the control will be denoted by a 5-tuple (current state, current symbol, next state, symbol written, movement of head), where movement of head is as follows: move left by one square (L), move right by one square (R), or remain stationaly (S). Initially, the TM head will be scanning the first '1' of the first argument.

The state transition diagram is shown in Figure 3.

The label of the edges denotes: (current read symbol, write symbol, movement of the head); in the diagram, the next state is indicated by the state at the end of that arrowhead. The state transition system has five states. Starting in state $s0$, it reaches the first b after the first block of '1's and changes the b to '1' and reaches state $s1$. At this point, the two blocks of '1's have been connected by a '1' and hence consists of a block consisting of $(n + 1) + 1 + (m + 1)$ '1's and the machine returns to the leftmost '1' entering state $s2$. Now that the remaining two extra '1's need to be removed, which is done in states $s2$ and $s3$ respectively, and enters the final state $s4$. The tape configuration shown below depicts the initial and final configurations for adding 3 and 2 on the left and right respectively. Note that the TM description assumes that there will be at least two '1's on the tape initially (Figure 4).

Remarks: It is evident from the above description that *any computation step described by a TM can be mechanically carried out* – this could be intrepreted to say that one can perform these operations on a paper and pencil.

Alonzo Church opined that any process which could naturally be called an effective procedure can be realized by a TM. Thus, the main thesis of computation takes the following form:

Church–Turing thesis: Turing machines are capable of solving any effectively solvable algorithmic problem. See Box 1 to appreciate the robustness of the thesis. Box 2, makes a qualitative comparison of the works of the two giants of compatibility: Alonzo Chruch and Alan M. Turing.

Universal computing machine (or universal turing machine)

The thesis discussed above naturally leads to the following question: does one need a different computing machine for every problem?

Turing showed that it is possible to invent a single machine that can be used to compute any computing sequence (or solve any effective procedure). The universal machine, say $\mathcal{U}$, can be constructed that has the capability of behaving like any other computable machine. $\mathcal{U}$ is like any other Turing machine, but its input is essentially the description of the given Turing machine along with its input on its tape as shown in Figure 5.

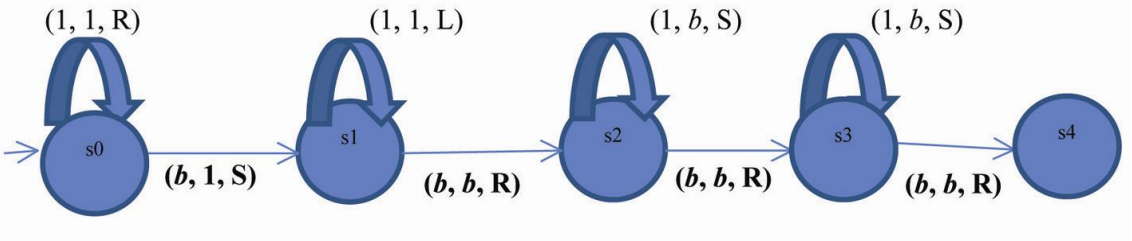

Figure 3. Finite state control of a Turing machine.

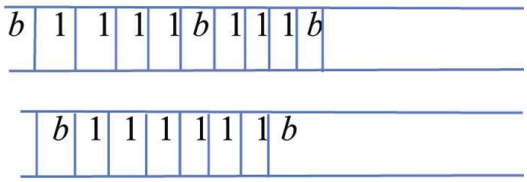

Figure 4. Initial and final configurations of an TM that adds two numbers, m and n.

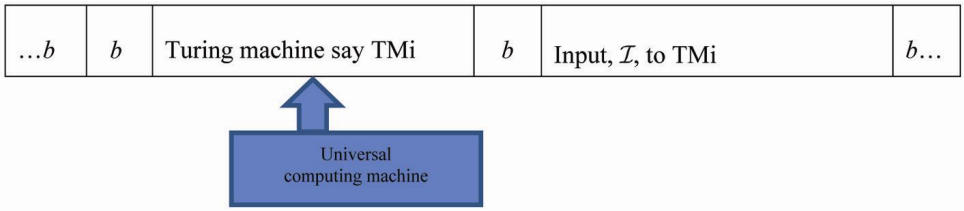

Figure 5. A universal Turing machine ($\mathcal{U}$).

Box 1. Robustness of Church–Turing thesis

- Church–Turing thesis implies that the most powerful supercomputer with the most sophisticated array of programing languages is no more powerful than a PC with a simple hardware and software.
- Both can solve precisely the same class of algorithmic problems.
- Non-computable problems are solvable by neither.
- If a new computer or a PL is designed, one needs to find reasons other than raw computing power.
- Solvability is extremely robust: It is invariant under changes of a programing language or computer hardware.

