How Gödel's Theorem Supports the Possibility of Machine Intelligence

Taner Edis
Department of Physics, Southern University and A & M College, Baton Rouge LA 70813, U.S.A. *

Abstract. Gödel's Theorem is often used in arguments against machine intelligence, suggesting humans are not bound by the rules of any formal system. However, Gödelian arguments can be used to *support* AI, provided we extend our notion of computation to include devices incorporating random number generators. A complete description scheme can be given for integer functions, by which nonalgorithmic functions are shown to be partly random. Not being restricted to algorithms can be accounted for by the availability of an arbitrary random function. Humans, then, might not be rule-bound, but Gödelian arguments also suggest how the relevant sort of nonalgorithmicity may be trivially made available to machines.

Keywords: human nonalgorithmicity, Gödel's Theorem, randomized computation, randomness

1. Not bound by rules?

Gödel's celebrated Incompleteness Theorem has often been thought to demonstrate the impossibility of a completely materialist theory of mind. More concretely, it has suggested that machine intelligence is a fundamentally misguided research program. The Gödelian argument takes may forms. For example, Lucas (1961) observed a human would be able to construct a Gödel sentence for any given formal system powerful enough to be interesting. The human mind would always be able to do better than a machine by virtue of not being bound by any rigid set of rules. There appears to be a basic 'Gödelian intuition' common to most critiques of AI based on Gödel's Theorem :

> (G1) Human Intelligence is not bound by rules. We are not locked into any formal system, always able to step outside of any set of axioms we happen to be working in.

If this is correct, no algorithm, however complex, will be able to adequately simulate human intelligence. The human mind 'computes' some kind of nonalgorithmic function.

(G1) is an empirical claim. However, it is difficult to interpret (Dennett 1972), and there is no test known to distinguish between human

* (e-mail edis@phys.subr.edu)

performance of a task and that of an arbitrarily complicated computer program. Algorithmicity simply is not a very strong constraint on behavior. One way to see this is to treat (G1) as a physical claim. Whether (Gl) – essentially a denial of the Church-Turing thesis for humans – is correct depends on features of the world, like whether physics is fundamentally continuous (Cleland 1993). But it is possible (though perhaps not likely) that one day we will discover our present physics is an approximation to an underlying 'digital mechanics' (Fredkin 1990). If this happens, we would expect practically nothing to change concerning the task of explaining human intelligence or achieving machine intelligence. Whether the universe is an extremely complex cellular automaton or not seems as relevant to AI as quantum mechanics is to civil engineering. Yet Gödelian arguments, like claims that classical mechanics cannot accommodate consciousness (see Ludwig 1995), turn on precisely such distinctions.

The most recent revival of the Gödelian argument (Penrose 1989, 1990, 1994) attempts to give claims like (G1) a firmer empirical basis. Penrose not only argues that humans are not restricted by algorithms, but also speculates on how the nature of human intelligence may be connected to the fundamental structure of physics. Penrose's views have drawn severe criticism,[1] and AI as a research program will have to stagnate considerably before any such grandiose anti-AI scenario gains acceptance. Nevertheless, (Gl) does capture something important about human intelligence. If it is only set aside as empirically unsupported, the Gödel–Lucas–Penrose argument will be periodically resurrected by critics of AI.

In the following, I will argue that something very like (G1) is probably true, though for quite ordinary empirical reasons having nothing to do with exotic new physics: The missing ingredient in an algorithm is something quite pedestrian: randomness. The Gödelian intuition, when understood properly, ends up supporting machine intelligence.

2. Nonalgorithmicity without oracles

If mind is beyond the machine, it must *at least* not be equivalent to a set of instructions some pile of electronics can execute. However, not just any nonalgorithmicity will do. For example, a random function is maximally nonalgorithmic, in the sense that only a finite number of bits in a random sequence can be specified by a finite algorithm (Chaitin 1987). But the ability to generate a haphazard bit sequence has no obvious connection either with intelligence or (G1). If human intelligence is distinctive, it must be able to do better than computers

in performing well-defined tasks with meaningful results. Consider Turing's halting function as an example of a meaningful nonalgorithmic function. We can specify exactly what h does, although we cannot produce an algorithm for it. For any program p,

$$h(p,n) = \begin{cases} \texttt{1} & p(n) \text{ halts,} \\ \texttt{0} & p(n) \text{ does not halt.} \end{cases}$$

Computing h would sharply set us apart from computers.

