14

Turing Transcendent: Beyond the Event Horizon

P.D. Welch

14.1 The beginning

Turing's seminal 1936 paper 'On computable numbers, with an application to the *Entscheidungsproblem*' is remarkable in several ways: firstly he had set out to solve Hilbert's decision problem of the title, with the definitive discussion of what a 'human computer' could do, and what would constitute an 'effective process', but secondly because it, well one cannot say 'almost in passing' because Turing was keen to put his observations on mechanical processes to work, but it was a second product of the paper that it lay also the foundations for the not yet even nascent theory of 'computer science'. The main point as Turing saw, was the 'universality' of his machine: there is a universal program or machine that can simulate any other. The story of the development of that science from Turing's work is retold elsewhere, and is not the aim or direction of this article. Nor are we going to trace the remarkable history of the theory of 'recursive functions' (now reclaimed as 'computable functions') thence until the present day. That story would involve not just the study of the Turing complexity of sets of numbers, and the computably enumerable sets, but later generalisations in 'generalised recursion theory' or 'higher type computability theory' where deep theoretical analyses of notions of *induction* and logical *definability theory* came into being in the late 1960s and 1970s. Such theories abstracted away from any machine model to notions of recursion or induction on abstract structures. These results are beautiful and very deep, and turn out to have intimate connections with mathematical logic and set theory.

Whilst this high road of increasing abstraction is very appealing intellectually, we have recently seen a flurry of research work drawing its inspiration from the original conception of Turing's machine model, and which tries to work with that, generalising what it could, in theory, compute when released from spatial and tem-

Published in *The Once and Future Turing*, edited by S. Barry Cooper & Andrew Hodges. Published by Cambridge University Press

poral boundaries. This article seeks to give some flavour of those ideas. Whilst it has to be said that a stretch of the imagination is required to send a computer off into a black hole, or to transcend ordinary time with its clock ticks of $0,1,2,\ldots$ into some transfinite stage 'ω' or even further beyond, thus 'transcending' the finite, this is a *mathematical* activity (or perhaps one in physics) where, as ever, we seek to further our knowledge of a theory, or of a model, its procedures, its limitations, by the process of generalising it. We try to relax current constraints purely 'to see what happens if ...'. It may be science fiction about Turing's machine, but it is a *mathematical* science fiction, and as such also has its boundaries. This kind of work perhaps argues less about the real, or even plausible, physical situation, but rather seeks to draw a containing line along the mathematico-logical boundaries to these kinds of speculations. It is, moreover, simply entertaining to think about these matters.

A computation is considered usually as a process where after a finite time we expect some output for an input. We are interested in the *result* of some sequence of manipulations of strings of symbols, perhaps (but not necessarily) coding numbers. This does not rule out continuous ongoing processes that machines can perform, when machines, or networks of machines, are designed to look after some system (such as the Web or other network), and in effect never finish their tasks, and are not turned off. When one observes closely, these machines are running processes that call for other 'subprocesses' which indeed do effect some computation and return some result. Such machines are administrating millions, if not more, computations of the kind to which I am alluding. Such a process can *interact* with its environment by asking for or receiving input following a query. Even the most complex of such interactive processes could be modelled by Turing's o-machines, or 'oracle-machines', which we shall take as the basic model for this article. For the purposes of this discussion such a machine has a single tape with a leftmost starting cell C_0 with further square cells to the right $C_1, C_2, \ldots$ We do not wish to bound in advance the machine's capabilities or requirements of space, so it is usual to allow *infinitely many* cells on the tape: $\langle C_i \mid i \in \mathbb{N} \rangle$. A read/write ('R/W') head moves back and forth a single cell at a time, reading *cell values* which we take from a simple alphabet of 0s and 1s. It has the capability of changing a read cell value, and then moving on. The *program* of such a machine is traditionally thought of as a sequence of tuples in a table, involving an initial *state* with each line of the table instructing the machine what to do when observing a particular symbol in a particular state, and to what state to change. There are only finitely many states, and hence only finitely many possible transitions from which to assemble a program. Programs themselves are finite lists of these transition instructions. How these instructions are written down is not important for this article, so we shall not go further into this. We shall note however that a single line, and thence a whole

program, can be *coded* by a single number e. There are many ways of doing this: first assign code numbers to instructions, and then use prime powers to code finite lists of such numbers. The fundamental theorem of arithmetic is then used to decode from a natural number e the program (if any) that it codes up. This is itself a computable process – and this last fact is important for Turing's result that there is a *universal machine*. We shall refer to this list of programs as $\langle P_e \mid e \in \mathbb{N} \rangle$ (where we have harmlessly assumed here that every $e \in \mathbb{N}$ codes a program). An *oracle machine* allows for a second tape on which information is written – again a string, possibly infinite, of 0s and 1s, which can code further pieces of information. It is usual to think of the oracle tape as coding a subset A of the natural numbers. Again it is inessential how this information is coded as long as it is done in such a way that the machine can decode from the tape the information it requires. The two tapes are placed in parallel, one above the other, and the R/W head is able to read simultaneously from both tapes at once, whilst only being allowed to write to the principal tape. The instruction set is expanded to allow the R/W head to query of the oracle tape what is written in the particular cell of the oracle tape. In this way the machine has access to yes/no answers to queries of the form *Is* $n \in A$? We denote the set of programs with this possibility as $\langle P_e^Z \mid e \in \mathbb{N} \rangle$ for any potential oracle $Z \subseteq \mathbb{N}$.

We have stated this much detail already because we are going to think of these machines as having a finite nature (notwithstanding the possibly infinitely long oracle tape sequence); we shall later be allowing them to run possibly into the infinite, and so we should be clear in what follows as to what is finite about our machines and what is not. Note that we have already assumed some other-worldly features, since we cannot literally build a machine with infinitely many parts; we may choose to get around this, by saying that the machine has an arbitrarily large number of cells enough for any computation in hand, or else we add on more cells to the paper tape, as and when we need them. Turing machines (TMs) are really useful for *thinking about* the nature of computation: the usual kinds of coding and programs one sees in text books rapidly involve astronomical amounts of paper tape and stages in time when performed on a TM, and are way beyond real world physical feasibility in any case. However this is not the point, of course.

As Turing explained there is a *universal* such machine or program $\mathscr{U}$ that can emulate all others. Let us recall how this is done.

When the machine runs, at any time a finite amount of information, a 'snapshot', tells us exactly the state the machine is in. This consists of a pair $\langle q, r \rangle$. The first term, q, tells us the line number of the program table at which the machine is (and this tells us the current state of the machine, what it will do depending on the symbol read etc., and the state it will go to, and where it will move after the instruction is performed). The second term, the number r, codes two things: r_0, the

number of the current cell the read/write head is hovering over; and r_1, a number coding the tape contents up to this point in time. (We assume the machine starts with a tape with cells C_i containing an initial sequence of 0s or 1s as input, and blanks thereafter running off infinitely far to the right say, and that the head writes only 0s and 1s; r_1 then must code up the current working tape's contents of a finite sequence of 0s and 1s. This also assumes that no blank cells intervene between a written 0 or 1 cell; with some care we can arrange our program to ensure this too.) A successful computation is then one that enters a *halting state*, and we deem the machine to have halted with whatever numerical binary output is on the working tape. (In other programs if we are only interested in a *yes/no* answer then we can ask that this be recorded as $1/0$ in C_0 say.)

The program $\mathscr{U}$ works by decoding the pair $\langle e, i\rangle$ where e is a code number of the program, and i is to be the binary input on the tape at the start (if any). The *oracle* as we have described it can be considered to be a bit-stream of 0s and 1s written to the oracle tape. Although this is not often stated, we do not wish to assume in advance a bound to the queries that a program on differing inputs can throw up, and it is considered quite usual to have an infinitely long such bit-stream. This again is an other-worldly feature.

In the computations that we are normally interested in, we consider only halting computations. Since this happens after a finite number of stages of time, only finitely many cells can have been written to, and only finitely many queries can have been made of the oracle. Thus, for this particular computation a purely finite machine, with finite-length tapes would have sufficed. However, for the next computation on this program, but with a different input, perhaps more, perhaps less tape and more or fewer queries would be needed. So we allow arbitrary lengths of tapes.

