DETERMINISM, MECHANISM, AND EFFECTIVE COMPUTABILITY

"Garbage in . . . garbage out"
"Yeah but is it computable garbage?"
(Graffiti from wall of Men's Room,
Experimental Engineering Bldg.,
University of Minnesota)

The examples of determinism studied in previous chapters should make it clear that determinism does not entail mechanism in the crude sense that determinism necessarily works by means of a mechanical contrivance composed of gears, levers, and pulleys. But it remains open that determinism involves mechanism in the more abstract sense that it works according to mechanical rules, whether or not these rules are embodied in mechanical devices. In the converse direction we can wonder whether mechanistic rules are necessarily deterministic. To make such questions amenable to discussion we need a model of mechanism and a codification of the rules by which the model works. I will take as the starting paradigm of mechanism the device which increasingly and irresistibly colors modern life — the digital computer. To understand the gist of operation of these devices it is best not to get too abstract too quickly, but to begin with the minimal embodiment described by Alan Turing in 1937.

1. TURING MACHINES

The inputs and outputs to a Turing machine are recorded on an infinite paper tape which is divided into squares. In each square one of three symbols, '0', '1', or '*B*', appears. In its pristine state, before input, the tape is completely blank ('*B*' printed in each square). The machine 'scans' one square at a time and performs one of the following basic operations: it erases the symbol in the square it is currently scanning and prints one of the other symbols; it shifts one square to the left; it shifts one square to the right; or it puts up a flag and halts. For sake of definiteness, we can suppose that one basic operation is performed per second. The sequence of operations is governed by a finite list of

deterministic rules. The guts of the machine, as distinguished from the tape, are at any instant in one of a fixed finite set of states, $s_1, s_2, \ldots, s_N$, one of which is the starting state s^* and another of which is the halting state s^{**}. The rules of performance have the following form: if at t_i the internal state is ___ and the symbol on the square being scanned is ___, then perform the basic operation ___ and shift into state ___ at t_{i+1}. Determinism here simply means that no two rules have the same filling for the first two blanks but a different filling in either of the last two blanks, and that there is a rule to cover each possible combination of initial internal state and tape symbol. It is also understood that the rules are time translation invariant, i.e., they are independent of the index i on the time t_i. An input to the machine will be a code $\bar{m}$ for a natural number m with '0' representing 0 and a string of m consecutive '1's representing a positive m. By convention, the input code is flanked on both sides by 'B's with the first blank to the right of the code being the square scanned when the machine is in the start state. If for given input the machine does not halt, then by definition there is no corresponding output. But if for given input the machine does halt, then the corresponding output is defined as the tape code when s^{**} is reached, and it is arranged that the output code is flanked by 'B's with the machine resting on the first 'B' to the left of the output (see Fig. VI.1).

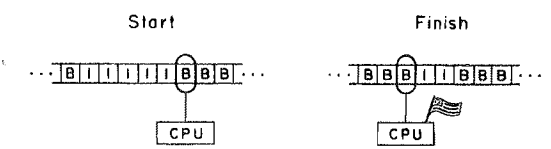

Fig. VI.1

With any such machine we can associate a partial function $f: \mathbb{N} \to \mathbb{N}$: for $m \in \mathbb{N}$, if the machine does not halt for input $\bar{m}$, then $m \notin \mathrm{dom}(f)$; if the machine does halt for input $\bar{m}$ giving output $\bar{n}$, then $f(m) = n$. A (partial) function of the natural numbers will be said to be *Turing computable* just in case it is associated with some Turing machine.

It is intuitively compelling that any such function should count as being effectively, mechanically computable, at least in principle. What is not so clear is the converse. Part of the worry here can be removed by proving that we do not get a larger class of computable functions by enlarging the alphabet, or by allowing the machine to scan more than