Having an oracle for a meaningful function like h is the most straightforward way of achieving nonalgorithmicity. However, this is an extravagant claim if made about humans. Furthermore, it is not clear what the availability of any definite oracle has to do with (G1). If the brain could somehow intuit h, we would still be rule-bound in many other tasks. To give the claim that humans are not bound by any system of rules concrete content, we have to see how nonalgorithmicity can exist without oracles for functions like h.

The Gödelian intuition can be stated in a modest form, without claiming the availability of any specific nonalgorithmic function. The thrust of (G1) is that humans are not subject to the kind of limitation which arises *solely* because an algorithm is a rule-bound way of performing a task. We can take our inspiration from Gödel's Theorem. We might have a theorem-proving algorithm, even a very good theorem-proving algorithm. But we can construct a case where it must fail solely because it is a system of rules. Perhaps, then, (G1) can be refined to situate human intelligence somewhere between rigid algorithms and magically available oracles:

> (G2) Human intelligence is not bound by rules, or restricted to any particular set of algorithms. We can succeed in cases where algorithms must fail on account of being algorithms, even though we do not have meaningful oracles available to us.

We can illustrate oracle-free nonalgorithmicity by a very simple cop-and robber game (Figure 1). There are two houses which a cop and a robber can move in between. The cop and robber start in different houses. Each turn, the players have the option of changing houses, or staying put. If they end up in the same house, the robber is caught and the game is over.

Let us say the cop conducts her house-to-house search according to regulations in the police manual. However, there is no general algorithm for catching the robber. For if the robber follows the same algorithm for switching or staying – perhaps having learnt the contents of the police manual – he will never be caught.

Even though this game is extremely simple, it has the basic Gödelian flavor. The algorithm in the police manual will fail for some robbers.

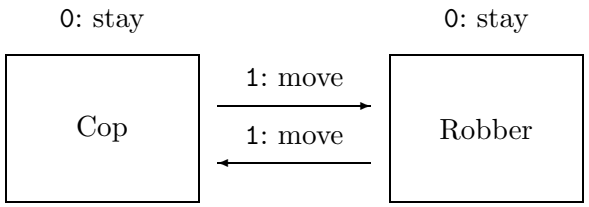

Figure 1. The moves of the cop and robber can be described as bit sequences. If the algorithms for the sequences match, the robber eludes capture.

The police chief knows exactly what kind of robber they will fail with. So, like adding a Gödel sentence to the list of axioms of a theorem-proving algorithm, the chief can try and compensate by complicating the police manual. But just as theorem-provers cannot be patched up, since they generate a new Gödel sentence, no amount of tinkering will perfect the police manual. A mathematician observing the game can always construct a robber-algorithm for which the cop *must* fail – it is simply the same algorithm she follows.

This game allows for very little difference between search strategies; other nonalgorithmic tasks are more interesting. However, solving the cop's problem, which arises entirely from being rule-bound, gives us an idea of what (G2) means. The solution is simple. In decision theory, games involving intelligent opponents can call for randomized decision rules (Berger 1980, pp. 10–13). If the cop acts randomly, tossing a coin to decide whether she will switch or stay, she will catch the robber in a finite number of turns. Capture is certain if the robber behaves according to an algorithm, and it has probability 1 otherwise.

This suggests that the key to constructing machines which behave according to (G2) is to equip them with a random bit generator. By virtue of being rule-free, and quite useless aside from being rule-free, a random function may be exactly what we need.