With the input as above, $\mathcal{U}$, it behaves exactly like TMi for its input, $\mathcal{I}$. That is, $\mathcal{U}$ yields exactly the same results as TMi on input $\mathcal{I}$; further $\mathcal{U}$ terminates if TMi terminates on $\mathcal{I}$ giving the results and does not terminate on $\mathcal{I}$ if TMi does not terminate on $\mathcal{I}$. Note that the simulation is done using polynomial transformations (b in the diagram denotes blank tape symbol).

The above results have a far-reaching consequence and can be interpreted as follows:

> Any algorithmic problem for which an algorithm can be found in any programing language on any computer (existing or that can be built in future) requiring unbounded amounts of resource is also solvable by a TM.

In other words, Church–Turing thesis implies that the most powerful supercomputer with the most sophisticated array of programing languages is no more powerful than a PC with a simple hardware and software up to polynomial loss in efficiency.

Computability and non-computability

In an earlier section, we considered an example of a TM with quintuples. Consider the quintuples underlying Figure 3 explicitly as given below:

($s0$,1,1,R,$s0$); ($s0$,b,1,S,$s1$); ($s1$,1,1,L,$s1$); ($s1$,b,b,R,S2); ($s2$,1,b,S,$s2$); ($s2$,b,b,R,$s3$); ($s3$,1,b,S,$s3$); ($s3$,b,b,R,$s4$).

It must be evident that the Turing machine is described clearly by these tuples. With a binary alphabet, let us use the unary system to represent symbols and numbers and encode each tuple with the following encoding: (i) '0' denotes a blank square on the TM tape denoted in unary by '1', (ii) n+1 '1's denote the number $n + 1$, (iii) state si denoted by number 'i' in unary, (iv) for the tape head movements let us represent no movement or stationary by

Box 2. Alonzo Church and Alan M. Turing

- In April 1936, Church proved the unsolvability of the Entscheidungsproblem of logic.
- Through λ-definability Church defined the notion of 'effective computability'.
- He proved effective unsolvability of various mathematical and logical problems.
- In Gödel's view, definition of effective computability was thoroughly unsatisfactory.
- Turing submitted the results in May 1936 'On computable numbers …' (ref. 1).
- Established the formal equivalence of λ-definability and the Turing definability in 1937.
- Church's review: 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. Of these, the first has the advantage of identification with effectiveness in the ordinary sense evident immediately… . The second and third have the suitability for embodiment in a system of symbolic logic.

11 (number 1), L by 111 (number 2) and R by 1111 (number 3), and (v) each component in the quintuple is separated by '0'. The first tupe is encoded by

…00 1 0110110111101 00… .

Now TM can be denoted by the set of tuples in some order separating each tuple from the other by '00' (i.e. two blanks). The corresponding encoding is given below:

…00 1 0110110111101 00 101011011011 00 11011011 0111011 00 110101011110111 00 111011010110111 00 11101010111101111 00 11110110101101111 00 11110101011110111 00.

The encoding is just a natural number and can be treated as a serial number of TM. Thus, every TM will have a distinct serial number. Since the serial numbers are natural numbers, the following result follows:

The number of distinct Turing machines is countable (note 3).

Above, we have shown how the whole TM can be encoded by a number. We also know that the number of functions on the natural numbers is uncountable. Thus, from the Church–Turing thesis, we can conclude:

There are non-computable functions.

In the following, an explicit example of a non-computable function due to Tibor Radó taken from Feferman *et al.*[3].

Busy beaver (BB) problem: We shall consider an example of a non-computable function (cf. ref. 4). Consider a Turing machine, T, that when started on a completely blank tape, eventually halts with some number of '1's. Productivity, p, of a TM is defined as follows:

- If TM T starting on a blank tape, leaves n, '1's then its productivity is n.
- If T does not halt then its productivity is 0.

Thus,

p: Turing machine descriptions $\rightarrow$ natural number.

There are several TMs with the same number of states. Thus, it is possible to define the notion of maximum productivity that a TM with that number of states, say k, can have. This new function will be:

natural numbers $\times$ the number of states $\rightarrow$ natural numbers.

Let us call the function BB. Thus,

BB(k) = n, implies the maximum productivity of a k-state Turing machine is n.

Further, it may be noted that there may be several different k state machines with maximum productivity n. Let us call any of these machines a *BB for k*.

Theorem 1. *There is no TM which will compute BB(k), i.e. which when started in a standard configuration on a tape with k '1's will halt in standard configuration on a tape with BB(k), '1's. This example is due to Tibor Rádo*[3].

Proof: Informally, the problem can be interpreted as follows: There is no TM that prints the maximum number of ones an N-state halting TM can write when started on a blank tape. Let us assume on the contrary that there is a machine B for the BB problem that has k states. Now, make the following construction:

1. Construct a TM, say A, with n states which writes n '1's on an initially blank tape.
2. Construct a new machine, say C, which connects the halting state of A to the start state of B.
3. Connect the halting state of B to the start state of another copy of B.

