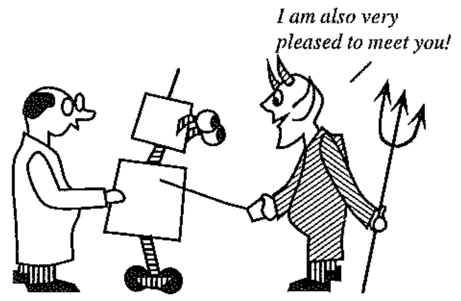
6

The Demons of Computability



1. Introduction

Formal learning theory has been invented and developed to a large extent by experts in the theory of computability. This is no accident. The theory of computability suggests a strong analogy between the situation of a computer working on a purely formal problem and that of an empirical scientist working on a purely empirical problem, and learning theorists have instinctively followed this analogy when thinking about inductive inference. I will argue, in fact, that uncomputability and empirical underdetermination have the same, underlying logical structure, so that it is difficult to dismiss the demons of induction while acknowledging the significance of uncomputability. On the one hand, this analogy counters the tendency in traditional epistemology to study ideal norms for agents free of all computational limitations. On the other hand, it raises questions about the motivation behind the sharp divergence of philosophical attitudes toward formal and empirical inquiry. The purpose of this chapter is to explore the analogy between computability and inductive reliability, both to illustrate the strong ties between computability theory and skeptical methodology and to introduce some computational tools that will be vital to the discussion of computationally bounded inductive inference in chapter 7.

2. Church Meets Hume

The problem of induction begins with the assumption that the human perspective on the universe is spatiotemporally bounded. In traditional epistemology, at most finitely many experimental trials can be observed by a given time. This locality of observation is captured by allowing the scientist

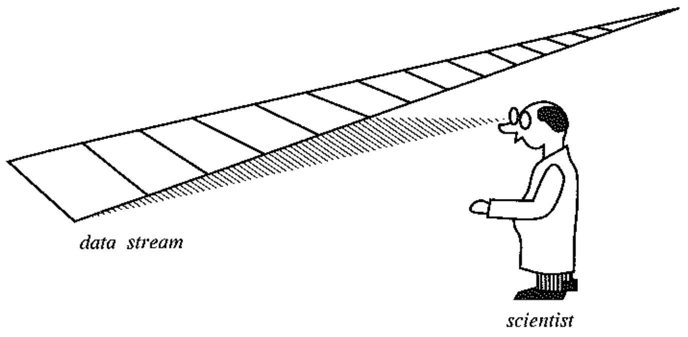


Figure 6.1

access only to a finite initial segment of a data stream at a given time (*Fig. 6.1*).

Alan Turing reasoned from a philosophical explication of the limitations of a human following an explicit procedure to a formal model of computability.[1] A *Turing machine* consists of a *read-write head* scanning a potentially infinite *tape*, in the sense that another square of tape is always provided when the machine uses up the tape at hand. The read-write head is like an experimental scientist according to the definition given in chapter 2. The experimental acts available to the read-write head are to move right on the tape, to move left on the tape, or to write something down on the square of tape currently scanned. Its observation at a given stage of inquiry is the symbol written on the tape square scanned at that stage. Finally, its current experimental act is determined entirely by the current datum and its current *state*, of which there are at most finitely many. We may think of these states as the possible configurations of a finite memory store. Just as a scientist sees only a finite, initial segment of his data by any given time, the read-write head sees only a finite chunk of the tape by any given time (*Fig. 6.2*).

Church's thesis is the claim that a Turing machine can do whatever a person (called a *computor*) following an explicit, step-by-step procedure can do. Church's thesis is nontrivial, because Turing computability is a mathematically precise notion, whereas following an explicit procedure is not. Turing's account of the limitations of the human computor provides an informal argument for this thesis. A major limitation figuring into the argument is that a human doing a computation can only see or alter at most a finite number of discrete jottings at once.[2] The human computor is transformed into the Turing read-write head, and his bounded range of observation is transformed into the fact that the read-write head can scan at most one square at a time. So both the classical problem of induction and the modern theory of computability are founded on the idea of a bounded perspective on the infinite.

[1] Turing (1936).
[2] Turing (1936). For a detailed discussion of Turing's analysis of computability and its relevance to Church's thesis, cf. Sieg (1994).