For randomness to help in less simple-minded tasks than capturing the robber, we need to be able to describe complicated strategies which are partly algorithmic and partly random. The natural way to do this is to use a random bit generator ρ as an oracle, i.e. employ a 'probabilistic Turing machine'.[2] This oracle does not stand for any meaningful nonalgorithmic function – it is only there to provide a haphazard bit sequence. In this way, we can describe more complex strategies. For example, a certain nonalgorithmic function u could be described as:

```
function u(n)
    if (n is odd) then u(n) = 1
    else u(n) = ρ(n)
```

```
end
```

According to (G2), we cannot rely on $\rho(n)$ producing any particular random sequence in order to make use of the function $u(n)$. Only the fact that ρ gives a patternless bit sequence should be relevant. This means the above description for $u(n)$ applies to a *set* of functions, and ρ must stand for the set of all random functions. So $u(n)$ describes the set of functions which take on random values for even arguments, and are equal to `1` otherwise. As a strategy, u must work independently of any particular function $r \in \rho$ which a random bit generator actually produces.

This gives us a language describing sets of functions, including at least some which are nonalgorithmic. We can use ρ in arbitrarily complicated ways to specify all sorts of functions in between algorithms and full-scale randomness. We will now need to prove a completeness theorem: *every* function, no matter how nonalgorithmic, falls in a set *finitely described* by a scheme like that for u above. Furthermore, every function has a finite 'minimal description' exhibiting its algorithmic structure, leaving its nonalgorithmic aspects to a random sector. We will use completeness to show how randomness, and only randomness, allows a machine to satisfy (G2).

The idea behind completeness is simple. We know how to specify algorithms for certain functions. We also have a concept of randomness: there are sequences for which no algorithm can specify more than a finite number of bits. Completeness is the claim that *all* nonalgorithmicity can be described by randomness. A finite partly random description exists for every function.

Completeness is also instrumental in blocking a possible objection to the thesis of this paper. If combining algorithms and randomness could not finitely describe some functions, there would be too much temptation to argue that the peculiar quality of mind resides in just that part of function space beyond both algorithms and randomness. While (G2) could still be correct, there would be another level of incompleteness which would give rise to another variety of Gödelian objection to AI. If Gödelian objections are not only to be deflected but harnessed in support of machine intelligence, we must demonstrate that the inclusion of randomness leads to a complete description scheme.

3. A completeness theorem

Following Chaitin (1987), we start with randomness in finite sequences. A finite binary sequence can always be described by specifying an algorithm. Sequences containing patterns, such as $o =$ `1111111111`...

can be described by a very short algorithm, while for others like $r =$ `011010111011001`..., we can do no better than to provide a table listing the value of $r(n)$ for each n. Such descriptively incompressible sequences are 'algorithmically random'.

For a formal definition, we need the concept of algorithmic complexity:

$$H(s) \equiv \min_{U(p)=s} |p| \tag{1}$$

where p is a program string used by a universal computer U to produce the sequence s. The complexity $H(s)$ is the minimum length among programs that reproduce s. Program strings must be in a self-delimiting language; i.e. no valid program may be a prefix of another (Chaitin 1987, p. 106). Ordinary computer languages are *prefix-free*, as they are required to have properly balanced 'end' instructions or parentheses. The criterion for algorithmic randomness is $H(s) \approx |s|$, i.e. the minimal algorithm for s must be about equal in length to s itself.

An infinite sequence r is random if the complexity of its initial segment r_n of length n 'does not drop arbitrarily far below n: $\liminf H(r_n) - n > -\infty$' (Chaitin 1987, p. 134) or

$$\exists c \forall n \left[H(r_n) \geq n - c\right] \tag{2}$$

Equivalently, random infinite sequences are the $n \to \infty$ limit of algorithmically random finite sequences. Let the set of algorithmically random sequences of length n be

$$\rho_n = \{s_n | H(s_n) \geq n + O(1)\} \tag{3}$$

where $O(1)$ denotes a function bounded from above and below (Chaitin 1987, p. 111). The set of random sequences is then $\rho = \lim_{n\to\infty} \rho_n$.