An infinite sequence is thought of as a *function* $f : \mathbb{N} \longrightarrow \{0,1\}$ with $f(k)$ the value at the kth place. If a program P_e on input k halts with the (binary) value m on the output tape, we denote this by $P_e(k)\downarrow m$. We may think of the program P_e as computing a function $f : \mathbb{N} \longrightarrow \mathbb{N}$ given by $f(k) = m$ iff $P_e(k)\downarrow m$, and such a function is deemed (TM-)*computable*. We allow for the possibility that the function may not be defined for every input k (this means that if it is not defined for k, but we insist on running the machine on input k, then the computation $P_e(k)$ will run for ever). Again, by means of coding pairs of integers by integers such a function can be identified with a subset of $\mathbb{N}$ and in turn any subset $X \subseteq \mathbb{N}$ can be identified with its characteristic function $c_X : \mathbb{N} \longrightarrow 2 = \{0,1\}$ via $k \in X \leftrightarrow c_X(k) = 1$. An infinite strings of 0s and 1s then can be thought of as such a characteristic function; the set of all such strings is denoted $2^{\mathbb{N}}$.

The rightly famous *halting problem* (in one version) asks: if $\mathscr{U}$ is given $\langle e, 0\rangle$ will it halt? (We abbreviate this as $\mathscr{U}(\langle e, 0\rangle)\downarrow?$) Turing showed by a so-called diag-

onalisation argument, that there is no machine, i.e. program, which will itself give $0/1$ output answering this question on input e for every e. We can see in a naive fashion the problem: we may run $\mathscr{U}(\langle e,0\rangle)$ simulating $P_e(0)$ and, after finitely many stages, if the latter simulation halts, we can arrange for a 1 to be in C_0 and then the overall computation $\mathscr{U}(\langle e,0\rangle)$ halts. This process is finite, perhaps $P_e(0)$ takes N steps itself, and indeed we can code up the whole successful *course of computation* $\langle q_0,r_0\rangle,\langle q_1,r_1\rangle,\ldots,\langle q_N,r_N\rangle$ by a single integer $y=y(e)$ (where $\langle q_i,r_i\rangle$ is the 'snapshot' at the ith stage). This y then contains all the information needed to reconstruct the whole sequence of computations. However if $P_e(0)\uparrow$ (meaning $P_e(0)$ never halts) then clearly there is no such y. Since $\mathscr{U}$ can do these simulations in a uniform manner (meaning the way it does this is not dependent on any special features of the program coded by e), we could think of $\mathscr{U}(\langle e,0\rangle)$ as calculating for us such a $y=y(e)$, but again *only if it exists*. Otherwise $\mathscr{U}(\langle e,0\rangle)$ will run forever. Such ideas are behind Kleene's T-theorem which formalised these ideas of Turing in a single summary statement (here somewhat simplified just for one-place functions).

Theorem 14.1.1 (Kleene's T-theorem) (i) *There is a computable predicate T of three variables such that for any program index e:*

$P_e(k)\downarrow n \leftrightarrow$ *there is a minimal* $y\in\mathbb{N}$ *such that* $T(e,k,y)$*; and moreover such a y codes the course of computation witnessing that* $P_e(k)\downarrow n$.

(ii) *Moreover, given any e, there is an algorithm for determining an e' with the property that for any $k\in\mathbb{N}$ we have*

$$P_e(k)\downarrow n \leftrightarrow P_{e'}(k)\downarrow y \text{ with } T(e,k,y).$$

Kleene used (the full version of) this as a basis for establishing a series of fundamental results on computable functions: the *enumeration theorem* the *S-n-m* theorem, and the *recursion theorems*.

The computation of $\mathscr{U}(\langle e,0\rangle)$ is such that we can regard it as *searching for* a y that does the job. We could arrange for $\mathscr{U}(\langle e,0\rangle)$ to simply inspect each $y=0,1,2,\ldots$ in turn until it finds the right one (running off to do tests each time to see if y indeed does code the right snapshots). The program $\mathscr{U}$ is thus performing an *existential search*: it searches for an existent y that does the job. Such existential searches thus typify the limits of the capability of the Turing machine model: we can always get a program/machine to look for a number which has some simple properties, such as being the code of a course of computation, and if successful output it. What has been slipped in here is the phrase 'simple properties'. This has to be clarified or else we may end up classifying the halting property of $P_e(0)$ as 'simple'. What is required of 'simple' is that it be a property Φ that is *decidable* by

Figure 14.1 S.C. Kleene 1909–1994

a Turing machine in *finite time*, i.e. a routine may be run to check $\Phi(n)$ which will assuredly halt with a yes/no answer.

Examples of simple: a prime number; a power of 3; a code of a program state; a code of a program; a course of computation of some $P_e(k)$; being a solution to a priorly given polynomial.

Examples of non-simple: a number e such that $P_e(0)\downarrow$; a code of a number pair $\langle e,n\rangle$ such that $P_e(n)\downarrow$; a number e such that $P_e(k)\downarrow$ for infinitely many k ...

The point to note about simple properties is that they are all *local*: a computation involves only numbers *local or near* to the given number (typically often only smaller numbers). The process may involve a *search* through some numbers, but it will be a *bounded search*: we shall know in advance how far to look. In terms of logical expressions a 'simple property' turns out to be precisely those that can be written out by a number-theoretic formula which involves only 'bounded quantifiers'. (These are called by logicians 'Δ_0 properties'.) The non-simple examples given involve *unbounded searching* or equivalently *unbounded quantifiers*: to justify $P_e(0)\downarrow$ we must be prepared for a search through all possible course of computation numbers y. We ask for something to exist: it is an unbounded existential quantifier '$\exists y[\cdots]$' and is thus an *unbounded existential search*. To justify the nega-

tion, i.e. $P_e(0)\uparrow$, we have to search through all numbers to check that something does not happen. This is an unbounded universal statement:

$$\text{`}\forall y \textit{ not } [\cdots]\text{'}.$$

Logicians classify these statements as '$\exists$' (also called 'Σ_1') and '$\forall$' (also called 'Π_1') respectively. The last statement that "there exist infinitely many k such that ..." is more complex still: it is '$\forall\exists$' or Π_2: $\forall n\exists k[n < k \& \cdots]$. In general a formula of arithmetic may have more quantifiers still, and they are classified as Σ_n which means n alternations of quantifiers starting with a $\exists$, and then a matrix $[\cdots]$ which contains no unbounded quantifiers. Similarly we define 'Π_n', which are then the negations of Σ_n. Sets of numbers defined by such formulae are, in the language of arithmetic, appropriately called 'arithmetic'.

Another point to note is that if a property Φ has both an existential form, $\exists n\Psi_0(n)$, and a universal form, $\forall m\Psi_1(m)$, then it is decidable: we may start searching simultaneously for an n so that $\Psi_0(n)$ and for an m so that $\neg\Psi_1(m)$, now two existential searches: if Φ holds the former will be successful after a finite stage; if $\neg\Phi$ then the latter, again after a finite stage.

Existential properties are also called semi-decidable: what we have just argued is that if both Φ and $\neg\Phi$ are semi-decidable, then in fact Φ (and so also $\neg\Phi$) is decidable. Thus we can effectively decide both by the use of TMs.

14.2 Limit decidable

We have seen that an existential search can be effected on a TM. We may thus have a function $f(k) = n$ which is definable by a Σ_1-property $\Phi(k,n)$, computed by a machine. To compute $f(k)$ the machine tests increasing values of m to see if $\Phi(k,m)$ and outputs n when (and if) it finds it. If $\mathrm{dom}(f) = \mathbb{N}$ then this will always halt, but if $f(k)$ is undefined then on input k this search continues forever. The following model is almost the next best thing.