First, A writes n '1's , followed by the first copy of B computing BB(n), and then the second copy of B takes over and computes BB(BB(n)). The total number of states in our machine is $n + 2k$. Our machine may be a busy beaver for $n + 2k$, but it is certainly no more productive than such a machine. Thus, if the busy beaver machine exists, we have

$$\text{BB}(n + 2k) \geq \text{BB}(\text{BB}(n)), \quad \text{for any } n.$$

It is easy to see that the productivity of Turing machines increase as states are added, i.e.

$$\text{if } i < j, \text{ then } \text{BB}(i) < \text{BB}(j).$$

Consequently, if B exists, then $n + 2k \geq \text{BB}(n)$, for any n. Since this is true for any n, it is true for $n + 11$, yielding

$$n + 11 + 2k \geq \text{BB}(n + 11), \quad \text{for any } n.$$

But it is easy to show that $\text{BB}(n + 11) \geq 2n$ by constructing a TM that has 11 states for doubling the number of '1's on the tape, and composing such a machine with the n-state machine for writing n '1's. Combining this fact with the previous inequality we have

$$n + 11 + 2k \geq \text{BB}(n + 11) \geq 2n, \quad \text{for any } n.$$

That is, $11 + 2k \geq n$, for any n must be true if the busy beaver exists.

This is an obvious contradiction of the hypothesis of existence of TM for BB(k). □

The halting problem: The halting problem of Turing machines can be stated as follows: Given a Turing machine, TM, with a finite number of non-blank tape symobols in any configuration, will the TM eventually halt? Unfortunately, this problem is unsolvable. This was first proved by Turing in 1936 (ref. 1). In the context of computer programing, the result can be stated informally as follows

> There is no computer program that can examine the code for a program and determine whether that program halts.

The unsolvability of the halting problem is formally stated below.

Theorem 2. *There is no algorithm to determine if a Turing machine in an arbitrary configuration will halt eventually.*

Proof sketch[3]*:* On the contrary let us assume that there is such a machine, say H. As UTM is yet another TM, let H be a UTM. Thus, $H(t, n)$ simulates the behaviour of TM T on input 'n'.

Let us construct a new machine using a copier TM as follows:

a. Let C be the copier machine which when started with a single block of '1's, halts after writing two copies of that block of '1's separated by a '0'. Thus, the number of '1's will be $2 \times n$, where n is the number of '1's in one block on the tape initially.
b. Now construct a new composed machine, say M, by joining the halt state of C to the start state of H.

   - Such a machine, when started on a tape with a block of n '1's, first modifies the tape to consist of two blocks of n '1's separated by a '0'. Because of construction, the machine H will have two blocks of n '1's separated by '0' and hence needs to answer the questions as to when a TM described by number 'n' will halt on an input 'n'.

c. Let us construct a machine, say N, that makes an infinite sequence of transitions if the tape contains TRUE when it starts, and halts if the tape contains FALSE. It is easy to construct such a machine using the encoding: TRUE is represented by '11' and FALSE by '1'.
d. Let us connect machine N with the halt state of H. The modified composed machine M halts if the machine with the input code n does not halt on an initial tape containing n (because if machine n does not halt on n, the halting machine will leave TRUE on the tape, and M will then go into its infinite sequence), and vice versa.
e. The impossibility of such a machine may be seen by considering the code for the modified M itself. What happens when M is started on a tape containing M's code? Assume that M halts on M, then by the definition of the machine M, it does not halt. But equally, if it does not halt on M, the definition of M says that it should halt.

This is a contradiction, and thus, there cannot be any TM that solves the halting problem. □

Note: The reader should note that it is possible to find a solution whether a specific Turing machine from a specific configuration can halt. The unsolvability refers to the general problem as stated above.

Remarks: There are several variations of Turing machines like: (i) allowing multiple tapes, (ii) multi-dimensional tapes, (iii) arbitrary movement of heads (rather one square to the left or right), etc. However, the expressive power of these enriched machines remains invariant. However, if you restrict the ability of the TM, then its expressive power could get reduced. Suppose we remove the ability to write, then the expressive power gets reduced to that of finite state machines.

In the next section, we shall discuss the way Turing defined computable numbers and sketch the way he established the unsovability of the Entscheidungs problem.

Computable numbers and the Entscheidungs problem

You can build an organ which can do anything that can be done. But you cannot build an organ which tells you whether it can be done.

J. von Neumann

Turing's original motivation was characterization of computable numbers and to show the unsolvability of Hilbert's Entscheidungsproblem.

> According to Turing, a number is said to be computable if its decimal can be written down by a machine described above.

He starts by constructing machines for computing sequences. For instance, consider the sequence 010101… . It is easy to observe that in the sequence, between every two '0's there is a '1'. It is is easy to understand that we can construct a TM for computing such a sequence starting from a blank tape. Similarly, the sequence, 00101101110111011111… can again be computed by Turing machine, noting that the number of '1's between two '0's increases by '1' as we proceed up the sequence.