We now need a way of describing functions by combining algorithms and randomness. Note that the basis of algorithmic randomness is that for some parts of a sequence, we can do no better than looking up bits in a table. So we define a *literal-free* language, designed to separate table-lookups from the algorithmic structure of a function. All references to literal bit values are done through table-lookups, by a special type of function call. The function u of the previous example will now be defined as:

```
function u(n)
    if (n is odd) then u(n) = ρ()
    else u(n) = ρ(i(n))
end

table ρ
```

`100010011...`

Here, $i(n) = n/2 + 1$, and ρ is a function returning the leading bit in a table with no arguments, or the nth bit if an offset n is provided. The function u defines a sequence `0101011101011111...`

In a literal-free language, actual numbers – the `0`'s and `1`'s – appear only in tables. This has a number of advantages: (i) it emphasizes the structure of table use which goes into algorithmic randomness; (ii) while normal languages can be simulated within a literal-free language, this is more expensive in terms of program size, so minimal programs use table-lookups; (iii) by replacing table functions with ρ denoting the set of random functions, a literal-free language is immediately convertible to a language describing function sets. For convenience, we introduce another condition: our language will have only a single table, which is simply a bit string following the algorithm sector of the program.

Such a language can describe all functions. For example, consider a function b which is an infinite collection of algorithms, all applying to their own part of parameter space; i.e. $b(n) = t_{j(n)}(n)$. This will be described as:

function $b(n)$
    $c_j = j(n)$'th code segment in ρ
    $b(n) = U(c_j, n)$
end

table ρ
    ... [code segment for t_1]... [code segment for t_2]...

Scanning from the beginning of the table ρ, we find an appropriate code segment c_1 for $j = 1$, then find another algorithm in the sequence following where c_1 left off, and so on. If there is no relation whatsoever between the t_j, a *random* table ρ is required. In proving completeness, we will see that we can find descriptions for all functions where (i) the table sector is random, and (ii) the instruction segment preceding the table is finite in length.

We now define our function description scheme, beginning with:

$$S \equiv \{s | s \text{ is an } \textit{infinite} \text{ bit string}\} \tag{4}$$

$$F \equiv \{f | f(n) \in \{\mathtt{0}, \mathtt{1}\}\} \tag{5}$$

$$D : S \mapsto F \quad , \quad D(p) = U(p, \bullet) \tag{6}$$

The description we seek is a map D from program strings to functions, a sort of universal computer. We restrict our attention to total predicate functions since they are straightforwardly equivalent to sequences; this

will not affect the generality of the completeness proof. The language defined by D is:

1. *Literal-free,* as described above. A program $p = p^i + p^\rho$, a concatenation of an instruction segment and a semi-infinite table sector. We require $|p^i| < \infty$.

2. *Prefix-free* in p^i. The instruction segment has a definite end; everything that follows is part of the table.

3. *Syntax-free:* any bit string p is a valid program string with probability 1 (Chaitin 1987, p. 92). This is for convenience only.

Note that program strings are infinite, which means uncomputable functions are describable by D. The program-size cost function C then needs to be modified:

$$C(p) \equiv \min_{D(p_n)=D(p)} n \tag{7}$$

i.e. the cost is the length of the shortest initial segment of the program string sufficient to fully specify the function. In the following, we shall use $|p|$ to stand for $C(p)$, since C is a kind of 'length' for program strings. The complexity is defined as in Equation (1),

$$H(f_n) \equiv \min_{D(p)_n=f_n} |p| \tag{8}$$

where f_n is the initial segment of length n of the sequence defined by function f. $H(f) = H(f_\infty) < \infty$ iff f is a computable function, since it will not require an infinite table to list function values.

Now we can define a recursively enumerable description scheme based on D. We simply label function sets by the finite strings p^i corresponding to the instruction segments. Table references $\rho()$ now designate the set of random functions. With P^i standing for the set of instruction segments:

$$D^* : P^i \mapsto F, \quad D^*(p^i) = \{f | f = D(q) \text{ where } q \in \rho \wedge q^i = p^i\} \tag{9}$$

Proposition 1 *Every function is finitely described by D^*, or*

$$\forall f \in F \; \exists p^i \in P^i \;\; \left(f \in D^*(p^i)\right) \tag{10}$$

This can be proven by demonstrating the truth of another proposition:
Proposition 2 *Every function can be described by an infinite random sequence;*

$$\forall f \in F \; \exists p \in \rho \;\; (f = D(p)) \tag{11}$$

This will be expressed as $F = D(\rho)$ for short.