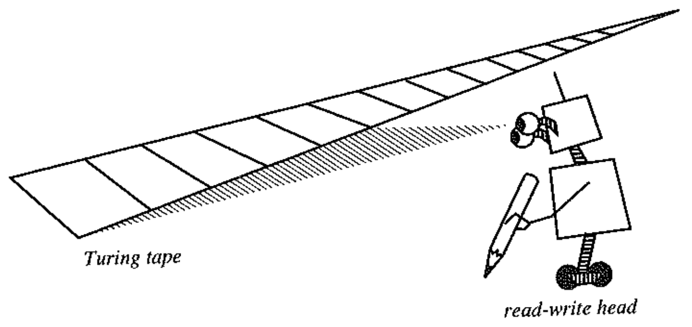


Figure 6.2

The bounded perspective of the read-write head leads to the key computational metaphor of *search*. In fact, search is what distinguishes the *recursive* functions from the *primitive recursive* functions. The latter can always be computed with an a priori bound on how long the read-write head must wait before finding a relevant datum (compare to empirical decidability by time n), whereas the recursive functions may require an unbounded search for their computation (compare to empirical decidability with certainty). In search, it is natural to think of the read-write head as primed to wait for some construct to appear on the tape, without ever knowing whether it will appear. Even though the initial input to the Turing machine, together with the machine's program, mathematically determines what patterns will arise, the read-write head doesn't always "know" what it will "see" in the future. It has to generate the sequence of patterns and inspect them, just the way an empirical scientist looks at increasing data about his unknown world. The sequence of patterns seen by the read-write head on a given input may be thought of as a kind of data stream, except that the data concerns a mathematical structure rather than the physical world.

Despite the similarities between the situation of the read-write head and that of an experimental scientist, the theory of computability has pursued a very different course from that adopted by most scientific methodologists. Instead of attaching degrees of belief to various hypotheses about what will eventually appear on the tape, or about what the correct solution to a computational problem might be, the machine is expected to find the correct answer. In other words, the theory of computability is a logical reliabilist arena for the assessment of computer programs. Indeed, the criteria of reliability assumed in the theory of computability are exactly parallel to those applied to inductive methods in the preceding chapters.

3. Programs as Reliable Methods

To speak of a Turing machine as computing outputs from inputs, we have to say what it is to give the machine an input and what it is for the machine to

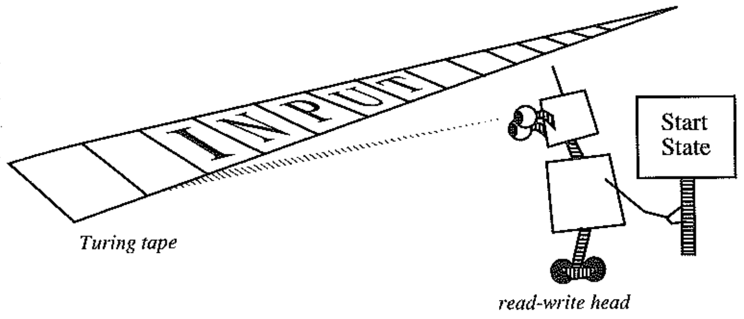


Figure 6.3

produce an output. It turns out not to matter much which conventions we choose, so long as we stick with them. To input n, put the read-write head into its designated start state and then write $n + 1$ 1s to the right of the head on an otherwise blank tape (*Fig. 6.3*).

A Turing machine outputs n when the read-write head goes into its designated halting state and an uninterrupted sequence of n 1s appears to the left of the read-write head on an otherwise blank tape (*Fig. 6.4*).

Just as we could interpret one and the same inductive method as trying to succeed in various different ways (e.g., verification in the limit, decision with certainty), the theory of computability can interpret one and the same Turing machine as converging to an output in correspondingly different ways. Turing machine M can be viewed as computing a function ϕ from inputs to outputs, called a *partial recursive function*. If M fails to make an output on some input x, then ϕ is undefined for x and we write $\phi(x)\uparrow$. Otherwise, we write $\phi(x)\downarrow$. If ϕ is total, then ϕ is called a *recursive function*. Let $\mathcal{PR}$ denote the class of all partial recursive functions and let $\mathcal{R}$ denote the set of all (total) recursive functions. A *decision procedure* is a Turing machine that computes the *characteristic function* of some relation on the natural numbers. That is, a decision procedure outputs 1 if a given sequence of numbers is in the relation, and outputs 0 otherwise. A *positive test* is a Turing machine that returns 1 on a given input if and only if the input is in a given set. On objects not