Let *machine configuration* be interpreted to mean the state of the finite control, the symbol under the scan head and the complete contents of tape that are not blank. From each such configuration, one can have the following possibilities: (a) the machine can move based upon the symbol being scanned and the state of the machines, and (b) there is no possible move by the machine. Notice that the moves of the machine are entirely determined by the configuration. In terms of the moves, one can say: (a) the machine makes a finite number of moves and reaches a configuration from which there is no other possible move, or (b) the machine moves from one configuration to another forever. In case (b), one can have two interpretations: (1) is it printing any useful information at all, (2) there is no useful information being printed. For a clear understanding, let us use the following terminologies.

Definition 1. There are two kinds of symbols that will be written on the tape: (a) numbers 0 or 1 reflecting the value of a number at that configuration – referred to as *type-1 symbols*, and (b) symbols written to capture some memorization or inference on the configurations reached so far – referred to as *type-2 symbols*.

Definition 2. If the machine is supplied with a blank tape and set in motion, starting from a correct initial configuration, the subsequence of type-1 symbols printed by it is called the *sequence computed by the machine.* The real number whose binary expression as a binary decimal is obtained by prefixing the sequence with the decimal point is called *the number computed by it*.

Definition 3. A computing machine is said to be *circular* if it reaches a configuration from which there is no possible move, or it has moves forever printing only symbols of type-2 and unable to print any symbols of type-1. Otherwise, the machine is said to be *cycle-free*.

Using the above definitions, computable numbers are defined as follows.

Definition 4. A sequence over say '0's and '1's is said to be *computable* if it can be computed by a circle-free machine.

Definition 5. A number is said to be *computable* if it differs by an integer from the number computed by a cycle-free machine.

The above definitions lead to

Definition 6. A number is *computable* (Turing computable) if there exists a Turing machine starting from a blank tape that computes an arbitrarily precise approximation to that number.

Note as proved earlier that compuatble numbers are enumerable.

Noting that one cannot define general computable functions of a real variable as there is no general method of describing a real number, Turing showed that large classes of numbers are computable utilizing recursive definitions using integral variables and computable variables.

Some examples of computable numbers

1. Let $\varphi(m, n)$ be computable, where m and n are integral numbers and r is some integer. Then, $\eta(n)$ is computable where

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

2. If $\varphi(m, n)$ is a computable function of two integral variables, then $\varphi(n, n)$ is a computable function of n.
3. If $\varphi(n)$ is a computable function whose value is always 0 or 1, then the sequence whose nth figure is $\varphi(n)$ is computable.
4. It must also be noted that the Dedekind's theorem (note 4) does not hold by replacing 'real' by 'computable' throughout. However, 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. This is formally stated below.

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

a. $(\exists\alpha)\,(\exists\beta)\,\{G(\alpha)$ and $(\neg G(\beta))\}$,
b. $(G(\alpha)$ and $G(\beta)) \rightarrow (\alpha < \beta)$;
$\rightarrow$ denotes 'implication',

and there is a general process for determining the truth value of $G(\alpha)$, then there is a computable ξ such that $G(\alpha) \rightarrow \alpha \leq \xi$ and $\neg G(\alpha) \rightarrow \alpha \geq \xi$.

5. Computable convergence: A sequence β_n of a computable numbers *converges computably* if there is a computable integral valued function $N(\varepsilon)$ of the computable variable ε, such that if $\varepsilon > 0$ and $n > N(\varepsilon)$ and $m > N(\varepsilon)$, then $|\beta_n - \beta_m| < \varepsilon$. Using this definition, the following real numbers are shown to be computable:

a. A power series whose coefficients form a computable sequence of computable numbers is computably convergent at all computable points in the interior of the interval of convergence.
b. The limit of a computably convergent sequence is computable. From this result and the series $\pi = 4(1 - 1/3 + 1/5 - \ldots)$, it is possible to deduce that π is computable. Similarly, from the series for e it can be shown to be computable.
c. In similar ways, Turing established that a large class of numbers such as real parts of algebraic numbers, real parts of the zeroes of the Bessel functions, etc. are all computable.

Unsolvability of Entscheidungsproblem

What Gödel had established: Gödel had shown that in the formalisms like *Principia Mathematica*, there are propositions, say P, such that neither P nor $\neg P$ is provable. As a result, it was shown that no proof of consistency of Principia Mathematica can be given within that formalism.

What Turing established: There is no general method whether a given formula P is provable in the restricted Hilbert's functional calculus (note 5).

It must be noted that if the negation of what Gödel has shown had been proved, then there would have been a solution to the Entscheidungsproblem (Boxes 1 and 2).