Proposition 2 implies Proposition 1. Recall that every program p is prefix-free in its initial instruction segment p^i, followed by a semi-infinite table sequence p^ρ. All $|p^i| < \infty$, i.e. p codes for an instruction segment terminated by an '`end`' instruction, or balanced parentheses, or some equivalent device. If $p \in \rho$, it must have an initial segment that terminates in this fashion. Note that it *must* terminate: otherwise there would be a pattern in the sequence (e.g. if '`en`' was coded for, it could not be followed by a '`d`'), which is not present in random sequences by definition. Since all possible finite p^i segments are included in ρ, $F = D(\rho)$ implies Proposition 1.

Now, let us focus on initial segments of length n as in Equation (3). Let $m(f_n)$ be the lowest-cost program reproducing f_n, the initial segment of function f; i.e. $|m(f_n)| = H(f_n)$. $m(f_n)$ has two important properties:

1. $m(f_n)$ *is itself an algorithmically random sequence:* If $H(m)$ were significantly less than $|m|$, there would exist a string s, interpretable as a program for m, which was less costly. In that case, an implementation of the function $D(s)$ would be the minimal program, so that $|m(f_n)| > H(f_n)$. Since the cost of a subroutine for D is bounded,
$$H(m(f_n)) = |m(f_n)| + O(1) \tag{12}$$
In other words, $m(f_n) \in \rho_{|m|}$ as defined in eq. (3).

2. $\forall n\, |m(f_{n+1})| \geq |m(f_n)|$*:* each additional bit either fits the extrapolation of the present minimal program, in which case $|m|$ does not change, or fits a new minimal program which must be costlier than the previous one.

We now use the algorithmic randomness of $m(f_n)$ to show that the minimal program for any function is a random sequence. Consider the limit $n \to \infty$. Since $|m(f_n)|$ is a non-decreasing function of n, it either goes to infinity with n, or converges to a finite value. $\lim_{n\to\infty} |m(f_n)| < \infty$ means the function is computable. This trivially fits Proposition 2.

For the $|m| \to \infty$ case, we have
$$\lim_{n\to\infty} m(f_n) \in \lim_{|m(f_n)|\to\infty} \rho_{|m(f_n)|} \;\Rightarrow\; m(f_\infty) \in \rho \tag{13}$$

The fact that $m(f_\infty)$ is random proves Proposition 2. As $m(f_n)$ grows with n, $|m^i|$ always remains finite. With uncomputable sequences, only the cost of the *random* table sector grows to infinity. The uncomputable part of a function becomes extracted out to a random sector, leaving a finite minimal description m^* displaying the algorithmic structure of the function.

4. A random world

So now we have a completeness theorem along with Gödel's Incompleteness Theorem. Gödelian arguments concern activities such as mathematics, where humans produce results expressible by a finite bit sequence. And in these cases, human behavior *must* be describable as a partly random function. There is no other option. We now need to interpret what such a result means, under the assumption that humans do not have access to any meaningful oracle.

No algorithm can compute more than a finite number of bits in a random sequence. This sort of randomness exists in the very structure of mathematics; e.g. the function telling us whether a given exponential diophantine equation has a finite or infinite number of solutions is random (Chaitin 1987, p. 159). Every nonalgorithmic function is at least partly random in exactly this manner. The halting function h has a minimal description $m^*(h)$ which is partly random and partly algorithmic, for we know many special cases for which we can determine $h(p, n)$. For example, we can prove that certain programs containing no branching instructions will halt, and we know how to detect certain endless loops which ensure p will not halt. But there *must* be randomness in $m^*(h)$, and we can only determine a finite number of the relevant random bits in the best of circumstances. Knowledge of randomness in such cases only gives us a precise description of our ignorance.