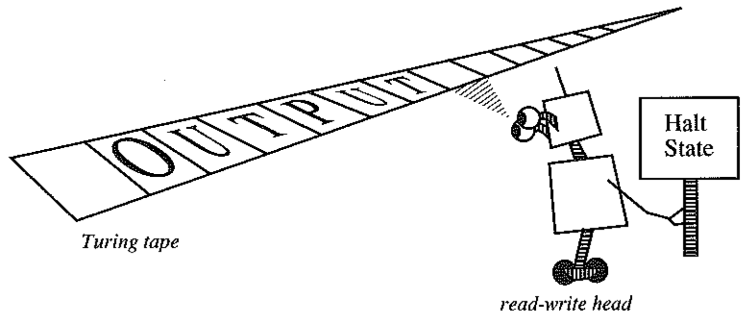


Figure 6.4

in the set, the machine is entitled to produce any output other than 1 and is also free to go into an infinite loop. A relation that has a decision procedure is said to be *recursive*. A relation that has a positive test is said to be *recursively enumerable*, or *r.e.* for short. If a relation's complement is r.e., then the relation is said to be *co-r.e.* Thus, a co-r.e. relation has a *negative test* (just reverse 1s and 0s in the positive test for its complement).

An r.e. relation may be thought of as a formal proposition that can be verified with certainty by a Turing machine, and a co-r.e. relation corresponds to a proposition that is refutable with certainty. A recursive relation can then be thought of as a hypothesis that can be decided with one mind change regardless of first conjecture. Indeed, all of the criteria of inductive success introduced in chapter 3 can be adopted as notions of computational success. Consider a machine that produces an infinite sequence of 1s and 0s in the limit after it is started with an input number n. Then define:

M is a limiting positive test for S
$\Leftrightarrow \forall n \in \omega$, $n \in S \Leftrightarrow$ *the output stream of $M[n]$ stabilizes to* 1.

M is a limiting negative test for S
$\Leftrightarrow \forall n \in \omega$, $n \notin S \Leftrightarrow$ *the output stream of $M[n]$ stabilizes to* 0.

M is a limiting decision procedure of S
$\Leftrightarrow$ *M is both a limiting positive test and a limiting negative test for S.*

S is limiting r.e. $\Leftrightarrow$ *S has a limiting positive test.*

S is limiting co-r.e. $\Leftrightarrow$ *$\bar{S}$ is limiting r.e.* [$\Leftrightarrow$*S has a limiting negative test*].

S is limiting recursive $\Leftrightarrow$ *S has a limiting decision procedure.*

S is an n-trial predicate $\Leftrightarrow$ *S has a limiting decision procedure that changes its mind at most n times on each input.*

The terms *limiting recursive* and *limiting r.e.* are taken from E. Mark Gold.[3] H. Putnam referred to the former as *trial and error predicates*.[4] Putnam also introduced the notion of n-trial predicates.[5] The limiting recursive relations are analogous to hypotheses decidable in the limit. The limiting r.e. relations are analogous to hypotheses verifiable in the limit. And n-trial predicates are analogous to hypotheses decidable with n mind changes. As in the empirical case, we can usefully refine the notion by specifying the first conjecture:

S is an n-trial predicate starting with 1
$\Leftrightarrow$ *S has a limiting decision procedure that changes its mind at most n times on each input $n \in \omega$, starting with* 1.

[3] Gold (1965).
[4] Putnam (1965).
[5] Putnam (1965).

One might weaken the demands on the machine further, and require only that it produce a sequence of code numbers for rationals that gradually approaches 1 if and only if $n \in S$.[6] Sets having such machines may be called *gradually r.e.*, and sets having gradual refutation procedures may be called *gradually co-r.e.*, while sets having both may be called *gradually recursive*.

4. The Arithmetical Hierarchy

All these notions of success can be located in a hierarchical setting entirely parallel to the finite Borel hierarchy introduced in chapter 4. Much early work in the theory of computability concerns the *arithmetical* or *Kleene hierarchy*. Where R is a k-ary relation on ω, define:

$R \in \Sigma_0^A \Leftrightarrow$ *R is recursive.*

$R \in \Sigma_{n+1}^A \Leftrightarrow$ *there is some $k+1$-ary $R' \in \Sigma_n^A$ such that for each $n_1, \ldots,$ $n_k \in \omega$, $R(n_1, \ldots, n_k) \Leftrightarrow \exists n_{k+1}$ such that $\neg R'(n_1, \ldots, n_k, n_{k+1})$.*

As usual, the dual and ambiguous classes are defined in terms of Σ_n^A.