Suppose we have a property $\Phi(n)$ of numbers. We say that $\Phi(n)$ is *decidable-in-the-limit* if there is a computable function $f : \mathbb{N}\times\mathbb{N} \longrightarrow \{0,1\}$ such that:

$$\begin{array}{rcl} \Phi(n) & \longleftrightarrow & \mathrm{Lim}_{y\to\infty} f(n,y) = 1, \\ \neg\Phi(n) & \longleftrightarrow & \mathrm{Lim}_{y\to\infty} f(n,y) = 0. \end{array} \qquad (*)$$

The 'Limit' notation above means that if, e.g., $\Phi(n)$ holds then the value of $f(n,y)$ as y increases will at some point eventually be 1 *and will stay at* 1 for all larger values of y. The form of $(*)$ says that *either* the value will be 1 from some point on, *or* it will be 0. If only we could stand back at the 'end of time' after all the finite steps, and look back and see to which value $f(n)$ has settled down, then we should

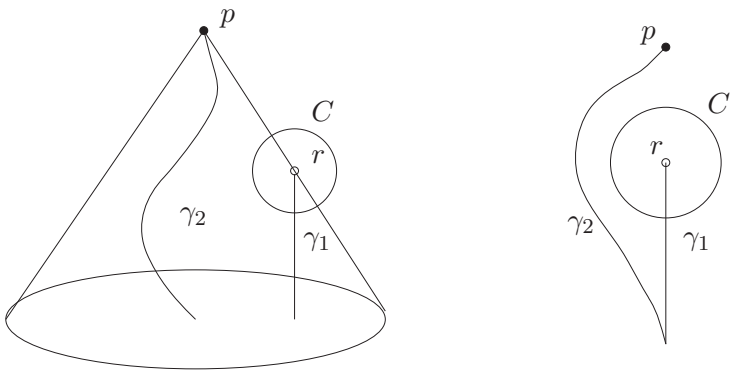

Figure 14.2 MH spacetime.

know if $\Phi(n)$ holds! Eventually because of the properties of f given above, 'in the limit' we should have the right answer on the tape.

We see the essential feature that both the expression formulating the property, and its negation, are Σ_2: they are of the form $\exists y_0 \forall y > y_0 \cdots$ meaning that 'something holds from some point onwards'. Moreover it can be shown that any property with this essential feature can in turn be expressed also in the form $(*)$ above. However, we also have to have the capacity to go to the end of the stages $0, 1, 2, \ldots$ in time to check our output tape. We cannot know at any finite stage of time whether we have arrived at the final answer. Nor, it can be shown, is there any way to define a computable function that given n will tell us how long to wait! We truly do have to wait through all time to know that we have the right answer on the tape. How can we do this?

14.3 Malament–Hogarth spacetimes

Various authors (Etesi-Neméti, Hogarth, Pitowski) have pointed out at various times that Einstein's equations for general relativity (GR) allow for spacetimes where in the causal past of one observer, call her $\mathscr{O}_p$, there may lie a path of infinite proper time length for a second observer $\mathscr{O}_r$. Such spacetimes have been dubbed 'Malament–Hogarth spacetimes' (MH). Such a situation is pictured in Figure 14.2 on the left.

This is non-contradictory since $\mathscr{O}_r$ is heading towards a singularity at r along the path γ_1. One can imagine such a situation as arising in the following way. One takes a regular flat Mostowski spacetime, where outside a compact region C the curvature is flat. Then one smoothly allows the curvature to go off towards infinity as r is approached. The point r is then removed from the manifold; but the whole

path γ_1 remains in the causal past of $\mathscr{O}_p$ although now, by the calculations of GR it has infinite proper time length. Ideally then, one can send a TM along the path γ_1 checking say some universal $\forall$ property, such as Goldbach's conjecture (which says that for all even numbers greater than 2, they are the sum of two primes) or that one's favourite mathematical theory is *consistent* – for example PA, the axioms due to Peano for arithmetic; or ZFC, the axioms of Zermelo–Fraenkel set theory (the *consistency* statement is again a 'for all' one: it says for all finite sequences of formulae making up a proper proof from the axioms, if the final concluding sentence is s say, then s is not a contradiction, such as '$0 = 1$'). If the property fails, this is because some counterexample to Goldbach's conjecture has been found: namely an even number that fails to be a sum of two primes, or, a proof of '$0 = 1$' from the PA axioms. The TM then sends a signal to $\mathscr{O}_p$. If a signal is received, then $\mathscr{O}_p$ knows that the Goldbach conjecture is false, or that PA is inconsistent. Theoretically (and summarised diagrammatically on the right of Figure 14.2) then, the observer $\mathscr{O}_p$ (travelling along γ_2) could herself set the TM running along γ_1, go out to lunch, and then if no signal is received at point p when she returns, say an hour later, then she knows that the universal statement is true since no counterexample has been reported as found.

Now of course in many respects this situation is wonderfully fantastic: the TM that is to be sent down the path γ_1 now absolutely has to have an arbitrarily long tape, since it is going to have to check Goldbach's conjecture for arbitrarily large even numbers, and representing these on the tape alone requires arbitrarily long sequences of 1s even before any calculation takes place. Since we cannot literally get hold of an infinite tape, we have to tell a story about the machine, or the generations of workers we send with the machine, extending the tape as it, or they, go along etc. Others have pointed out physical difficulties of another kind, concerning the arbitrarily large blue-shifts in the signals that the human observer receives (as for her the time of the TM becomes arbitrarily speeded up, and hence the wavelengths of the signal she receives could be arbitrarily small), or the increasing energies needed for the signals to be distinguished from background radiation.

Nevertheless certian spacetimes that have been studied earlier and quite independently of this issue have the MH property: Gödel's famous model where there are time loops, anti-de Sitter spacetime, and Reissner–Nordström spacetimes. Etesi and Neméti have been proponents of Kerr's solution to the GR equations in the form of a rotating black hole – for which there is observational evidence (see Figure 14.3). In such a model the singularity is not in the form of a point or centre, but is a ring around the black hole. In their arrangement the observer $\mathscr{O}_q$ descends into the ring, whilst the TM orbits the black hole! They have arguments to get around the blue-shift problem mentioned above (for example by the computer calculating what the blue-shift might be from the human observer's end and sending a rocket

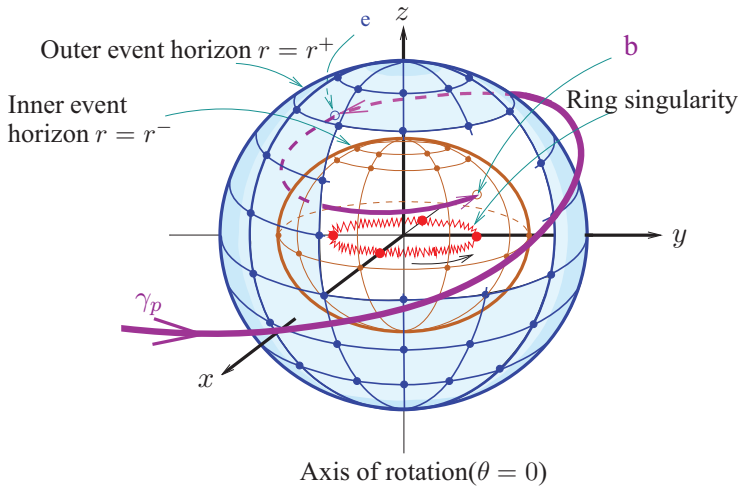

Figure 14.3 A slowly rotating (Kerr) black hole has two event horizons and a ring-shaped singularity. The ring singularity is inside the inner horizon $r = r^-$ in the 'equatorial' plane of axes x, y. With grateful acknowledgements to G. David and I. Neméti 'Relativistic computers and the Turing barrier', *App. Math. and Comp.*, **178**, 2006.

in the reverse direction to counter its effects!) and for some of the following difficulties. The spacetimes listed all have varying causal properties but, in common, as Hogarth showed, they all violate a property called 'global hyperbolicity', which in essence says that the future can be determined by a full account of the initial conditions over a complete hypersurface – a so-called 'Cauchy surface'. Thus an assumption of global hyperbolicity rules out any spacetime as being MH.

Besides this, other hypotheses, variants on Penrose's *cosmic censorship hypothesis* which rules out 'naked' singularities (i.e. singularities that are not 'hidden' by an event horizon such as the points r on the curves γ_1), also fail. These are reasonable hypotheses about causation in spacetimes, in general, but which have not been deduced from the GR equations. Nevertheless they encapsulate properties that one might want any GR solution to have, and although the matter is far from settled, many physicists endorse one or other of them.

However, let us cavalierly brush all these grumbles aside and see what *logically* comes out.