one square at a time, or by using two tapes instead of one, etc. But such results do nothing to assuage the worry that a mechanical device operating in a very different fashion could compute a Turing uncomputable function. For example, we might start with the Turing hardware but operate it by non-deterministic rules which, for given internal state and state of the square being scanned, allow for a (finite) number of choices consisting of the next basic operation and the next internal state. Of course, such a non-deterministic Turing machine cannot be directly regarded as a function computing device if, for some input, it halts for some but not all subsequent histories or else gives different outputs for different halting histories. Nevertheless, we will say that f is non-deterministically computable just in case there is a (possibly) non-deterministic Turing machine such that for each $m \in \mathrm{dom}(f)$, the machine halts in some allowable history following the input $\bar{m}$, giving the output $\bar{n}$ where $n = f(m)$, and for $m \notin \mathrm{dom}(f)$, the machine does not halt in any allowable history following the input $\bar{m}$. Intuitively, such a function should count as being effectively computable; for we can effectively generate sequences of selections from a finite number of choices, and combining such a sequence with a non-deterministic Turing machine gives unambiguous instructions for computing a (partial) function. So, evidently, determinism is not essential to this form of mechanism. But any function computable by a non-deterministic Turing machine can be computed by a standard Turing machine. This can be seen by, first, using a three tape Turing machine which employs one tape to hold the input, a second to generate a sequence of choices, and a third to reproduce the results of one of its non-deterministic cousins for the generated choices, and by, secondly, appealing to the fact that a standard Turing machine can do everything a multi-tape version can do. (In a sense, computing power may be gained by going to non-deterministic machines, for by making clever or lucky choices non-deterministic Turing machines may be able to accomplish tasks in a smaller number of steps than their deterministic brethren. The following is, apparently, an open question: If $f(x)$ is computable by a non-deterministic Turing machine in a time $t(x)$ which is a polynomial in x, is it also computable in polynomial time on a deterministic machine?)

One can still worry that some altogether different device, not resembling either a deterministic or non-deterministic Turing machine in hardware or software, could compute a function not computable by either type of Turing machine. Such doubts can never be entirely

banished, but they are mitigated by the remarkable convergence of a number of independent lines of investigation. For example, Kleene offered a different definition of effectively computable functions of $\mathbb{N}$ using the concept of recursiveness; but his definition picks out exactly the same class of computable functions as does Turing's. And other definitions by Church, Markov, and Post and others also prove to be extensionally equivalent to Turing's.[1]

Church's thesis (as it is called) that effective, mechanical computability for functions of $\mathbb{N}$ is to be identified with Turing computability, is now accorded such faith that it is an acceptable mode of informal argumentation to conclude Turing computability or recursiveness from the existence of an informal algorithm.[2] The trick is to recognize when an informal procedure corresponds to a genuine algorithm.

2. DETERMINISM AND EFFECTIVE COMPUTABILITY: FIRST TRY

The starting question as to whether determinism implies mechanism can now be reformulated as a series of questions about effective computability:

> If the laws L are Laplacian deterministic, does it follow that there is an effective procedure for generating the solutions of initial value problems? Will the (unique) solution of any given initial value problem be an effectively computable function? Will any solutions determined by effectively computable initial data be effectively computable? Will effectively computable initial data always be transformed into data which, at each future instant, will also be effectively computable?

To facilitate thinking about these questions, consider a deterministic discrete state system that operates in discrete time. At each instant, $t = 0, \pm 1, \pm 2, \ldots$, the state of the system is given by specifying a non-negative integer ('occupation number') for each of an infinite number of slots (e.g., number of balls in an urn, number of atoms at a given energy level). Thus, a history of the system is a function ϕ: $\mathbb{N} \times \mathbb{Z} \to \mathbb{N}$, where $\phi(m, n)$ is the occupation number of slot m at time n. The reader can amuse herself by constructing examples where the allowed histories form a deterministic set but the questions posed above have negative answers. Try, for example, to design deterministic transition laws so that for some allowed history ϕ, $\phi_0(\cdot) \equiv \phi(\cdot, 0)$ is Turing computable (i.e., the occupation numbers at time $t = 0$ are effectively enumerable) but for some $n > 0$, $\phi_n(\cdot) \equiv \phi(\cdot, n)$ is not

Turing computable (i.e., the occupation numbers at $t = n$ are not effectively enumerable).

However, there is little payoff to be gained in pursuing such examples unless they can be brought to bear on realistic physical systems. And here we meet a conundrum: all of the examples we have studied from physics involve functions of the real numbers, but, so far, we have given to characterization of effective computability for such objects. A construction by Grzegorczyk promises to fill the gap.