$R \in \Pi_n^A \Leftrightarrow \bar{R} \in \Sigma_n^A$.

$R \in \Delta_n^A \Leftrightarrow R \in \Sigma_n^A$ & $R \in \Pi_n^A$.

Just as the Borel hierarchy starts out with clopen sets (sets of data streams decidable by an ideal scientist on the basis of data), the arithmetical hierarchy starts out with recursive sets (sets of numbers decidable by a computer). Just as Σ_1^B sets are the open sets, which are verifiable with certainty by an ideal scientist, Σ_1^A sets are r.e. sets, which are verifiable with certainty by a computer. The correspondence goes all the way up to ω (*Fig. 6.5*).[7] By way of illustration, let's establish the limiting r.e. case.

Proposition 6.1 (Gold 1965, Putnam 1965)

S is limiting r.e. $\Leftrightarrow S \in \Sigma_2^A$.

Proof: ($\Leftarrow$) Suppose that $S \in \Sigma_2^A$. Then S is of the form $\{x: \exists n\, \forall m\, R(x, n, m)\}$, where R is a recursive relation. We construct a machine M that proceeds as follows. Let M_R be a decision procedure for R. On input x, M sequentially outputs $M_R(x, 1, 1)$, $M_R(x, 1, 2), \ldots$ until for some n, $M_R(x, 1, n) = 0$. If a zero is received, then M outputs 0 and then outputs $M_R(x, 2, 1)$, $M_R(x, 2, 2), \ldots$ until

[6] This idea is proposed and developed in Hàjek (1978).
[7] Recall that the Borel hierarchy does not stop at ω. The arithmetical hierarchy does stop at ω.

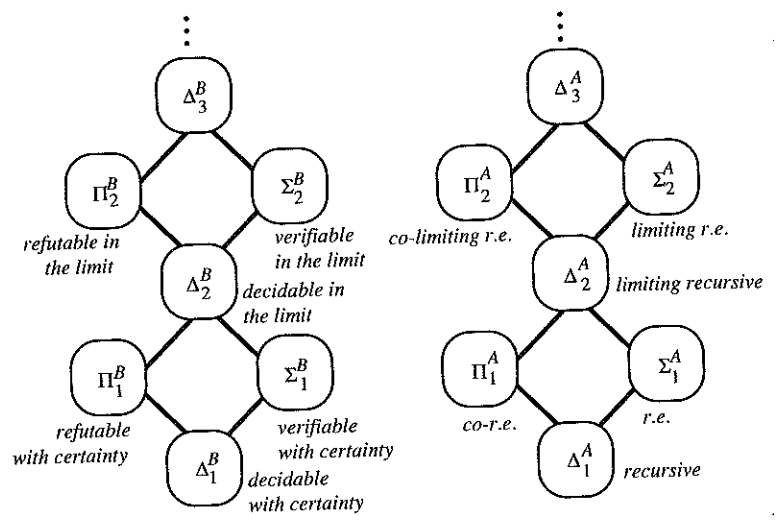


Figure 6.5

another zero is received, and so forth. All of this is effective, so by Church's thesis, it can be implemented as a Turing machine M. It is easy to verify that M is a limiting positive test for S.

($\Rightarrow$) Suppose that S is limiting r.e. Then there is some machine M such that

$$x \in S \Leftrightarrow \exists n\ \forall m > n \textit{ the mth output of } M[x] \textit{ is defined and equal to } 1.$$

Since M is assumed to output an infinite sequence, the predicate

$$\textit{the mth output of } M[x] \textit{ is defined and equal to } 1$$

is recursive. Hence, S has form $\exists n\ \forall m\ R[x, n, m]$, where R is recursive. Thus, $S \in \Sigma_2^A$. ■

This is all parallel to the characterizations of decidability and verifiability in the limit presented in chapter 4. Just as fans provide basic empirical tests that can be built up using quantifiers into ever more subtle empirical hypotheses, recursive relations provide basic formal tests that can be built up using quantifiers into ever more subtle computational problems.

5. Uncomputability and Diagonalization

It might be objected at this point that it is quite appropriate for the theory of computability to insist on reliability because computation is just deduction and deduction can be guaranteed to yield certainty. Inductive inquiry, on the other hand, concerns answers we can never be sure about, and hence demands that

we introduce notions of justified belief, evidential support, and probability. The truth is quite otherwise. The whole force of the theory of computability is to show that a bounded agent like the Turing read-write head cannot be guaranteed to arrive at certainty concerning most formal questions. The reason for this limitation is entirely parallel to the reason for inductive skepticism, namely, a bounded perspective on an unbounded data stream. To illustrate this point, let us consider how uncomputability arguments can be recast as classical demonic arguments for the local underdetermination of empirical hypotheses.