14.4 Infinite ordinals: beyond arithmetical

To discuss sets of numbers that are more complex than arithmetic (and later, computations that are longer than those given by natural number stages alone), we must invoke Cantor's theory of ordinal numbers. We shall not need this whole theory, but

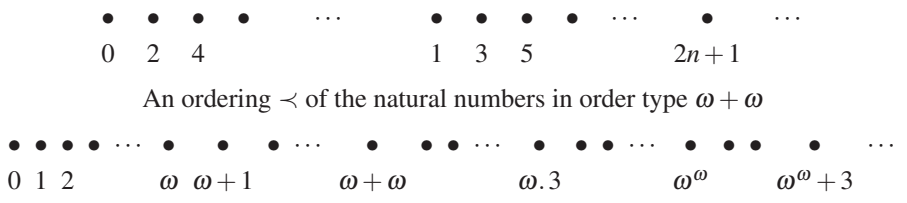

Figure 14.4 Some ordinals.

we do need the notion of a *countable wellordering*. (A set is called *countable* if it can be put into one-to-one correspondence either with a natural number $k \in \mathbb{N}$ or with all of $\mathbb{N}$.) Some infinite wellorderings are pictured in Figure 14.4.

Figure 14.4 gives pictorial representations of these ordinals, in the same way that four dots in a row can represent an ordering of four. Cantor showed how we can have arithmetical operations on ordinals that extend the usual addition, multiplication, etc. that we have on the finite numbers. That won't concern us much here, but two things will count: (a) the pictures of these transfinite ordinals and (b) the fact that countable ordinals can be represented by characteristic functions of subsets of $\mathbb{N}$.

To illustrate (b), consider $\omega + \omega$. (In set theory the class of finite numbers is given different names depending on the role it is playing at any moment: by '$\mathbb{N}$' we mean the *set* of all the usual finite ordinal numbers starting with 0; on the other hand 'ω' whilst being strictly the same set, emphasises the *ordering* role of the set of numbers. We think of an ordinal number as the set of its predecessors. This makes $3 = \{0,1,2\}$; $n = \{0,1,\ldots,n-1\}$ and $\omega = \{0,1,\ldots\}$, then $\omega + 1 = \{0,1,\ldots\,\omega\}$. The ordering we are considering is then that on the set $\omega + \omega = \{0,1,\ldots,\omega,\omega + 1,\ldots\}$.)

We can represent pictorially that ordering, as at the top of Figure 14.4, by putting all the even numbers before the odd numbers to get something of 'double $\mathbb{N}$ length'. However we need a mathematical definition of this picture, something that a computer can work with, so we define mathematically a new ordering $\prec$ on $\mathbb{N}$ by $i \prec j \iff i, j$ *are either both even or both odd and* $i < j$ *or* i *is even and* j *odd.* Then if we draw the picture of $\prec$ it looks like the first example above. Now consider the function $\pi(n,m) = 2^n \cdot (2m+1) - 1$. This is a one-to-one mapping, a injection, from pairs from $\mathbb{N}$ into $\mathbb{N}$. We can use this to code orderings or indeed any subsets of $\mathbb{N} \times \mathbb{N}$: we let $E_\prec \subseteq \mathbb{N}$ be the set of k of the form $\pi(i,j)$ where $i \prec j$. This $E_\prec$ codes exactly the ordering $\prec$. Conversely given any $F \subseteq \mathbb{N}$ we can look at $\prec_F =_{\text{df}} \pi^{-1}\text{“}F = \{(i,j) | \pi(i,j) \in F\}$. Then it may or may not be the case that $\prec_F$ is an ordering at all. But to be a *linear ordering* it is an $\forall\exists$ question about F:

First: F is a *transitive* ordering iff $\forall p,q,r(p \prec_F q \,\&\, q \prec_F r \Rightarrow p \prec_F r)$ iff $\forall k_0 \in F \forall k_1 \in F \forall p,q,r \exists k_2 [\pi(p,q) = k_0 \wedge \pi(q,r) = k_1 \longrightarrow \pi(p,r) = k_2 \in F]$.

Second: two more clauses that express that $\prec$ is *connected* and a *strict* ordering ($n \not\prec n$) have to be added to this to ensure that F codes a *linear* ordering of a subset of $\mathbb{N}$. This makes sure that all the elements in the picture are lined up. It was Cantor who suggested that we extend the quintessential feature of the natural number system: that it is *well-ordered*; this means for a linear ordering $\prec_F$ that if $X \subseteq \mathrm{Field}(F)$ and X is non-empty, then there is an $\prec_F$-*least* element k_0 in X. Thus: there is no $p \in \mathrm{Field}(F)$ with $p \prec k_0$. All the pictures above have this property. But the property is a special one: many natural orderings are ill-ordered: the negative integers with their usual ordering are ill-ordered, because they themselves have no least element; the usual ordering of the rational numbers in $[0,1]$ is ill-founded, as the set $\{\frac{1}{n+1} \mid n \in \mathbb{N}\}$ has no least element. Notice also that being well-ordered has to be expressed by a quantification over all subsets X of $\mathbb{N}$ since it cannot be expressed by number quantifiers $\forall n, \exists m$, etc. In fact many sets F that code well-orderings are rather simple; the set coding the ordinal $\omega + \omega$ of evens followed by odds is actually computable: we can write a program P_e such that on input k it applies π^{-1} to k, so yielding a pair (n,m), and checks whether $n \prec m$ in that ordering. However, many code-sets of wellorderings are highly complex; indeed the set $\{e \mid P_e$ computes a wellordering$\}$ in the above manner is itself a highly non-computable set of program indices.

As well as defining wellorderings we can drop the requirement that the orderings be linear. We allow pictures of *trees* (Figure 14.5) where we draw the tree *downwards* from its root (thus we have $2 \prec 5$ in the tree). We say that $F \subseteq \mathbb{N}$ codes a tree, if we drop the connectedness requirement from being a linear ordering. It is then a *well-founded tree* if it is both a tree and, now, for any $X \subseteq \mathrm{Field}(F)$, we still require if it is non-empty that there is some element $k_0 \in X$ that is $\prec_F$-*minimal*. (Now of course there may be many such $\prec_F$-minimal elements of X because the tree splits in many ways. Indeed any node may have *infinitely many* elements immediately below it. Notice that any such countable well-founded tree can be considered (and is often called) a *finite path tree* because any path starting anywhere in the tree and proceeding only downwards is finite in length. (It cannot be infinite because that path would constitute a subset $X \subseteq \mathrm{Field}(F)$ without any minimal element.)

Now notice that no Turing machine, given say X as an oracle, can answer the question of whether X codes a linear ordering (we have seen that already requires a two quantifier condition to be answered). Nor can we say given a program number e, whether we can run another program to decide whether P_e computes the characteristic function of a linear order: this would require a machine to answer infinitely many questions about P_e's behaviour which no machine can do. Because this is a Π_2 question, even if we allow computation-in-the-limit it is still impossible, as this latter kind of computation only allows us to answer questions which can be simultaneously expressed as Σ_2 *and* as Π_2; questions that are Π_2 expressible it cannot

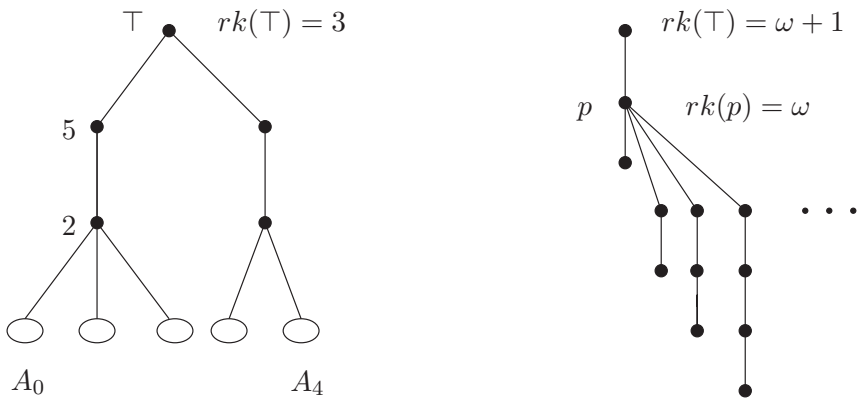

Figure 14.5 Finite-path trees.

handle. We have to go to a stage at which we can look back at the previous sequence of all stages and come to a judgement: we need to be standing at time $\omega+1$, at some node or point beyond all times $0, 1, 2, \ldots, k, \ldots$ How can we do this?

Hogarth (2004) suggested putting diagrammatic elements of his spacetime continua (the right of Figure 14.4) together to be able to answer any arithmetic question. If one were to draw a schematic diagram of such a question, such a tree would be of the kind on the left of Figure 14.5.