But before turning to functions of the reals, it is worth noting that the discrete state machine contemplated above affords another means of characterizing effective computability for functions of the integers that is extensionally equivalent to Turing's but conceptually more appealing. An *unlimited register machine* (URM)[3] consists of an infinite number of registers R_1, R_2, ... each of which holds a natural number r_n. A program for operating this machine consists of a finite list of instructions of four basic types. First, a zero instruction changes the contents of a designated R_n to 0 while leaving the other registers unaffected. Second, a successor instruction adds 1 to the contents of a designated R_n while leaving the others unaffected. Third, a transfer instruction interchanges the contents of two designated registers R_n and R_m again leaving the others unaffected. And finally, a jump instruction compares the contents of R_n and R_m and orders the machine to proceed to intruction number q or else to the next instruction according as $r_n = r_m$ or not. A (partial) function f: $\mathbb{N}^n \to \mathbb{N}$ is said to be URM computable if there is an URM which computes it in the following sense: if $(a_1, a_2, \ldots, a_n) \notin \mathrm{dom}(f)$, then the machine does not halt when the initial contents of the registers are $a_1, a_2, \ldots, a_n, 0, 0, \ldots$; but if $(a_1, a_2, \ldots, a_n) \in \mathrm{dom}(f)$, then with the initial contents $a_1, a_2, \ldots, a_n, 0, 0, \ldots$ the machine does halt with $b = f(a_1, a_2, \ldots, a_n)$ in R_1. (To make the URM behave as a deterministic system in our sense, with the contents of the registers at any time determining the contents at any later time, we would need to add an initial register R_0 to hold the number of the next instruction to be executed and also add to the programming instructions a rule to modify r_0 in the appropriate way. I leave it to the reader to supply the details.)

Combining the remarks from the first part of the section with the URM characterization of computability reawakens worries about Church's thesis. There are innumerable numbers of deterministic ways to run the register machine that outstrip any standard URM. Why

can't some of these ways be used to compute a Turing uncomputable function? Such questions will be held in abeyance until Sec. 7.

3. GRZEGORCZYK COMPUTABILITY

The strategy is to move along the well-charted path from the integers through the rationals to the reals, effectivizing definitions as we go. Thus, a sequence of rational numbers $\{x_n\}$ is said to be effectively computable just in case there are three Turing computable functions a, b, c from $\mathbb{N}$ to $\mathbb{N}$ such that $x_n = (-1)^{c(n)}a(n)/b(n)$. A real number r is said to be effectively computable if there is an effectively computable sequence $\{x_n\}$ of rationals which converges effectively to r, i.e., there is an effectively computable function d from $\mathbb{N}$ to $\mathbb{N}$ such that $|r - x_n| < 1/2^m$ whenever $n \geqslant d(m)$. Taking $\{x'_n\} = \{x_{d(n)}\}$, it follows that $|r - x'_n| < 1/2^n$ for all n. Continuing in this vein, a sequence $\{r_n\}$ of reals is said to be effectively computable if there is a computable double sequence $\{x_{kn}\}$ of rationals such that $|r_k - x_{kn}| < 1/2^n$ for all k and n.

It remains to say what an effectively computable function of the reals is. Grzegorczyk's (1955) concept of effective computability for a function f of the reals was originally stated in terms of recursive functionals, but this definition is not at all easy to apply to concrete examples in analysis. Grzegorczyk (1957) showed that the original definition is equivalent to several others, including the following which is the one most often used by analysts:

(i) f is sequentially computable: for each effectively computable sequence $\{r_n\}$ of reals, $\{f(r_n)\}$ is also effectively computable, and

(ii) f is effectively uniformly continuous on rational intervals: if $\{x_n\}$ is an effective enumeration of the rationals without repetitions, then there is a three place Turing computable function l such that $|f(r) - f(r')| < 1/2^k$ whenever $x_m < r$, $r' < x_n$ and $|r - r'| < 1/l(m, n, k)$ for all $r, r' \in \mathbb{R}$ and all $m, n, k \in \mathbb{N}$.[4]

Yet another equivalent definition in terms of an effective polynomial approximation of f is given by Pour-El and Caldwell (1975).

Provisionally accepting Grzegorczyk's definition, we will go on to use it to answer questions about the relation between determinism and effective computability in physics.