It turns out that the easiest problems to prove intractable are those that have a kind of self-referential character based on the effective assignment of code numbers to Turing machines. Assume some such fixed assignment of numbers to Turing machines. Let W_i denote the set of all inputs on which machine M_i halts. W_i is an r.e. set, since we can trivially modify M_i to output 1 whenever M_i halts. Moreover, each r.e. set S is some set W_i, since the machine M_i can be produced by modifying the positive test for S to go into an infinite loop whenever it halts with an output not equal to 1.

The *halting problem* K is defined as follows:

$$K = \{i: i \in W_i\}.$$

That is, the halting problem contains all indices i such that the machine with index i halts when fed its own index i as an input. It is easy to see that K is r.e. A machine can simulate the operation of M_i on input i until M_i is observed to drop into its halting state. At that point, the simulating machine outputs 1. If the halt state is never reached, the simulating machine waits forever. The simulation just constructed is a computable positive test for K. (Notice the close analogy between the strategy of the machine and that of an empirical scientist waiting for some crucial datum to verify an empirical hypothesis.)

We will now see that $\bar{K}$ is not r.e. by means of a simple *diagonal argument.*

Proposition 6.2

$\bar{K} \notin \Sigma_1^A$.

Proof: We can form an infinite table T as follows (*Fig. 6.6*):

$$T(x, y) = \begin{cases} 1 & \textit{if} \quad y \in W_x \\ 0 & \textit{otherwise.} \end{cases}$$

We can think of an r.e. set S as uniquely represented by a sequence of 0s and 1s, with 1 in position y if $y \in S$ and 0 in position y otherwise. So each row in the table represents the characteristic function of some r.e. set. The important point is this: since we have seen that each r.e. set is some set W_i, each r.e. set is some row in the table.

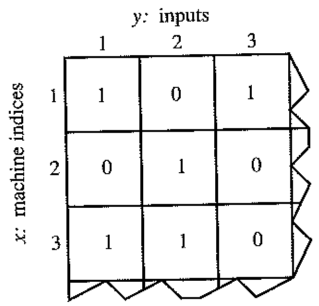


Figure 6.6

Now let us consider whether $\bar{K}$ is represented by some row in the table. If not, then $\bar{K}$ is not r.e. Recall that $K = \{x: x \in W_x\}$. The question whether $i \in W_i$ is answered by looking at table entry $T_{i,i}$. Hence, K is represented by the diagonal sequence of the table. Therefore, $\bar{K}$ is represented by the result of reversing 1s and 0s along the diagonal of the table to form the *counterdiagonal sequence* of the table (*Fig. 6.7*).

But the counterdiagonal sequence differs from each row of the table in at least one place (namely, on the diagonal). Therefore, $\bar{K}$ is represented by no row in the table, and hence is not r.e. ■

The sense in which the preceding proof is a diagonal argument is clear. It proceeds by depicting $\bar{K}$ as the counterdiagonal of a table whose rows include the characteristic functions of all r.e. sets.

6. The Demons of Uncomputability

The diagonal argument rehearsed in the preceding section doesn't look like a demonic argument, since there is no demon who feeds misleading data to the

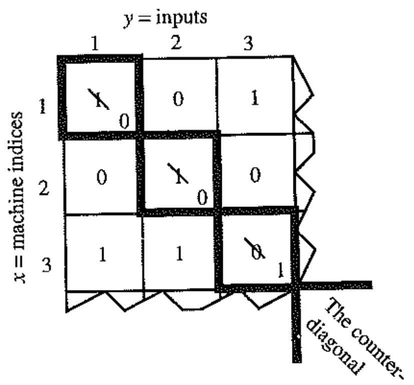
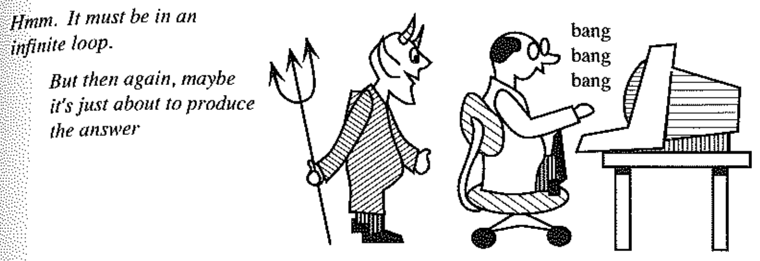