The subquestions that need to be answered by such an array of machines, in an array of spacetime regions, can be represented as a special kind of finite path tree, where we allow infinite splitting below nodes, but *every path must have a fixed bound*, k say, on its length, corresponding to the number of quantifiers in the question.

We say that a finite-path tree T has a *(finite) rank k* if, assigning the leaves rank 0 and for any node $p \in T$, we define $\mathrm{rk}(p) =$ least number $k >$ all $\mathrm{rk}(q)$ for $q \prec_T p$. If $\top$ is the topmost node, then we set $\mathrm{rk}(T) = \mathrm{rk}(\top)$. Then $\mathrm{rk}(T)$ is always finite in the case of trees associated to arithmetic-deciding bits of spacetime. But what if $\mathrm{rk}(\top) = \omega$? Or something larger? Initially the reader may be surprised that a tree can have infinite rank but only finite-length paths. However, see the right-hand diagram of Figure 14.5. Such trees can be used to build sets B of complexity far beyond the arithmetical and hence properties ('being a member of B') of numbers that are more complex than arithmetical. The first such sets B are *relatively* simple because we can describe a *computable protocol* for their construction: we take a *computable* finite path tree, and at the minimal bottom-most nodes we attach some simple sets A_i $(i \in \mathbb{N})$, say computable or arithmetically defined ones. Then we inductively proceed up the tree, attaching to every node $p \in T$ a set A_p. If p is

terminal we have already done this; if the node immediately above p, p' say, only has p below it, we set $A_{p'}$ to be the complement of A_p; otherwise $A_{p'}$ is the union of all the A_q, with q immediately below p'. In this way a set gets attached to the topmost node $\top$, and we set $B = A_\top$.

The countable collection of sets defined in this way are called the *hyperarithmetic sets*. Note two things: first, that they include all the arithmetic sets (we indicated that these are obtained by using just trees of finitely bounded path length) and thus are far from computable sets. Second, although we emphasised that we were taking computable trees, with computable assignments of arithmetic sets to the leaves thereby obtaining one of a potentially countable set of possible computable descriptions of such B, mathematically we could have dropped these computability requirements and enlarged our class by considering *all* possible finite path trees and *all* possible assignments of arithmetic sets to leaves. One can argue that as opposed to the hyperarithmetic sets, which form a countable collection, there are then uncountably many such sets so created – this much wider class is called the class of *Borel* sets, and they are arranged in a hierarchy of complexity depending on the minimal ordinal rank of a constructing tree.

14.5 Returning to MH spacetimes

Now we can see that we could extend the argument of Hogarth deciding membership of arithmetic sets, to membership of hyperarithmetic sets, or even potentially of any particular Borel set B given the right spacetime manifold. We need a region of spacetime with a tree structure of SAD components reflecting the tree structure of B's construction. We use the Hogarth-type 'arithmetic trees' as starting component regions for the leaves at the bottom; these components must then be considered as within a hypothetical region that itself reflects the finite path tree constructing B with those arithmetic sets at the leaf positions.

Indeed, with care, and if we were able, we could enumerate all possible computable protocols $e_0, e_1, \ldots$ enumerating the hyperarithmetic sets, and one could hypothetically find a region of spacetime that would accommodate all such trees simultaneously. One could have an overall Command TM, that when submitted a pair (e,n) representing the query "*Is* $n \in B_e$?" where B_e is the eth hyperarithmetic set, would send n to the machine at the entry point to the eth region. In short, we should have a *hyperarithmetically deciding region*, HAD. The reader may be forgiven at this point for thinking that we have departed from reality by now (if not some while ago). For example, the enumeration of the computable protocols cannot be a computable process for one thing (we could get a contradiction from assuming it were, by a diagonal argument). Nor is it even arithmetic. We also have the *recognition problem* of even seeing when or where there is a SAD region in

our spacetime, let alone these exotic varieties. However, the point is to push the envelope of what is *logically* conceivable with MH spacetimes, granted the physical assumption that a single SAD is consistently available. We are not discussing the physics or (even potentially) the physical realisability.

However we can establish a purely logico-mathematical limit to even to these very theoretical, very speculative, theoretical computations in MH spacetimes. Given an MH spacetime (perhaps our universe after all is such a one), could there be sufficiently many regions that any Borel question could be answered? The answer (Welch, 2008) is no: we do hit a limit, a *universal constant* of the spacetime manifold $\mathscr{M}$:

Theorem 14.5.1 *Given an* MH *spacetime manifold* $\mathscr{M}$ *there is a countable ordinal* $w(\mathscr{M})$ *such that no Borel question of rank* $w(\mathscr{M})$ *can be 'answered' by a region contained in* $\mathscr{M}$.

14.6 The $\aleph_0$-mind

What the above forcefully illustrates is that we have to have some way of coping with infinitistic arguments concerning infinite sets and processes: we can only think about and work with a hyperarithmetic set if we can in some way comprehend the totality of its description (which, although given by a number index e say, has to be unpacked into the characteristic function of an infinite finite path tree). If only our minds were capable of this! The phrase '*the* $\aleph_0$*-mind*' has been used to describe a mind capable of assimilating and manipulating countable amounts of numerical information in one step. One way of thinking further about this is to imagine a Turing machine where we may survey the whole infinite tape at one glance, and ask of an oracle Z "Is the *whole of the work-tape's* contents an element of Z?" Thus Z is now to be thought of as not just a set of integers but a set of infinite strings of 0s and 1s. Depending on an answer, or our program, we then in a single movement effect some computable process on the whole tape (such as flipping every other bit, or moving and storing the string somewhere). Now the notion of computation is radically changed: we are computing not on natural numbers but on infinite (but countable) objects, the infinite strings on the tape.

In a series of papers in the late 1950s and early 1960s, S.C. Kleene developed an equational calculus for such processes and even higher generalisations. Kleene was a pioneering researcher in Princeton in the early days of computability theory working with Church and Gödel in the 1930s.[1] He developed an equational calculus for presenting the 'generalised recursive functions' essentially due in one form to Gödel. This class of functions Turing showed to be precisely the class

[1] This early history is well surveyed in detail in Soare (2009).

computed by TMs. The calculus is a sort of axiomatic listing of clauses as to how such functions from $\mathbb{N}^k$ to $\mathbb{N}^n$ (for any k,n, say) could be built up in an inductive fashion. Numbers are thought of as being of *type 0*, namely at the bottom of a hierarchy. A function $f : \mathbb{N} \to 2$ is a *type 1* object. A *type 2* object would be a function $F : \mathbb{N}^{(2^{\mathbb{N}})} \to 2$, and so on. Kleene axiomatised *higher type recursion theory* by extending his original axioms for the generalised recursive, or Turing computable, functions in a natural way to axioms for 'type-*n* recursion'. This subject became a significant area of research in the 1960s and early 1970s, with people trying to explore his definitions, ironing out the quirks or looking at possible alternatives in the axioms. This bottom-up approach rapidly becomes rather complicated (as are his original equational sets). In the late 1960s, work of Moschovakis (later joined by Kechris, Harrington, and Normann amongst others) began to build on the work of Kleene, Spector and Gandy, to give a definition of a generalised recursion theory which emphasised *logical definability* rather than *mechanical* definability (or perhaps better put, *computably derived* notion of definability of Kleene's equations). This was a very flexible approach and rapidly connections were made with the theory of inductive definitions and descriptive set theory (the latter seeks to prove results concerning, not all sets of numbers, but sets as classified by their descriptions, i.e. logical definitions; this has become a central area of modern set theory.) Indeed Moschovakis was able to define notions of 'inductive' and 'hyperarithmetic' not just for the natural number system but for *any* structure satisfying a modicum of coding possibilities. (Such structures he called *acceptable*. For example such a structure should have a way of pairing off elements of the domain M: there should be in the structure $\mathscr{M}$ a function $\pi : \mathscr{M} \times \mathscr{M} \to \mathscr{M}$.) Whilst such a view can be said to have given what must be considered the definitive theory of inductive definability and the associated recursion theories, these leave the machine or mechanical approach or intuition far behind.