4. DETERMINISM AND EFFECTIVE COMPUTABILITY: ORDINARY DIFFERENTIAL EQUATIONS

Consider the first order ordinary differential equation

(VI.1) $\dot{\phi}(t) = F(t, \phi(t))$

subject to the initial condition

(VI.2) $\phi(0) = \phi_0$

Uniqueness for the initial value problem is not guaranteed if we merely required continuity of F. Suppose in addition we demand that F satisfy a Lifshitz condition on a rectangle about the origin. (Recall that this means that there are constants α, β, and K such that $|F(x, y) - F(x, y')| \leqslant K|y - y'|$ for all (x, y) in the rectangle $|x| \leqslant \alpha$ and $|y - \phi_0| \leqslant \beta$. K is called the Lifshitz constant.) Then local existence and uniqueness theorems for the initial value problem are forthcoming.

Moreover, the existence theorem actually provides an effective procedure for cranking out a sequence of approximations converging to the (unique) solution. The 0th approximation is just the initial value ϕ_0. The next approximation is

$$\text{(VI.3)} \quad \phi_1(t) = \phi_0 + \int_0^t F(\xi, \phi_0)\, \mathrm{d}\xi$$

and in general

$$\text{(VI.4)} \quad \phi_k(t) = \phi_0 + \int_0^t F(\xi, \phi_{k-1}(\xi))\, \mathrm{d}\xi, \; k = 1, 2, 3, \ldots$$

The convergence of this sequence is uniform, and it is also effective since given α, β, K and the bound $M \geqslant |F(x, y)|$ on the rectangle, we can effectively compute how many times the crank needs to be turned to come within the desired approximation of the solution. Further, if F is an effectively computable function and the initial value ϕ_0 is an effectively computable number, then the approximating functions ϕ_k are also effectively computable since plugging a computable number in a computable function, composing computable functions, and integration

are all computability preserving operations. The upshot is that if F and the initial data are computable, then so is the (unique) solution; for if $\phi_k \to \phi$ uniformly and effectively, then ϕ is Grzegorczyk computable if the ϕ_k are.

Of course, we know by cardinality considerations that most of the solutions of (VI.1)—(VI.2) will not be Grzegorczyk computable even when F is; for there are an uncountable number of solutions but only a countable number of computable functions. But it is remarkable that the solutions picked out by computable initial data are computable and that there is an effective procedure for generating them.

When the conditions needed to prove uniqueness are relaxed, then even though F is computable, the non-unique solutions need not be; in fact, there are cases where none of the solutions are computable (see Pour-El and Richards (1979)).

Thus, there is a strong and deep connection between determinism and effective computability for first order ordinary differential equations. Higher order ordinary differential equations can sometimes be reduced to a system of first order equations, in which case the connection between determinism and computability carries over. For partial differential equations the story is both more complicated and more interesting.

5. DETERMINISM AND EFFECTIVE COMPUTABILITY: PARTIAL DIFFERENTIAL EQUATIONS

For the non-linear shock wave equation (III.7) we saw that initial data $u(x, 0) = u_0(x)$ determines a unique weak solution $u(x, t)$, $t > 0$, if entropy conditions are imposed. We can choose $u_0(x)$ to be very smooth and Grzegorczyk computable. But in general the computability of the initial data is not preserved since $u_c(x) = u(x, c)$, $0 < c =$ constant, may not be continuous and, therefore, not Grzegorczyk computable.

For the classical heat equation (III.4) we saw that Laplacian determinism in the future direction holds if supplementary boundary conditions at infinity are imposed. We also saw that in the unique future solution the smoothness of the initial data does not degrade — just the opposite, any roughness in the initial temperature distribution disappears after even so short a time — so that a breakdown in computability cannot occur because of a loss of continuity. And in fact, a

computable initial temperature distribution determines a computable solution (see Pour-El and Richards (1983)).

The discussion of the relativistic wave equation (IV.1) requires that we respect the separation already made between the case of one-dimensional space and the case of higher dimensions. In the former case the form of the solution

$$\text{(VI.5)} \quad u(x, t) = \tfrac{1}{2}[f(x + t) + f(x - t)] + \int_{x-t}^{x+t} g(\xi)\, \mathrm{d}\xi$$

shows that if the initial functions $f(x) = u(x, 0)$ and $g(x) = \partial u(x, 0)/\partial x$ are computable, then so is the solution $u(x, t)$. In higher dimensions we saw that C^1 initial data can degrade to C^0 but not below, so computability for solutions determined by differentiable initial data will not break down for the sorts of reasons that applied to the shock wave case. Nevertheless, Pour-El and Richards (1981) have constructed examples for the three-dimensional case where $g(x^1, x^2, x^3) = 0$, $f(x^1, x^2, x^3)$ is C^1 and computable, but the unique C^0 weak solution is not Grzegorczyk computable. Further, they give an example where $f(x^1, x^2, x^3)$ is computable and therefore continuous but the corresponding solution $u_1(x^1, x^2, x^3) = u(x^1, x^2, x^3, 1)$ at $t = 1$ is continuous but not computable.