Figure 6.7



Hmm. It must be in an infinite loop.
But then again, maybe it's just about to produce the answer
bang
bang
bang

Figure 6.8

Turing read-write head. Nonetheless, there remains a strongly intuitive feeling that any would-be decision procedure is foiled for inductive reasons: no matter how long the decision procedure waits for machine M_i to halt on input i before concluding that the computation $M_i[i]$ is in an infinite loop, the computation may halt at the very next moment. Anyone who encounters a bug in a slow-running program knows the feeling. One doesn't know whether to turn off the machine and start over (wasting all the time spent up until now if the program was just about ready to make an output) or to continue to wait (which may take forever) (*Fig. 6.8*).

There is one important disanalogy between a computer and an inductive method: the computer receives no infinite data stream and an inductive method does. So whereas an inductive demon can feed misleading data to an inductive method in response to its conjectures thus far, a computational demon cannot. But a computational demon can still watch what machine M_i does through time, waiting, for example, for M_i to declare its certainty by returning 1 on a given input. Moreover, the demon itself can be a computer program, since it is no problem for one computer program to simulate another to see what it does on a given input. The plot thickens when the index provided to M_i in the demon's simulation is the demon's *own* index. Then when M_i is observed to become certain (in the demon's simulation) that the demon's index has a given property, the demon can make an output so that his own program fails to have the property M_i is certain that it has!

In particular, we can use such a demonic argument to show that $\bar{K}$ is not r.e. Suppose that M_i aspires to be a positive test for $\bar{K}$. The demonic program M_{d_i} works like this. On input y, it simply ignores y altogether and proceeds to simulate M_i, step by step, on input d_i (i.e., the demon's own index). The demon continues this simulation until the computation $M_i[d_i]$ returns 1, at which time the demon returns 1 as well. Otherwise, the demon continues to wait for $M_i[d_i]$ to return 1, and hence makes no output on y (*Fig. 6.9*).

Thus we have that $d_i \in W_{d_i} \Leftrightarrow M_i[d_i]$ returns 1, so M_i is not a positive test for $\bar{K}$. Assuming that such a d_i exists for each i, no M_i is a positive test for $\bar{K}$ and hence $\bar{K}$ is not r.e. The index d_i, if it exists, is a "computational disguise" for an inductive demon dedicated to the befuddlement of machine M_i (*Fig. 6.10*).

But it is not a trivial question whether such a demonic index d_i exists for each machine M_i. It seems that we have to know what d_i is before we can write

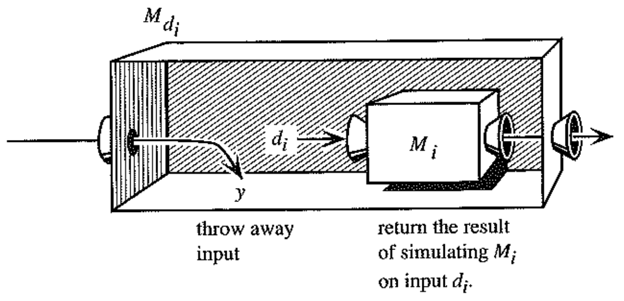


Figure 6.9

down the demonic procedure M_{d_i}, because M_{d_i} feeds d_i to M_i. But we cannot know what d_i is until after we write down the procedure M_{d_i}, since d_i is the code number of that procedure. So the question remains whether the inductive demon can always don a computational disguise d_i appropriate for a given machine M_i.

The self-referential miracle required for the demon's disguise is effected by the *Kleene recursion theorem.* Before proving it, we will need a pair of simple results.

Proposition 6.3 The *s-m-n* theorem

Suppose $\psi(x, y)$ is a partial recursive function. Then there is a total recursive function s such that for each $x, y \in \omega$,

$$\phi_{s(x)}(y) = \psi(x, y).$$

Proof: Let $M[x, y]$ be a program for ψ. To compute $s(a)$, take the program M, stick in input y, and return the code number of the resulting program $M[a, y]$, which now only accepts input y. ∎

Proposition 6.4 The universal machine theorem

There is a $u \in \omega$ such that for each $i, x \in \omega$, $\phi_u(i, x)$ $= \phi_i(x)$ if $\phi_i(x)$ is defined and is undefined otherwise.