Theorem 3. *Entscheidungsproblem is unsolvable.*

Proof idea: For each TM, $\mathcal{M}$, he constucted a formula $\mathcal{F}(\mathcal{M})$, and showed that if there is a general method for determining whether $\mathcal{F}(\mathcal{M})$ is provable, there is a general method for determining whether $\mathcal{M}$ ever prints 0. The latter is impossible by the halting problem described above and hence the unsolvability of the Entscheidungsproblem. □

Seeing computing everywhere

One of the hallmarks of Turing was that he was seeing computation everywhere. He firmly established in a variety of ways that traditional mathematical concepts specified by finitely definable approximation, such as measure or continuity could be made computational. Some of his notable contributions of significance explicitly to computing have been

- LU decomposition;
- Finite approximations of continuous groups (of interest to John von Neumann);
- Computation over the reals (which has taken deep roots recently), etc.

In fact, he had also worked on devising a method for the practical computation of zeros of Riemann zeta function during a vacation break at Cambridge from his Ph D work at Princeton.

In the following, we touch upon the rationale and main result of his Ph D thesis (ref. 5; appears in ref. 6 and recently reprinted as ref. 5).

System of logic based on ordinals: Turing worked for his Ph D at Princeton under the supervision of Alonzo Church. He started wondering about the ways in which Gödel's incompleteness theorem can be confined.

He started with the argument that if there are already informal or intuitive reasons for accepting the axioms of the system to be true, then one ought to accept the statement of its consistency as a new axiom. Then apply the same considerations to the new system by iterating the process of adding consistency statements as new axioms. As Teresa Numerico[7] states, Turing tried to repair the problems created by Gödel's incompleteness theorem for the formalistic approach to mathematics.

In his thesis, Turing established that Gödel's incompleteness can be overcome for an important class of arithmetical statements (though not for all) and investigated the process systematically by iterating it into the constructive transfinite taking unions of logical systems at ordinal notations. Turing proposed the construction of succession of 'ordinal logics'. Each of the ordinal logics in the hierarchy for each ordinal number up to some transfinite ordinal included the Gödel statement '*I am unprovable in L*' for the previous logic L in the hierarchy. He opined that with a complete ordinal logic one should be able to confine the non-mechanical steps entirely to verifications that particular formulae are ordinal formulae

(it could lead to automatic refereeing). But the question is: how easy is it to verify the ordinal formula.

In this context, Turing reflects on the role of intuition and heuristics in mathematical discovery. To quote '… In consequence of the impossibility of finding a formal logic which wholly eliminates the necessity of using intuition, we naturally turn *non-constructive* systems of logic with which not all the steps in a proof are mechanical, some being intuitive.' *According to Turing, it was unlikely that interesting theorems would be demonstrated in mathematics without the use of intuition and heuristics.*

Building computing engines

While Turing is a celebrated genius for his theoretical contributions through his writings, he has also contributed immensely to the development of computing engines or computers. Before going with his involvement with building computer systems, let us first understand that the *stored programing concept* often attributed to John von Neumann was really the concept embedded in Turing's universal machine. The Box 3 highlights these aspects succinctly.

Turing's first involvement with actual computing started with the Colossus – a code-breaking computer developed and built at Blechley Park during World War II (Figure 6). Turing completed the logical design of the famous Bombe, built to break German ENIGMA messages towards the end of 1939. This was built by the British Tabulating Machine Company in Letchworth. It was essentially a computing engine with the limited scope to find the positions at which the German message has been encrypted that would lead to likely candidates that could be tested on an copy/replica of an ENIGMA machine; if German text emerged (even if only a few words along with some nonsense), the candidate settings were treated as the right ones. The design of Bombe was based on electromagnetic relays. A large-scale electronic machine, Colossus (Figure 6) was built in 1943 that would get settings from computing engines like Bombe and decipher and print out the German text. The architecture of Colossus had borrowed ideas from Turing's paper and the machine had flexibility, it was far from using the stored program concept as in the UTM. While working with Colossus, both Turing as well as M. H. A. Newman (who had been consulting on Colossus) realized that the idea of stored program as available in UTM could be constructed for the future machine and thus realize a real automatic universal machine.

Box 3. Stored programing concept

- The concept usually covers the following three scenarios
  - Instructions can be stored in memory as numbers. Gödel's arithmetization technique used by Turing in UTM is an example of this aspect.
  - Instructions and data can be stored as numbers and thus there is no distinction between the two. This feature is an integral part of the UTM definition.
  - Instructions expressed in the language of numbers can be manipulated like any other numbers, leading to the idea of program modification.
- Thus, one can say that John von Neumann engineered Turing's ideas of programs as data to realize the first stored program computer often referred to as von Neumann machines.

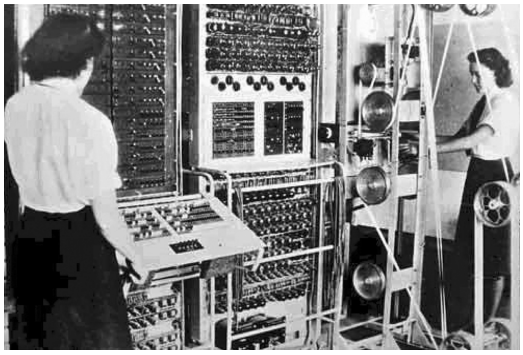

Figure 6. The Colossus, 1943.