6. EXTENDED COMPUTABILITY

On Grzegorczyk's definition, discontinuity automatically brands a function of the reals as being non-computable. This is counterintuitive,[5] as indicated by the simplest example of a discontinuous function, a step function such as $s(x) = c$ for $x < x_0$, d for $x \geqslant x_0$. If the jump point x_0 and the jump values c and d are computable numbers, then we would like to be able to count s as being computable. This can be done while maintaining contact with Grzegorczyk's approach, for we can say that the step function s is computable because it can be effectively approximated by Grzegorczyk computable functions, at least if we are willing to measure the degree of approximation in a sufficiently liberal way. For simplicity, restrict attention to functions defined on a compact interval $[r_1, r_2] \in \mathbb{R}$. The $L^p[r_1, r_2]$ norm for such a function is defined as

$$\|f\|_p \equiv \left[\int_{r_1}^{r_2} |f(\xi)|^p \, \mathrm{d}\xi \right]^{1/p}$$

It is easy to see that taking the effective closure of Grzegorczyk computable functions in, say, L^1, captures our step functions. It turns out the choice of p does not matter much: for any $1 \leqslant p < +\infty$ the step functions captured in this way are exactly the ones with computable jump point and jump values (Pour-El and Richards (1983)). In general, we can say that f is computable in the norm $\| \ \|$ if there is a sequence of Grzegorczyk computable functions $g_1, g_2, g_3, \ldots$, such that $\|f - g_k\|$ converges effectively to 0 as $k \to \infty$. If $\| \ \|$ is the usual sup norm then we collapse back to the original class of Grzegorczyk functions.

We now have to rework the cases where determinism did not preserve Grzegorczyk computability and ask whether computability in the extended sense for some norm appropriate to the problem is preserved. The qualifier 'appropriate' introduces an annoying vagueness, but in various concrete cases bounds on appropriateness are usually evident.

For the relativistic wave equation what was true for Grzegorczyk computability continues to be true for the extended concept of L^p computability: in three spatial dimensions the wave equation does not preserve computability in the L^p norm ($p < +\infty$). However, in the energy norm, computable initial functions determine a computable solution (see Pour-El and Richards (1983)).

For the shock wave equation we can restrict attention to cases where $u(x, 0) = u_0(x)$ has compact support; then, $u_c(x) = u(x, c)$, $c > 0$, also has compact support. The appropriate norm here seem to be L^1, and for this choice computability in the extended sense need not be preserved since discontinuities in $u_c(x)$ need not occur at computable points or with computable jump values.

7. GENERALIZED COMPUTABILITY

Turing machines, URMs, and the other devices commonly used to characterize effective computability are very special examples of deterministic systems. It is therefore natural to wonder whether more general deterministic systems can be used to 'effectively compute' functions which are not Turing-Grzegorczyk computable. The scare quotes are an acknowledgment of the danger that too hasty a generalization may lead to triviality. If we allow the initial state of an URM to contain too much information, any total function f of $\mathbb{N}$ becomes 'computable'. Take the

initial contents of the registers to be a, $f(1)$, $f(2)$, ... and take the program instructions to transfer the contents of R_1 and R_{a+1} and halt. A similar trivialization occurs if the program is allowed to contain an infinite list of instructions. And beyond these trivial trivializations, other more interesting ones lurk. While being aware of the trivialization danger, we should not allow it to prevent us from exploring non-Turing notions of computability.

For functions of the reals more than idle curosity motivates the exploration. It seems more natural to try to characterize computability for these functions directly in terms of analogue computers which are designed to handle continuous variables than in terms of computations on rational approximations to reals. It remains to be seen how analogue computability of functions of the reals is related to Grzegorczyk computability.