Proof: u is the index of a program that decodes i and that simulates the resulting program on input x, returning whatever output the simulated program yields, if any. ∎

Now we may proceed to the main result.

Proposition 6.5 The Kleene recursion theorem[8]

For each total recursive f, there is an index i such that $\phi_{f(i)} = \phi_i$.

[8] The nice, tabular proof that follows is adapted from Cutland (1986).

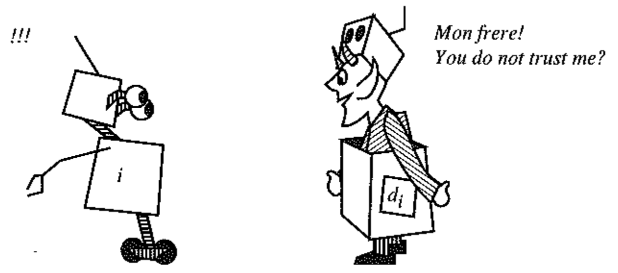


Figure 6.10

Proof: Consider a table filled with partial recursive functions so that at position x, y, we have the function $\phi_{\phi_x(y)}$, where we stipulate that $\phi_{\phi_x(y)}$ denotes the everywhere undefined function if $\phi_x(y)$ is undefined. Consider the table's diagonal sequence of functions:

$$D = (\phi_{\phi_0(0)}, \phi_{\phi_1(1)}, \phi_{\phi_2(2)}, \ldots).$$

Since $h(x) = \phi_x(x)$ is partial recursive, there is some i such that $h = \phi_i$, so D is also the ith row in the table (*Fig. 6.11*).

Let f be a total recursive function. Let

$$D^* = (\phi_{f(\phi_0(0))}, \phi_{f(\phi_1(1))}, \phi_{f(\phi_2(2))}, \ldots).$$

Then D^* is a row of the table, since $h'(x) = f(\phi_x(x))$ is computable and hence is some ϕ_k. If ϕ_k is not total, we can choose a total k' that gives rise to the very same row by the following lemma, so that we may assume that ϕ_k is total.

Lemma

If ϕ_k is not total, we may replace k with j so that $\phi_j(x)$ is an index for the everywhere undefined function whenever $\phi_k(x)$ is undefined and $\phi_j(x) = \phi_k(x)$ otherwise, so that the jth row is identical to D^*.

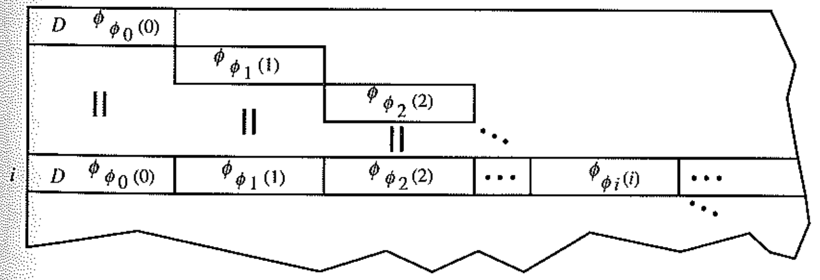


Figure 6.11

Proof of lemma: Define: $\psi(x, y) = \phi_u(\phi_k(x), y)$, where u is the universal index guaranteed by the universal machine theorem (proposition 6.4). $\psi(x, y)$ is partial recursive and amounts to the result of simulating index $\phi_k(x)$ on input y if $\phi_k(x)$ is defined, and is undefined otherwise. So we may apply the *s-m-n* theorem (proposition 6.3) to yield a total recursive s such that $\phi_{s(x)}(y) = \psi(x, y)$. Let j be an index for s. ■

Returning to the proof of the proposition, observe that $\phi_{f(\phi_k(k))}$, the kth entry of D^*, is on the diagonal because D^* is row k of the table. Since row i is identical to the diagonal sequence D, the kth entry of D is identical to the on-diagonal entry in D^*. Thus

$$\phi_{f(\phi_k(k))} = \phi_{\phi_k(k)}.$$

This is immediate from *Figure 6.12*. But by the lemma, $\phi_k(k)$ is defined since ϕ_k is total. Thus, $\phi_k(k)$ is the index promised in the theorem. ■

How does this theorem yield the self-referential disguise d_i for the demon's program for fooling M_i? Consider a procedure that on inputs x, y simulates the computation of M_i on input x. When the procedure observes that M_i returns 1 on input x, it returns 0 for x, y. Otherwise, it continues to wait for M_i to return 1 for x. The procedure's behavior is the following partial recursive function:

$$\psi(x, y) = \begin{cases} 0 & \textit{if } M_i \textit{ returns } 1 \textit{ on input } x \\ \uparrow & \textit{otherwise.} \end{cases}$$