In June 1945, J. R. Womersley, Head of Mathematics Division at NPL, wanted Turing to design a computer called automatic computing engine[7] (ACE); the word engine was a deliberately borrowed from Charles Babbage. Turing gave the proposal for ACE (Figure 7) before the end of 1945 and wanted it to be electronic, using binary numbers with 1 MHz clock using mercury delay line storage. Turing expected an addition of 32 bit numbers requiring 32 μs and 32-bit multiplication requiring over 2 ms.

Two important characteristic features of Turing's design are summarized below:

- It had a small number of primitive instructions mostly for transfers between memory and registers – resembled reduced instruction set computers (RISC) which came later in 1990s. It can be seen that Turing's concern was 'speed'.
- Use of a stack and its instructions like PUSH and POP called BURY and UNBURY; in a sense, Turing was the first to use the stack in a computer architecture.

Some of the highlights from his lecture to the London Mathematical Society, 20 February 1947 (ref. 8) are quoted below:

- … machines such as ACE are in fact practical versions of the universal machine …
- … the complexity of the job the machine must do is concentrated on the tape – (that is software) and does not appear in the universal machine in any way.

- I believe that the provision of proper storage is the key to the problem of the digital computer, and certainly if they are to be persuaded to show any sort of genuine intelligence much larger capacities than are yet available must be provided. In my opinion, this problem of making a large memory available at reasonably short notice is much more important than that of doing such as multiplication at a high speed.
- On self modifying capability – It would be like a pupil who had learnt much from his master, but had added much more by his own work. When this happens, I feel that one is obliged to regard the machine as showing some intelligence.

From the above, it must be clear how Turing had the whole of computing in his design. He had introduced subroutines and interpretive codes which became common in later designs. A broad impact of Church and Turing on the programing language area is summarized in Box 4.

Turing getting frustrated with the progress of ACE, Turing moved to University of Manchester (note 6) in September 1947, where MARK I was being built. This was later marketed by Ferranti Limited commercially. He was responsible for the programing aspects. He defined programing as 'an activity by which a digital computer is made to do a man's will, by expressing this will suitably on punched tapes'. MARK I used 40-bit words and could store 20-bit instructions. These words were displayed on CRTs in 5-bit groups. The encoding was done by a base-32 notation, where each 5-bit code was represented by a teleprinter character corresponding to that code. Thus, the code for a character was necessary to be known by each programer.

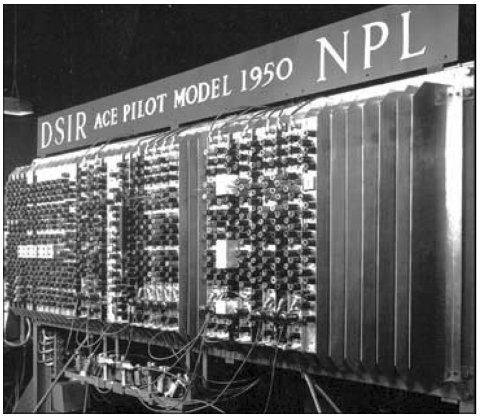

Figure 7. Pilot model of the automatic computing engine.

Box 4. Programing languages: Church vs Turing

- Turing machines simulate λ-calculus.
- Descendants of Church's notation work better than those of Turing's.
- Lisp and ALGOL programers: There was a need to split between Church and von Neumann style (due to limitations of computers).
- With progress in technology, formalisms such as ML, Haskell have had impact and are consciously closer to λ-calculus.

Artificial intelligence and digital forecast

' ... If a machine is expected to be infallible, it cannot also be intelligent'

A. M. Turing, London Mathematical Society Address, 20 February 1947.

Turing test[9,10]

While Turing described a machine capable of computing all effectively computable functions, he formulated a test which has come to be known as Turing test for testing *normal human intelligence* (Figure 8). The Turing test is an imitation game played by three people. In this game, a man and a woman are in one room, and a judge is in another room. The three cannot see one another, so they communicate via e-mail. The judge questions them for 5 min, trying to discover which of the two is the man and which is the woman. This would be easy, except that the man lies and pretends to be a woman. The woman tries to help the judge. If the man is a really good impersonator, he can fool the judge 50% of the time. But, it seems that in practice, the judge is right about 70% of the time.

Now, the Turing test replaces the man with a computer pretending to be a human. If it can fool the judge 30% of the time, it passes the Turing test. The rationale of the test may be seen in Turing's own words in his BBC interview: 'The idea of the test is that the machine has to pretend to be a man, by answering questions put to it, and it will only pass if the pretence is reasonably convincing … . We had better suppose that each jury has to judge quite a number of times, and that sometimes they really are doing with a man and not a machine. That will prevent them from saying "It must be a machine" every time without proper consideration.' The underlying arguments against the test can again be seen in his own words: 'The game may be criticized on the ground that the odds are weighted too heavily against the machine. If the man were to try and pretend to be the machine he would clearly make a very poor showing. He would be given away at once by slowness and inaccuracy in arithmetic. *May not machines* carry out something which ought to be described as

Figure 8. Turing test.

thinking but which is very different from what a man does? This objection is a very strong one, but at least we can say that if nevertheless, a machine can be constructed to play the imitation game satisfactorily, we need not be troubled by this objection.'