A paradigm example of a general purpose analogue computer is the differential analyzer, used to solve ordinary differential equations. Each variable in the equation corresponds to a shaft in the analyzer with the value of the variable being proportional to the number of rotations of the shaft. Mechanical connections among the shafts are used to enforce the desired mathematical relations among the variables. Calling the independent variable t and the dependent variables $v_1, v_2, \ldots, v_M$, Claude Shannon (1941) says that the system of equations is *solvable* by the analyzer just in case when its independent shaft t is turned, the dependent shafts $v_1, v_2, \ldots, v_M$ turn in accord with the equations for any initial conditions. He says that a function f of $\mathbb{R}$ can be *generated* by the analyzer just in case it is a solution function $f(t) = v_i(t)$ for some i and some initial conditions.

If we abstract from the hardware, we are left with not much more than the idea of a system which is governed by laws L that are futuristically deterministic and time translation invariant (so that the development of the system does not depend on the instant at which the initial conditions are specified, as is discussed more fully in Ch. VII). If $(\mathring{v}_1, \mathring{v}_2, \ldots, \mathring{v}_M)$ is an allowed initial state at t_0, the uniquely determined solution functions $v_i(t)$, $t \in [t_0, +\infty)$, can be said to be analogue computable relative to the laws L. The justification for this terminology is that there is a computer — Nature herself — which computes the function as follows. We prepare the system in the state $(\mathring{v}_1, \mathring{v}_2, \ldots, \mathring{v}_M)$ at any instant t_0, wait $t - t_0$ seconds, and then read off the values displayed by the system.

Shannon (1941) and Pour-El (1974) consider laws in the form of first order ordinary differential equations representing the operation of a general purpose analogue computer which performs the operation of integrating a variable, multiplying a variable by a constant, and multiplying and adding two variables. They show that hypertranscendental functions are not analogue computable (or generable in Shannon's terminology) by such a device. It follows that these devices are not capable of computing (or generating) some functions which are digitally computable by approximations, for some hypertranscendental functions, such as the reciprocal of the gamma function, are known to be Grzegorczyk computable. One would like to construct a more general analogue computer which would compute all Grzegorczyk computable functions, or else show why this is not possible.

Montague (1962) formalized the notion of a generalized computer which computes functions as follows. If x is an argument for which the function value is desired, the signal input variable w_1 is brought to its special starting value w^* and the argument input variable w_2 is brought to the value x. The computer then chugs away until the output signal variable w_3 flashes its special value w^{**} indicating that the simultaneous value y of the output variable w_4 is to be read as the value of the function for the argument x. This generalized computer may be run according to either deterministic or indeterministic laws. In either case the laws are assumed to be time translation invariant, and for simplicity it is assumed that if the signal output variable takes on its special value w^{**} then there is a first instant at which it takes it. The deterministic mode of operation requires of the laws L that for any histories, primed and unprimed, and any times t_0 and t_1, if $w_i'(t_0) = w_i(t_0)$, $i = 1, 2, 3, 4$, $w_1'(t_0) = w^*$ and if $t_1 \geqslant t_0$ is the first instant at which $w_3'(t_1) = w^{**}$, then $w_i'(t_1) = w_i(t_1)$, $i = 1, 2, 3, 4$. The value y is said to be computed for the argument x just in case there is an allowed history and times t_0 and t_1 such that $w_1(t_0) = w^*$, $w_2(t_0) = x$, $t_1 \geqslant t_0$ is the first instant at which $w_3(t_1) = w^{**}$, and $w_4(t_1) = y$. The function f is said to be computable by the computer just in case for every $x \in \mathrm{dom}(f)$, $y = f(x)$ is the value computed for the argument x, and for every $x \notin \mathrm{dom}(f)$ no value is computed for the argument x. In the indeterministic mode of operation it is required that for any allowed histories, primed and unprimed, and for any time t_0, if $w_1'(t_0) = w_1(t_0) = w^*$ and $w_2'(t_0) = w_2(t_0)$ and if $t_1 \geqslant t_0$ is the first instant such that $w_3(t_1) = w^{**}$, then there is also a first instant $t_2 \geqslant 0$ such that $w_3'(t_2) = w^{**}$ and further $w_4'(t_2) = w_4(t_1)$.

From the perspective of foundations it is useful to have a notion of computability which is general enough that the various notions of digital and analogue computability can all be obtained by specializing the variables and the laws L by which the generalized computer operates. That is what Montague's approach promises to provide, at least with a little fiddling which I will not attempt here. But the worry arises that the sense of computability involved is so general as to be useless; for if L is allowed to range over every kind of deterministic law then presumably few if any functions will fail to be generalized computable. The worry can be assuaged by emphasizing that the generalized concept of computability is relativized to laws L. The subject is given content by proving results about what functions are and are not generalized computable relative to what laws, especially the kinds of laws encountered in mathematical physics. As usual, I leave it to the reader to supply the labor.