Let us return to the $\aleph_0$-mind. Kleene showed that his equational calculus allowed a notion of what is now called 'Kleene computation', involving, as we have said, sets of numbers (or equivalently the infinite strings of 0s and 1s that make up their characteristic functions). Thus a type-1 recursion can be thought of as computed by the thought-experiment machine where, as we have suggested, we manipulate the whole string and ask the oracle questions about the whole string in single steps. He showed that in this case, what corresponded to the computable or decidable sets of integers, would now be the hyperarithmetic sets of integers and subsets in $2^{\mathbb{N}}$; what corresponded to the computably enumerable sets of integers would be the subsets of $2^{\mathbb{N}}$ of a complexity no more than that of the wellorderings of $\mathbb{N}$. (In descriptive terms of analysis, Kleene's 'computable enumerable' sets are Π^1_1 (also called 'co-analytic') sets, and all such sets can be computed (in the usual sense) from WO, the set of such codes. (Actually Kleene allowed elements of $2^{\mathbb{N}}$

as constants or parameters in these computations, thus in effect allowing not just computable finite path trees into one of his computations, but any such; the upshot is that for Kleene the ground computable sets were the Borel sets, not just the hyperarithmetic ones.) An essential feature of Kleene computations is that a particular computation could require an infinite string of subcomputations $s_0, s_1, \ldots$ to be completed to deliver up some information for the main computation to continue. (We see this for example in the description of the hyperarithmetic sets above, when the tree is infinitely branching: to know whether n gets into a set $A_{p'}$ at such a node p' we must look at all the possible sets A_q at nodes immediately below $p' \ldots$). A successful Kleene computation is then best thought of as having itself a *well-founded computation tree* structure; diagrammatically this would show the structure of all these subcomputations. We say well-founded, because we do not want an infinite string of sub-routine calls to endlessly go down an infinite path, as such a computation would never terminate.

The $\aleph_0$-mind that is trying to keep an eye on all these subcomputations, indeed must do so over the whole tree at once. At a particular node the process at that node must wait for information to come back up from lower nodes; it would be possible to 'linearise' this whole process, fulfilling in a linear order such computation calls. But, as indicated, some such calls would have to wait for infinitely many steps below the node to be completed. Such a linear order would have to be a wellordering of length, well what? ... some countable ordinal, as the overall tree is a countable finite-path tree. So maybe we should in any case analyse linear, but transfinite, computations? To do this we have to cross a different kind of event horizon: that of ω, the first infinite ordinal.

14.7 Infinite-time Turing machines

Infinite-time Turing machines (ITTMs) are just such an attractive model of computation, invented by J. Hamkins and J. Kidder in Berkeley in the 1990s (Hamkins and Lewis, 2000). Their motivation was really to find a definition of how one could allow a Turing machine to run transfinitely. This would mean a well-ordered sequence of computations with successor computation steps given by a standard Turing program. The machine was to be equipped otherwise with the usual Turing machine capabilities. Thus the *length* of a terminating computation would be some countable ordinal, now possibly infinite. Really all that has to be added is a description of what happens at limit ordinal times. Thus, for example, we may run the machine through finite steps $0, 1, 2, \ldots$ If this is a halting computation in the usual Turing sense, then for some $k \in \mathbb{N}$ the machine stops. All well and good. But if not? It will run throughout all the natural number stages. What happens at stage ω?

We avoid all considerations of 'Thomson lamp'-like difficulties (a body of literature in the philosophy of science that asks what happens after a light has been switched on and off infinitely often) by simply declaring by mathematical fiat that a cell C_k has the value at time ω say (which we shall denote $C_k(\omega)$), given in the following way: if there was a stage $n < \omega$ such that at all later stages m, with $n < m < \omega$, the value $C_k(m)$ is unchanged, then we set that value to be the value at time ω, $C_k(\omega)$. Otherwise we reset it to zero: $C_k(\omega) := 0$. An equivalent way to put this is to say the value at stage ω is 0, unless from some time before ω it was 1 and that 1 remained in the cell thereafter. This specifies what the cell values $C_k(\omega)$ are for all k. We have to specify where the R/W head got to! The first case to consider is that as $k \to \omega$ there was a particular cell C_n, say, that it returned to visit for unboundedly many stages $k < \omega$. In that case, for the least such n, call it n_0, we'll put the R/W head over this C_{n_0} at time ω. In the other case, the head has crept off down the tape to infinity: the 'liminf' of the cell numbers the head visits is not finite but is ω itself: for every cell C_n there is some time $t(n)$ such that for all later times t with $t(n) < t < \omega$ the head is reading a cell beyond C_n on the tape; i.e., it never revisits a cell C_m for any $m \leq n$ for $t > t(n)$. In that case we replace the head back on C_0 – the tape start at time ω, Finally we have to specify the next instruction state on the machine's program. This program is entirely a standard TM program, so it is a finite list $I_0, \ldots, I_N$ say. Hence if at time t we denote the instruction about to be performed as $I_{q(t)}$, then there must be a first (in our list) instruction state I_q on the list that was performed unboundedly often below ω: in other words, we have that this q is the liminf of the $q(t)$ for times $t < \omega$.

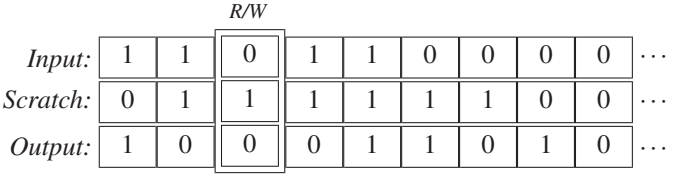

Figure 14.6 A three-tape infinite time Turing machine.

One attribute of these definitions is that they have the effect that the head (in the first case above) is thus placed, and the 'next instruction' is thus declared to be so that the machine is at the beginning of the outermost loop of any subroutine in the program that was called unboundedly often in time below ω. (Actually this is somewhat unimportant in what follows; indeed Hamkins and Kidder did not use this rule: they declared that the head would always return at limit stages to C_0 and the machine would enter a 'special limit state' added to the list of normal

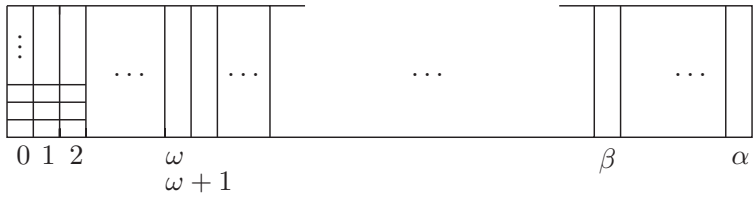

Figure 14.7 A course of computation of length α.

Turing states. They also used 'limsups' rather than 'liminf' considerations, but it is easy to argue that all these variations are minor and do not alter the classes of functions or sets computable by such machines.) What we have described at stage ω is now declared to hold true similarly for any limit ordinal $\lambda = \omega + \omega, \omega^2, \omega^\omega \ldots$ by replacing ω by λ in the above. The rules stay exactly the same.

Lastly, for conceptual ease, it is useful to think of the machine as having three tapes: for input, for scratch work, and for output, placed in parallel one above the other. The head reads simultaneously one cell from each tape but can only write to the scratch and output tapes. This again (with a minor caveat) has no difference in computational power from a single-tape machine.

Now we are in business! A number of things immediately occur to us: we have the possibility that a computation may halt not just at a finite time $t < \omega$ but at some transfinite ordinal time $\omega \leq t$ (we shall see an example of this below). However, some computations still will never halt. As we have set it up, such computations will continue throughout the eternity of all the ordinal numbers ON. However, it is intuitive to see (and can be proven) that any non-halting computation will essentially start looping at some *countable* ordinal. Let us say a 'snapshot' of the machine at a time t is a complete list of all the cell contents $\langle C_k(t) \mid k \in \mathbb{N} \rangle$, together with $q(t)$ and $r(t)$. In order for the machine to be permanently looping there must be a sequence of countable ordinals $\alpha_0 < \alpha_1 < \cdots < \alpha_n \cdots < \alpha_\omega$ where α_ω is the 'limit' of the α_n (thus any ordinal $\beta < \alpha_\omega$ is $< \alpha_n$ for some n), and where the snapshots at the times $t = \alpha_n$ are identical for all $n \leq \omega$. At this stage α_ω we know that thereafter the computation is doomed to cycle endlessly through the same snapshot configurations. (Figure 14.7 illustrates a course of computation of length α. One should think of each vertical section as a picture of the infinite three-tape sequence as time increases to the right.)

The other immediate possibility is that we can put an infinite string of 0s and 1s on the input tape at the start, and the machine has enough time to read the whole input tape and act on it. Likewise if the machine halts, there may be an infinite string of information on the output tape. In such cases we may think of the

machine computing a function $F : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$. We are thus definitely in the realm of computing on type-1 objects.