Even though it is not formally defined, it is a practical test applied to an existing entity that is 'running'. It consists of a conversation over a period of time between the tester and the entity being tested. This demands an ability to learn and adapt the contents and the structure of the sayings of the tester. Note that the testing becomes harder the longer it goes on. The point of the test is that if some entity passes it, it is hard to deny that it is intelligent and hence throws up the possibility of judging artificial entity to be intelligent. The basis for this is based on Turing's view that 'thinking is singularly and critically indicated by verbal behaviour indistinguishable from that of people as determined by a blinded experiment'. In summary, Turing test shares important properties with interactive proofs such as exponentially rare false positives, non-composability, non-transferability, etc. Turing's seminal contribution was in enabling blinded controls. While the test can provide an interactive proof of intelligence, it is not particularly useful as a research goal itself. While several internet sites offer Turing test chatterbots, none of them had passed the test till recently. On 14 June 2014, a new version of a chatbot developed by PrincetonAI (a small team of programers and technologists not affiliated with the Princeton University, and backed by computer and gee-whiz algorithms referred to as 'Eugene Goostman' was able to fool the Turing test 2014 judges 33% of time – good enough the threshold set by Alan Turing in 1950. For details the reader is referred to the competition on Loebner Prize (see http://www.bbc.com/news/technology-27762088, 9 June 2014).

Turing's work on artificial intelligence (AI) and his reflections/reactions on questions of 'definitions of AI' remind me of the views of Richard Feynman in his John Danz Lecture in 1963 on 'Uncertainty of science' quoted below:

> '… I think that extreme precision of definition is often not worthwhile, and sometimes it is not possible – in fact mostly it is not possible… . I am not going to say that everything has to be done the same way when a method of testing different from observation is used. In a different field perhaps, it is not so important to be careful of the meaning of words or that the rules be specific, and so on. I do not know.'

Digital forecast

Trying to understand the way nature works involves a most terrible test of human reasoning ability. It involves subtle trickery, beautiful tightropes of logic on which one has to walk in order not to make a mistake in predicting what will happen.

Richard Feynman, John Danz Lecture, 1963 (note 7)

It is well worth analysing Turing's forecast done in his paper 'Computing machinery and intelligence'. To quote:

> 'I believe that in about fifty years' time it will be possible, to program computers, with a storage capacity of about 10^9, to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning. The original question, "Can machines think?" I believe to be too meaningless to deserve discussion. Nevertheless I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted.'

To quote from the analysis of Turing's predictions done by the Turing laureate Jim Gray:

> 'With the benefit of hindsight, Turing's predictions read very well. His technology forecast was astonishingly accurate, if a little pessimistic. The typical computer has the requisite capacity, and is comparably powerful. Turing estimated that the human memory is 10^{12} and 10^{15} bytes, and the high end of that estimate stands today. On the other hand, his forecast for machine intelligence was optimistic. Few people characterize the computers as intelligent. You can interview Chatter Bots on the Internet (http://www.loebner.net/Prizef/loebner-prize.html) and judge for yourself. I think they are still a long way from passing the Turing test. But, there has been enormous progress in the last 50 years, and I expect that eventually a machine will indeed pass the Turing Test. To be more specific, I think it will happen within the next 50 years because I am persuaded by the argument that we are nearing parity with the storage and computational power of the mind. Now, all we have to do is understand how the mind works (!).'

While there are several examples wherein computers have assisted in arriving at proofs of several open problems in mathematics, including the four colour conjecture and also IBM's Deep Blue computer succeeded in beating Gary Kasparov, the world chess champion, we are still far away from building intelligent machines. In all the above intelligent tasks enumerated, computers have essentially acted as tools rather than forming new concepts.

Implicit in the Turing test, are two sub-challenges that in themselves are quite daunting: (1) read and understand as well as a human, and (2) think and write as well as a human. Both of these appear to be as difficult as the

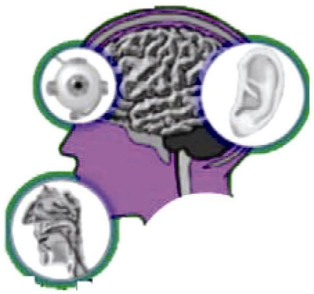

Figure 9. Three prosthetic challenges: vision, hearing and speech.