8. OBJECTIONS; CHURCH'S THESIS REVISITED

The notions of generalized computability introduced in the preceding section are open to a series of objections which are worth reviewing because their resolution serves to clear away several potential misunderstandings. The consideration of the objections also serves as an opportunity to the reconsider the meaning of Church's thesis.

Objection 1. Your generalized 'computations' are presented in terms of a Big Computer in Sky. This mythical machine violates a basic part of any plausible definition of 'computer'. In his book *Analogue Computation*, Albert Jackson writes:

> A computing device may be defined as a device that accepts quantitative information, arranges it and performs mathematical and logical operations on it, and makes a available resulting quantitative information as its outputs. (1960, p. 2)

But your Big Computer in the Sky does not make outputs available as information in the sense of symbols printed on paper and the like.

Response. I view this aspect of computation as an engineering problem and not as part of the analysis of computability per se.

Objection 2. You are missing the point. The key idea is that a computer accepts and then outputs information. The exact definition of 'information' is not at issue here except in so far as it necessarily involves symbolic representation. Thus the maxim, 'No computation

without symbolic representation', which is grossly flouted in your notions of generalized computability. *Response.* A definition of computable function might but need not make use of a coding of values by symbols. Whenever possible it seems best to characterize computability, digital or analogue, without reference to coding; for while no one can doubt that the standard Turing machine coding — a string of m consecutive '1's to code the positive integer m — is an effective coding, the general problem of distinguishing effective from non-effective codings is equivalent to the problem of deciding when a function is effectively computable, the very problem at issue. (It should be obvious, for example, that if the coding of integers is not effective, then any set of integers could be enumerated by a Turing machine.) In any case, when we move from an abstract generalized computer to a physical realization of it, there is representation; e.g., the variable v_3 (say) takes as values volts and r volts represents the real number r.

Objection 3. What we want of a representation is that it enables us to access the information, and the standard systems of symbolic representation, so conspicuously lacking in your characterization of generalized computability, are designed to guarantee such access. *Response.* Epistemic access is an important issue. And I would suggest that it is not merely the natural tendency to anthropomorphize when explaining the intuitive basis of computability but also the concern with epistemic access that explains the presence in Turing's original paper of such phrases as "scanning" a square, "immediate recognizability" of changes in squares, and "states of mind" of the computing agent. This concern also seems to be present in contemporary presentations of algorithms in terms of "a computing agent . . . which can react to the instructions and carry out the computations" (Rogers (1967), p. 2). But, to repeat, I do deny that computability is an epistemological concept. Turing computability can be presented in a purely abstract fashion, avoiding questions of representation and epistemic access; and just as the mathematical theory of Turing computability can be developed independently of these questions, so can the theory of generalized computability.

Objection 4. Granting for sake of argument that the generalized notions of computability do capture legitimate senses of effective, mechanical computability, these senses are so different from Turing's original sense that it is misleading to speak as if there were a unified concept of computability which covers all the bases. *Response.* It is and it isn't. It is useful to see Turing computability as a special case of a general notion of effective computability which covers digital and

analogue computers and other deterministic devices as well. But Turing computability is such an important and distinct special case that it deserves special handling.

Start with the notion of a programmable or algorithmically computable function of the integers. Roughly, this is a function which is computable by means of a stepwise discrete procedure which can be carried out according to a finite list of instructions each of which . . . (For the ellipsis, fill in your favorite intuitive account.[6]) Church's thesis, or proposal as I would prefer to call it, says:

(CP1) The class of programmable or algorithmically computable functions of the integers is to be identified with the Turing computable functions.

I have no doubts about the adequacy of (CP1)[7], especially as regards the originally intended application to Hilbert's decision problem. (Recall that Turing's original paper was entitled "On Computable Numbers, with an Application to the Entscheidungsproblem," and that Church's notion of effective calculability was introduced in the papers "An Unsolvable Problem of Elementary Number Theory" and "A Note on the Entscheidungsproblem.")

Church's initial proposal (CP1) could be extended to functions of the reals by

(CP2) The class of programmable or algorithmically computable functions of the reals is to be identified with the Grzegorczyk computable functions.