However, before we leap into this new world, there are plenty of questions to consider first about functions $f : \mathbb{N} \to \mathbb{N}$ (or perhaps $f : \mathbb{N} \to 2^{\mathbb{N}}$) where we may think of input n as being a string of $(n+1)$ 1s followed by 0s, or even just functions on 0 input? Actually, in terms of ordinal time, how long do such computations go on for. What kind of strings are on the output tape after the machine halts? More generally what kinds of reals can appear at *any* time? One can ask equivalents to the halting problem: what is $h =_{\mathrm{df}} \{e \in \mathbb{N} \mid P_e(0)\downarrow\}$? Suppose we write $P_e(x)\downarrow y$ for '*Programme indexed by e on input x halts with* $y \in 2^{\mathbb{N}}$ *on its output tape*'. We shall call such y '*writable-from-x*'. If $x \in \mathbb{N}$ we shall just say '*writable*'.

Question *What exactly are the writable sequences* $y \in 2^{\mathbb{N}}$?

Let us call a countable ordinal α *writable* if some writable $y \in 2^{\mathbb{N}}$ is a code for α.

Question *What exactly are the writable ordinals?*

This at first glance looks mysterious. Call an ordinal *clockable* if it is precisely the length of a halting computation $P_e(0)$ for some e.

Question *What are the clockable ordinals? Is every clockable ordinal writable?*

Question *What are the decidable, or the semi-decidable, predicates?*

These questions make sense for computations $P_e(x)$ where we consider priming the input tape with an input sequence x. We have then this higher type computational behaviour of the model to investigate. Again we may ask what are the decidable, or the semi-decidable predicates. More generally:

Question *What are the relations to the earlier Kleene recursion?*

Question *How do these ITTM-defined relations fit in with the later abstract recursion theories?*

Theorem 14.7.1 (ITTM T-theorem) (i) *There is an ITTM-computable predicate* T *of three variables such that for any program index e and for any* $x \in \mathbb{N} \cup 2^{\mathbb{N}}$*:*

$P_e(x)\downarrow z \leftrightarrow$ *there is a* $y \in 2^{\mathbb{N}}$ *such that* $T(e,x,y)$*, and moreover y codes the course of computation witnessing that* $P_e(x)\downarrow z$*.*

(ii) *Moreover given any e, there is an (ordinary) algorithm for determining an* e' *with the property that for any* $x \in \mathbb{N} \cup 2^{\mathbb{N}}$ *we have*

$P_e(x)\downarrow z \leftrightarrow P_{e'}(x)\downarrow y$ *with* $T(e,x,y)$.

Looking at the above we see the immediate difference between this and Kleene's original theorem: the presence of the y as now not an integer, but as an element of $2^{\mathbb{N}}$. However, this has to be so: even if $x \in \mathbb{N}$ or is zero, a course of computation is now in general a transfinite sequence of snapshots, each of which is essentially an element of $2^{\mathbb{N}}$. These must then all be amalgamated and coded together into a single $y \in 2^{\mathbb{N}}$. In part (ii), note that in order for there to be any chance of a program $P_{e'}(x)$ outputting such a y, we *must* have that y contains within it a code of the *length* of the halting computation, $P_e(0)$, say (if $x = 0$). For this to happen we need as a minimum that the ordinal length of any halting computation can itself be output as the result of another computation: that is, any 'clockable ordinal' must also be 'writable'. Fortunately this turns out to be the case and the theorem holds.

After the model's introduction, subsequent work gave us the answers to all these questions, and we now have a pretty exact idea of what the ITTM machine can do, at least in the case of integer input (see the survey article Welch (2009)).

It is at least pertinent to ask *What is this good for?* Clearly it has little to do with concrete everyday desk-top computation, so we are really working with a mathematical idealisation where we can compute *as if* we could transcend the finite integers. We started with a pure conceptualisation out of sheer curiosity, and ran with it to see where it goes. We do not expect it to directly affect everyday life. However, that much is true of most pure mathematical endeavours (at least initially). Here we do have a mathematical model that is an excellent test-bed, so to speak, for other models that compute discretely using fixed steps but also allow some infinitary action to take place. So, we can *simulate* the action of sending ordinary TMs through sequences of SAD or HAD regions of spacetime on an ITTM. In this respect, ITTMs are similar to TMs. Turing showed that many forms of recursion could be simulated on his machines; ITTMs too can mimick any form of (finite) computation on integers. Thus far ITTMs can simulate other forms of infinitary computational machine-models on integers which have been independently defined.

Just as the halting problem h_0 for TMs in a precise senses delineates the boundaries of what TMs can calculate, so we know now exactly what kind of set h is presented by the halting problem set for ITTMs. We thus have a similar boundary mapped. It turns out that this boundary lies far beyond that of Kleene recursion. The decidable predicates of reals in Kleene's sense are precisely the hyperarithmetic sets, but ITTMs can decide whether a $y \in 2^{\mathbb{N}}$ codes a wellordering, and this is beyond the capabilities of Kleene recursion. Although there is a sharply delineated boundary for the decidable predicates of ITTMs it is not the case that they could answer membership questions about *any* set built up from an *arbitrary* finite path tree description *unless* that description is coded into the input string. However if the protocol for building B is so coded on the input string, then for any num-

ber k the ITTM can answer if $k \in B$. It is just that there are only countably many programs whilst there are uncountably many such finite-path trees, and hence constructing protocols: the ITTM has only the ordinary Turing programs $\langle P_e \mid e \in \mathbb{N} \rangle$ to work with. Without the extra description as input, there are only just so many finite-path trees that can be defined by the ITTMs alone; but the rank of those path trees (or equivalently the wellorderings that are writable by an ITTM alone without input) give us the level at which the machines can calculate 'for themselves' so to speak. And that indicator is well beyond Kleene recursion and the hyperarithmetic sets.

14.8 Register machines and other generalisations

After the publication of the ITTM model, and with the realisation that this tied in with earlier work on generalised recursion theory, people started to look for ways to generalise it. Indeed, as many remarked at the time, the beautiful simplicity of the model, which extends the action of TMs into the transfinite, could easily have been invented decades earlier.

There are two possible approaches to generalisation: (i) try changing the properties of ITTMs or (ii) look at other models of computation from the standard everyday, finite arena, and let them loose into the transfinite realm.

We consider (i) first (although this is not in historical order). The over-arching feature of the ITTM is of course what it does at limit stages. One could (and should) reasonably argue that the whole of the capabilities of the machine have been hard-wired into the limit rules. Are there potential 'hypermachines' which exceed the ITTM's capabilities by strengthening the limit rules in some way? The rule as given above is essentially an $\exists\forall$- or Σ_2-rule. The value of a cell is 1 if and only if $\exists t < \lambda \forall s (t < s < \lambda \rightarrow C_k(s) = 1)$. Here the quantifiers on the right-hand side have this pattern. Could there be a Σ_3-rule or a $\exists\forall\exists$-rule? A Σ_n-rule for larger n? The presumption is that if such were sensibly defined, then a 'Σ_3-hyper-ITTM' could use the contents in previous stages in more subtle ways than simple liminf, in order to come up with a value at a limit stage λ.

It turns out that such Σ_n-hypermachines are definable for each $n \in \mathbb{N}$ and that they do indeed have the expected properties. In fact any real number whose existence can be proven outright in mathematical analysis can be 'computably written' in the sense above by some such machine in some Σ_n-class, with some such program on zero input. It has to be said that the Σ_n-limit rules are increasingly complex as n increases, and even for $n = 3$ the intuitive rationale of them is not obvious: instead of taking a 'liminf' along *all* ordinal times $t < \lambda$, one takes an unbounded subset of them which reflects some stable informational content. If anything the machine has said during the course of computation before λ about some real appearing on

its tapes before stage α has also been said prior to α then α is said to be a point of such stable informational content. To calculate the value of $C_k(\lambda)$ we take a liminf then for those $\alpha < \lambda$ whose ordinals of stable informational content are the same as those of λ which are below α. To define rules for Σ_4 or higher one proceeds in an inductive manner. However, even for Σ_4, the rule becomes rather complex, as indeed is inevitable, as this reflects the extra quantifier and such rules seem to move away from any machine-like intuition.