Turing test itself. Due to advances in computing technology, there has been tremendous progress in speech recognition, understanding, speech synthesizers, limited language translation, visual recognition, visual rendering, etc. While one may say the conceptual progress in these areas is limited, it is still a boon to the handicapped and in certain industrial settings. There is no doubt that these prosthetics will help a much wider audience and shall revolutionize the interface between computers and people (Figure 9).

When computers can see and hear, it will break communication barriers. It should be much easier and less intrusive to communicate with them. In a sense, it will allow one see better, hear better and remember better. Cell phone innovations are typical examples of these impacts which we have started seeing already.

Final remarks

Alan Turing's story is amazing. Starting from establishing the impossibility of Hilbert's dream, he founded computer science. His view of computing was 360°. His designs were beyond the times and cropped up much later almost as new discoveries. Turing's discoveries and inventions did not just confine to computing engines; his contributions spanned the theory of algorithmic randomness, chemical basis for morphogenesis and nonlinear dynamic simulation. His abstraction of intelligence through Turing test has had an impact in a variety of ways on the human–machine interface and also on human prosthetics. The world aptly celebrated his centenary and paid tribute to one of the greatest scientific minds of all times.

Notes

1. Readers who are not familiar with logic, may refer to any logic textbooks or even Wikipedia to understand these terms.
2. Gödel in his Gibbs Lecture says 'the greatest improvement was made possible through the precise definition of the concept of finite procedure, which plays a decisive role in these results. There are several different ways of arriving at such a definition, which, however all lead to exactly the same concept. The most satisfactory way, in my opinion, is that of reducing the concept of finite procedure to that of a machine with a finite number of parts, as has been done by the British Mathematician Turing' (Gödel, 1951, pp. 304–305) in Feferman[2]. It is also of interest to see Gödel's remark later in 1972 about Turing's argument saying mental procedures cannot go beyond mechanical procedures. To quote Gödel's remark[11]: 'What Turing disregards completely is the fact that *mind in its use, is not static, but constantly developing*, i.e. that we abstract terms more and more precisely as we go on using them, and that more and more abstract terms enter the sphere of our understanding'. Turing was well aware of these aspects become quiet evident if one analyses Turing's work on machine intelligence.
3. Readers not familiar with these terms may refer to countable and uncountable sets in elementary logic books or Wikipedia. Note that countable does not mean finite.
4. The unfamiliar reader is referred to any elementary real-analysis book.
5. First-order calculus.
6. MARK I was the first EDVAC-type electronic stored-program computer to be completed in the UK (June 1948).
7. In this era of machine learning, it would indeed be nice to read, Richard Feynman, *The Meaning of it All*, Penguin Books, 1998.

1. Turing, A. M., On computable numbers, with an application to the Entscheidungs problem. *Proc. London Math. Soc., Ser. 2*, 1936, **42**, 230–265, also reprinted in Davis[6].
2. Turing, A. M., On computable numbers, with an application to the Entscheidungsproblem: a correction. *Proc. London Math. Soc., Ser. 2*, 1938, **43**(6), 544–546, also reprinted in Davis[6].
3. Feferman, S. *et al*. (eds), *Collected Works of K. Gödel*, Oxford University Press, Oxford, 2001, vols I–III.
4. *Stanford Encyclopedia of Philosophy*, plato.stanford.edu/entries/turing-machine
5. *Alan Turing Systems of Logic*, The Princeton Thesis, Princeton University Press, Princeton, NJ, 2012.
6. Davis, M. (ed.), *The Undecidable: Basic Papers on Undecidable Propositions, Undecidable Problems and Computable Functions*, Raven Press, New York, 1965.
7. Jack Copeland, B., *Alan Turing's Automatic Computing Engine*, Oxford University Press, 2005.
8. Petzold, C., *The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine*, Wiley Publishing, Inc., 2008.
9. Turing, A. M., *Intelligent Machinery*, Reprinted in Cybernetics: Key Papers (eds Evans, C. R. and Robertson, A. D. J.), University Park Press, Baltimore, 1948.
10. Turing, A. M., Computing machinery and intelligence. *Mind*, 1950, **LIX**, 433–460.
11. Shagir, O., Gödel on Turing on Computability. In *Church's Thesis after 70 years* (eds Olszewski, A., Wolenski, J. and Janusz, R.), Ontos-Verlag, 2006, pp. 393–419.

ACKNOWLEDGEMENTS. This article is based on several lectures given by the author during the Turing centenary year in India as well as a special session organized by him in the annual meeting of the Indian Academy of Sciences, Bangalore at Dehradun in 2012. The article has been written to convey the depth and beauty of Turing's contributions both to the uninitiated reader as well persons exposed to computer science. Intentionally, some of the definitions given by Turing are preserved as that would provide the context of challenges and the rationale of research of various areas. There have been several expositions by a large number of authors spread over a period of time and the author is indebted to them. I thank my colleagues Prof. Jaikumar Radhakrishnan and Dr N. Raja for the feedback on the initial manuscript and Prof. Y. N. Narahari (Indian Institute of Science, Bangalore) for being patient enough in receiving the article.