However, (CP2) does not carry the conviction of (CP1) because Grzegorczyk's definition, though useful for proving results in analysis, is only one of various possible ways to generalize Turing computability to functions of the reals.

It is here that Turing's (monumental!) contribution ends. Turing is sometimes represented as having set and achieved the more ambitious goal of specifying the most general notion of what a 'machine' is and then using this notion to explicate the general notion of a mechanically computable function.[8] This corresponds to a third proposal.

(CP3) The class of effectively, mechanically computable functions is to be identified with the class of programmable or algorithmically computable functions and, thus, with the Turing-Grzegorczyk computable functions.

Even leaving aside the qualms about (CP2) that infect (CP3), (CP3) is unacceptable, for it is simply wrong that a Turing machine is the most general type of machine that can perform what is recognizably an effective, mechanical computation of a function. What is true is that the theory of non-Turing computability remains to be developed. Whether the development along the lines suggested in Sec. 7 above will prove to be worth the effort remains to be seen.

9. CONCLUSION

Our starting question about the relation between determinism and mechanism can be given a partial answer. Determinism does not necessarily entail mechanism in the Turing-Grzegorczyk sense of effective computability, but various interesting partial entailments hold for many types of deterministic laws in the form of ordinary and partial differential equations. In the converse direction, effective, mechanical computability does not entail determinism, but any function which can be computed by an indeterministic Turing machine can also be computed by a deterministic Turing machine (though the computations of the latter may not be as efficient). In general, however, half of the question tends to collapse, for any deterministic and time translation invariant system can be regarded as an analogue computer which computes values of the solution functions. As for the other half of the question, an analogue computer need not operate deterministically, but whether the resort to indeterminism enlarges the class of analogue computable functions is a matter that has to be settled on a case by case basis.

NOTES

[1] Rogers (1967) and Tourlakis (1984) provide comprehensive treatments of this and other topics touched on in this section.

[2] For examples, see Rogers (1967).

[3] These machines were first discussed in Shepherdson and Sturgis (1963). A detailed development is given in Cutland (1980) whose presentation I follow here.

[4] For functions of the reals with domains confined to a closed and bounded interval with computable endpoints, clause (ii) simplifies to the requirement that there is a one-place Turing computable function $l: \mathbb{N} \to \mathbb{N}$ such that $|f(r) - f(r')| < 1/2^k$ whenever $|r - r'| < 1/l(k)$. Clause (i) corresponds to S. Mazur's (1963) definition of computable function of the reals. There are weaker versions of clause (ii) which give rise to notions

of computable function of the reals weaker than Grzegorczyk's. Grzegorczyk's definition has some nice consequences to which I will appeal to below. For example, if $\{f_k\}$ is a sequence of Grzegorczyk computable functions that converges uniformly and effectively to a function g, then g is Grzegorczyk computable; further, Riemann integration preserves Grzegorczyk computability.

[5] Or rather intuitions divide. The step function is clearly not computable in the sense of being 'programmable' since there is no recursive procedure to decide whether x equals x_0 (where x_0 is assumed to be a computable real).

[6] See, for example, Rogers (1967), Ch. 1.

[7] But for the record I note that others have expressed doubts in both directions, some questioning the idea that all recursive functions are effectively computable, and others questioning the converse; see Péter (1957), Bowie (1972), and Thomas (1973).

[8] I do not know whether this representation is historically accurate, though I suspect that it contains a kernel of truth. For example, because he thought of machine states on the analogy with states of mind, Turing (1937) ruled out machines in which the states can get arbitrarily close together, because otherwise the computing agent might confuse them.

SUGGESTED READINGS FOR CHAPTER VI

Alan Turing's (1937) original paper "Computable Numbers" still rewards the effort of reading. Cutland's (1980) *Computability*, Rogers' (1967) *Theory of Recursive Functions and Effective Computability*, and Tourlakis' (1984) *Computability* cover the standard topics in algorithmic computability for functions of the integers. For an introduction to computable analysis, see Aberth's (1980) book of that title. The series of papers by Pour-El and Richards cited in the text provide definitive results about the preservation of Turing-Grzegorczyk computability by linear partial differential equations. Pour-El's (1974) "Abstract Computability and Its Relation to the General Purpose Analogue Computer" presents a concept of analogue computability that covers the types of analogue computers actually in use.