Of course, it is still possible that we magically know the precise random sequence needed to compute a meaningful nonalgorithmic function. One way of doing this is if we could somehow directly observe the relevant random sequence. The sequence of measurements of a quantum two-state system prepared a certain way is a random $r \in \rho$. We can 'compute' $r(n)$ simply by performing n measurements. This suggests that if we were capable of observing a Platonic realm of mathematical truths, we might be able to 'compute' meaningful nonalgorithmic functions. Gödelian objections to AI tend to involve claims about the nature of mathematical inspiration; indeed, Penrose advocates mathematical Platonism, as did Gödel himself (Wang 1987, p. 188). But it is clear that Platonism is not required to satisfy the Gödelian intuition about not being rule-bound.

The completeness theorem tells us that if humans are nonalgorithmic, they do in fact rely on a random function. If, following (G2), we also assume we do not have access to meaningful oracles, this means we cannot depend on any particular $r \in \rho$ being available to us: r must be arbitrary. The only relevant fact must be that r is completely haphazard. And if not being rule-bound is *all* that is being claimed for humans (no meaningful oracles or Platonism), then completeness

shows that randomness, and only randomness, provides us with exactly this feature.

Simple illustrations of how a strategy using randomness is not trapped by the rules of any formal system are easy to come by. Our cop-and-robber game is one, so are more complex variations on similar themes that is the province of game theory. Another familiar example of randomness being useful to avoid the limitations of rules is in numerical integration. Monte Carlo integration must fail for some functions if they are evaluated at points determined by an algorithm, e.g. a pseudorandom number generator. If we know the algorithm, we can design a function for which the integration result will be a gross mistake. With true randomness, no such failure can be guaranteed, and long-run convergence is ensured.

A more complex case where randomness may be of use is learning. For example, an algorithm to find out what a function is by sampling its output must fail for certain functions we can construct by virtue of knowing the learning algorithm. Every inferential algorithm must have such blind spots (Angluin and Smith 1987). Here too, we would expect randomness to help, in much the same way as in the cop-and-robber game. Because of pragmatic constraints on what makes a good learning strategy, we would expect that while randomness helps, it is best used in a rather complicated manner. For example, we can randomly generate algorithms, or use random variations from a current algorithm, to ensure we are not locked into any particular strategy. Addressing learning strategies in more detail is beyond the scope of this paper, aside from noting that use of a random number generator is necessary to avoid blind spots.

So we can refine our ideas about human nonalgorithmicity further, in a way favorable to machine intelligence. There are functions like the halting function or the function telling us about diophantine equations, which are at least partly beyond the capabilities of machines. But without Platonic magic, these seem to be intractable for human reasoning as well. What we *can* achieve is the ability not to be restricted by any particular set of rules. This nonalgorithmicity without oracles is available to machines as well.

> (G3) Human intelligence, and the behavior of machines incorporating randomness, is not restricted by any set of rules. There are tasks for which no general algorithm exists, but which can be solved by use of an arbitrary random number generator to ensure we are not locked into any particular algorithm.

Whether (G3) is actually true of humans, and how relevant randomness is to human intelligence, are empirical questions which need detailed consideration. However, our ordinary scientific knowledge about

the world gives us reason to suspect human reasoning has a random aspect.

The possible role of randomness in learning suggests an analogy to Darwinian evolution as a way of constructing new and complex structures from modest beginnings. This process need not be exactly analogous to biological evolution; only the potency of randomness coupled with selective retention of new features is relevant. Contemporary supporters of AI often see an analogy between evolution and intelligence, emphasizing a 'generate-and-test' process (e.g. Dennett 1995). This concept of mechanical thinking fits in well with (G3). It also is applicable to humans. Evolution is a standard argument in favor of mind as a physical process (e.g. Churchland 1988, pp. 20–21); if a process of random variation and selection were instrumental to the operation of the brain, this would come as no surprise.

The Darwinian analogy is a rough one, however. Much of our reasoning, even in coming up with new conjectures, does not depend on a strictly blind process of variation (Thagard 1988, pp. 102–105). All (G3) requires is that randomness be involved at some level in the brain, in order to prevent us being locked into any particular algorithm. It is not helpful in deciding exactly how randomness is used. Just as a highly abstract Turing machine says little about the hardware of a real-world computing device, the possible availability of randomness does not constrain the brain very much. However, we can at least say that randomness is in fact available – the physical world is awash with noise.