Another possibility for ITTMs, given that we have relaxed the time element in computations, is to relax the space element. Why not have tapes with cells that are ordered corresponding to some ordinal α? Or indeed why not go the whole hog and imagine a tape with a cell C_β for every ordinal number β? We stay with the original Turing programs but now have the possibility that the R/W head may escape from the finite numbered cells to, first of all, C_ω. We redesign the R/W rules so that at a limit stage its position is now the liminf of previous stages, even if that liminf is ω or some larger limit ordinal. We prevent the head moving back one cell, if it is at a limit cell and the program instructs it so to move (which it cannot do, since by definition a limit numbered cell has no immediate predecessor); then instead by *fiat* it gets sent back to C_0. Otherwise all is as before. Such machines now operate not just with infinite sequences in $2^{\mathbb{N}}$, but actually with potentially ON length such sequences of 0s and 1s. Whilst we will not go into detail here, such sequences potentially could code any set at all (assuming the Axiom of Choice), and one finds that such machines can output (a code of) any set X that occurs in Gödel's hierarchy of constructible sets L, with which he proved the consistency of the Axiom of Choice and the (Generalised) Continuum Hypothesis along with the other axioms of set theory. Such 'ON-ITTMs' thus give another presentation of this L-hierarchy.

If we restrict the lengths of such tapes to suitably closed limit ordinals α, called 'admissible ordinals', then one obtains a presentation of the sets which occurred in theories of 'α-recursion theory' in the early and middle 1970s. (The theoretical match-up here is not precise, because such recursion theories are involved very much with Σ_1 or extensionally defined processes whereas these machines are producing output by using a Σ_2-rule.)

We now consider possibility (ii), of generalising other machines from the finite arena. The immediate candidate is the notion of *register machine* (RM), due to Shepherdson–Sturgis and Minsky. Such a machine has a finite number of registers $R_0, \ldots, , R_n$ to hold natural numbers. It also has an instruction set: to set to zero, or to increment by 1, a register; to transfer the contents of register R_i to R_j, say (any $i, j \leq n$); and to *jump* from instruction I_p to I_q if a particular register content is equal to 0. It is a long-established fact that the RM model computes exactly the same functions as the TM model. Indeed a *universal* RM program can be written

which requires surprisingly few registers. We may define a transfinite behaviour for such machines, by the same liminf rule for the 'next instruction' at limit stages. Now, for each register R_k, we may set the value of $R_k(\lambda)$ at some limit time λ to be the $\liminf_{\alpha\to\lambda} R_k(\alpha)$ of the previous values *unless* that liminf is ω, in which case we re-set the register value to 0. (In this way we ensure that only natural numbers occupy registers.) This completely specifies the infinite time register machine (ITRM). Several surprises are in store: in contradistinction to the finite case RMs, there is no universal ITRM or program. This is essentially because one can show that, as we increase the number of registers, the power of the machines strictly increases. There is also a marked difference between the power of the ITRMs (taken together) and that of the ITTMs, which are, by a long way, considerably stronger. *Proof theorists* who try to measure the logical strength of various axiomatic theories have analyses of most mathematical systems that occur in mathematics or mathematical physics; however, the strength of the ITTMs outstrips our current proof-theoretical knowledge. ITRMs outrun Kleene recursion (just) since there is a fixed Turing program that when run on an ITRM (with a fixed, but small, number of registers) with an oracle $Z \in 2^{\mathbb{N}}$ can determine whether Z codes a wellordering. Indeed we have an exact measure of the theoretical strength of the class of the ITRMs taken as a whole in terms of a classical theorem of mathematical analysis: the latter states that any *closed* subset, C, of the real line $\mathbb{R}$ is the union of two parts; the first is a countable set of 'isolated' points (meaning that any such point has a neighbourhood which has no other points of C in it); the second part is a 'perfect' set of points, meaning that for every point p in the perfect part, the opposite happens: *every* neigbourhood of p contains an element of C. This theorem can be derived from the statement: *every ITRM with any number of registers, and any oracle Z, either halts or goes into a loop*. And the converse also holds true: from the theorem on closed sets we can deduce the assertion of the ITRM behaviour. We thus have two statements seemingly remote from one another actually being equivalent in the sense that each is provable from the other. (Strictly speaking we should also be specifying a weaker *base theory* in which we make these comparisons.)

Having defined RMs containing numbers, we could allow registers to contain ordinals – as these just generalise natural numbers. Now we no longer need to reset register contents if they get out of hand: if the liminf of a register content becomes infinite, or reaches a limit ordinal, then that ordinal is placed in the register at that limit stage. We have to specify an action if we try to decrease a register by 1 when it contains a limit ordinal; let us say that in this case the register content is set to zero. It can be shown that, remarkably, even with a small number of fixed registers such a machine can correctly decide the truth value of statements about any sets from Gödel's constructible hierarchy L of sets. That this works relies on the fact that any constructible set occurs at some least level L_α of that hierarchy, and each

level is itself well-ordered. We may thus label each constructible set with a finite sequence of ordinals $\langle \alpha, \ldots \rangle$; any formula about a finite sequence of such sets is then transformed into one about a finite sequence of ordinals, which may be coded up via pairing functions and submitted to a register machine.

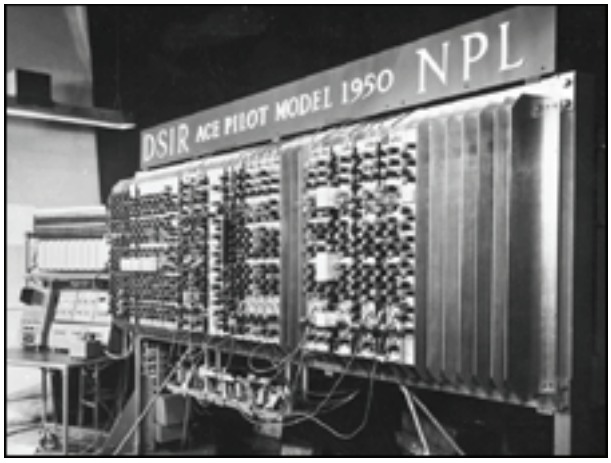

Figure 14.8 The pilot ACE computer.

14.9 Conclusions

What we have not done here is mention other discrete but potentially transfinite processes: automata theorists for example have also models with automata recognising words consisting of infinite sequences of 0s and 1s. We we have not touched on these. There are other models of computation: the Blum–Shub–Smale model of computation on algebraic rings – with one example being the ring of reals $\mathbb{R}$. Such a model also considers computations that halt in finite time where now a real number (which, we have already said, is also an infinitistic object, as in general it requires an infinite expansion of decimal or binary digits to specify) is treated as a single object of computation. Can these models also be remastered into a transfinite format? Lastly, there are many infinite mathematical processes, such as occur in ergodic theory or the theory of group actions. If we leave the machine model behind perhaps such mathematical processes, which are usually considered as limits of ω many natural-number steps, can be extended into the transfinite beyond ω?

As an example one may consider the action of an n-register ITRM as acting on an n-dimensional torus lattice of positive integer points. This gives it a quasi-dynamic feel, and questions asked about functions acting on points of such a torus can be translated into ITRM questions and vice versa. The results on the analytical

strength of the ITRMs are then equivalent statements about the functions on the torus. Some of these no doubt will turn out to be rather trivial or uninteresting extensions, but this may not be the case for them all; this is indeed speculative, but there may be some fascinating mathematics to uncover.

In conclusion, then, the above examples illustrate the richness of Turing's original conception: one that goes far beyond his original purposes. We may think in these 'mechanistic' terms in ways that he did not intend but which his model, and his example, has inspired.

References

J.D. Hamkins and A. Lewis (2000). Infinite time Turing machines. *J. Symb. Logic*, 65(2):567–604.

M. Hogarth (2004). Deciding arithmetic using *SAD* computers. *British J. Phil. Sci.*, 55:681–691.

R.I. Soare (2009) Turing oracle machines, online computing, and three displacements in computability theory. *Ann. Pure Appl. Logic*, 160:368–399.

P.D. Welch (2008). Turing unbound: the extent of computation in Malament–Hogarth spacetimes. *British J. Phil. Sci.*, 59(4):768–780.

P.D. Welch (2009). Characteristics of discrete transfinite Turing machine models: halting times, stabilization times, and normal form theorems. *Theor. Comp. Sci.*, 410:426–442.