An important source of noise for (G3), since we are concerned with strict randomness and not just extremely high complexity, is the randomness in quantum physics. Quantum spookiness has often been thought to be a basis for criticizing machine intelligence, e.g. by providing a peculiar 'unity of consciousness' (Hodgson 1991, pp. 383–387), or giving consciousness a role in deciding experimental outcomes. None of this is required by quantum mechanics (Mulhauser 1995, Stenger 1995), but the belief that the way humans think is deeply connected to fundamental physics persists. (G3) suggests a legitimate role for quantum mechanics in explaining human thought: it is our best reason for believing there is randomness in the world.

For the Gödelian intuition that we are not bound by rules to be true, we need only be able to behave partly randomly. Allowing use of a random number generator is a trivial extension of the notion of mechanical computation. There is randomness in the world, which can be plausibly said to have a role in our thinking. Given all this, there is no more reason to invoke Gödel's Theorem as a criticism of AI. The

notion that we are not rule-bound, far from undermining the possibility of machine intelligence, actually supports it.

Acknowledgements

Thanks are due V.J. Stenger for helpful discussions.

1. See comments following Penrose 1990, contributions to *Psyche* 2, May. 1995; 'http://psyche.cs. monash.edu.au/psyche-index-v2-l.html#som', and Grush and Churchland 1995.

2. A probabilistic Turing machine is a Turing machine with the extra ability to toss a fair coin. Such machines have been studied for their theoretical interest; they can be quite powerful in some instances (Karpinski and Verbeek 1996).

References

Angluin, D., and C.H. Smith. (1987) 'Inductive Inference', in S. Shapiro (ed.,) *Encyclopedia of Artificial Intelligence*, Vol. l, New York, NY: Wiley, pp. 409–418.
Berger, J.0. (1980) *Statistical Decision Theory*, New York, NY: Springer Verlag.
Chaitin, G.J. (1987) *Algorithmic Information Theory*, Cambridge, UK: Cambridge University Press.
Churchland, P.M. (1988) *Matter and Consciousness* (revised ed.), Cambridge, MA: MIT Press.
Cleland, C.E. (1993) 'Is the Church-Turing Thesis True?', *Minds and Machines*, 3, pp. 283–312.
Dennett, D.C. (1972), Review of J.R. Lucas, *The Freedom of The Will*, in *Journal of Philosophy*, 69, pp. 527–531.
Dennett, D.C. (1995), *Darwin's Dangerous Idea*, New York, NY: Simon and Schuster.
Fredkin, E. (1990) 'Digital Mechanics', *Physica D* 45, pp. 254–270.
Grush, R. and P.S. Churchland. (1995) 'Gaps in Penrose's Toilings', *Journal of Consciousness Studies*, 2, pp. 10–29.
Hodgson, D. (1991), *The Mind Matters*, Oxford, UK: Clarendon Press.
Karpinski, M. and R. Verbeek. (1996), 'On Randomized versus Deterministic Computation', *Theoretical Computer Science*, 154, pp. 23–39.
Lucas, J.R. (1961), 'Minds, Machines, and Gödel', *Philosophy*, 36, pp.112–127.
Ludwig, K. (1995), 'Why the Difference Between Quantum and Classical Physics is Irrelevant to the Mind/Body Problem', *Psyche*, 2, p.16.
Mulhauser, G.R. (1995) 'Materialism and the "Problem" of Quantum Measurement', *Minds and Machines*, 5, pp. 207–217.
Penrose, R. (1989), *The Emperor's New Mind*, Oxford, UK: Oxford University Press.
Penrose, R. (1990) 'The Nonalgorithmic Mind', *Behavioral and Brain Sciences*, 13, pp. 692–705.

Penrose, R. (1994) *Shadows of the Mind*, Oxford, UK: Oxford University Press.
Stenger, V.J. (1995), *The Unconscious Quantum*, Buffalo, NY: Prometheus.
Thagard, P. (1988), *Computational Philosophy of Science*, Cambridge, MA: The MIT Press.
Wang, H. (1987), *Reflections on Kurt Gödel*, Cambridge, MA: MIT Press.