By proposition 6.3, there is a total recursive s such that for all x, y, $\phi_{s(x)}(y) = \psi(x, y)$. Now apply the recursion theorem to obtain an index d_i such that

$$\phi_{s(d_i)} = \phi_{d_i}.$$

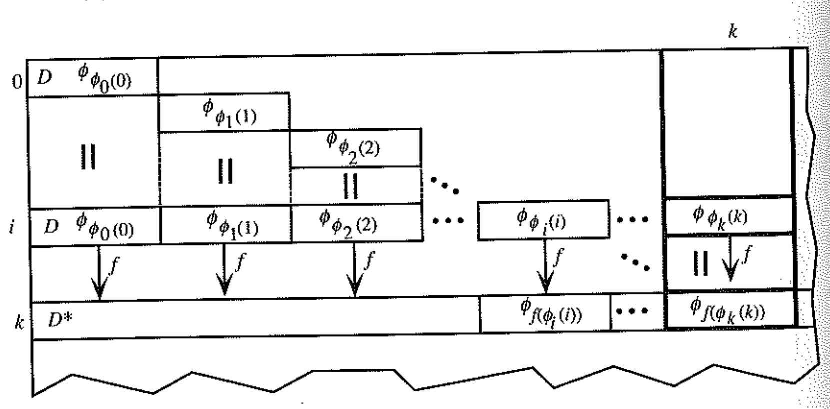


Figure 6.12

The self-referential magic has occurred, for now we have

$$\phi_{d_i}(y) = \begin{cases} 0 & \textit{if } M_i \textit{ returns } 1 \textit{ on input } d_i \\ \uparrow & \textit{otherwise.} \end{cases}$$

One can also use the recursion theorem to prove that a problem is not limiting r.e. in a fashion very similar to the limiting skeptical arguments of chapter 3. Consider the problem $T = \{i\colon \phi_i \textit{ is total}\}$. Let M_i be a Turing program that would like to be a limiting positive test for T. If M_i fails to produce an infinite sequence of outputs on some input, then it fails to be a limiting positive test in any case, so suppose that M_i always does so. On inputs x, y, the demonic procedure feeds x to M_i and waits until there are at least y stages when the current output is less than 1. Then the demon returns 0. The procedure computes the function:

$$\psi(x, y) = \begin{cases} 0 & \textit{if there are at least } y \textit{ stages when } M_i[x]\textit{'s current output is not } 1 \\ \uparrow & \textit{otherwise.} \end{cases}$$

Applying propositions 6.3 and 6.4 just as before yields an index d_i such that

$$\phi_{d_i}(y) = \begin{cases} 0 & \textit{if there are at least } y \textit{ stages when } M_i[d_i]\textit{'s current output is not } 1 \\ \uparrow & \textit{otherwise.} \end{cases}$$

This function is total if M_i fails to stabilize to 1 on k and has finite domain otherwise, so M_i is not a limiting positive test for T. Since i is an arbitrary machine that outputs an infinite sequence of outputs for each given input, there is no limiting positive test for T.

The same demonic technique can be used to prove that broad classes of problems are nonrecursive or non-r.e. Let $\mathcal{A} \subseteq \mathcal{PR}$ (recall that $\mathcal{PR}$ is the class of all partial recursive functions). Let *index*($\mathcal{A}$) denote the set of all indices for functions in $\mathcal{A}$. That is,

$$index(\mathcal{A}) = \{i\colon \phi_i \in \mathcal{A}\}.$$

Let $S \subseteq \omega$. S is an *index set* just in case for some $\mathcal{A} \subseteq \mathcal{PR}$, $S = index(\mathcal{A})$. This amounts to the requirement that no partial recursive function has one index in S and another out of S. For example, the halting problem K is not an index set. To see why this is so, we note that every partial recursive function has infinitely many distinct indices (take a program M_i for ϕ and add arbitrarily many do-nothing clauses to M_i). But using the Kleene recursion theorem, we can construct a partial recursive function ϕ_n that halts only on n, so that $n \in K$

but no other index of ϕ_n is in K. The construction is as follows. The following function is partial recursive by Church's thesis:

$$\psi(x, y) = \begin{cases} 0 & \textit{if } y = x \\ \uparrow & \textit{otherwise.} \end{cases}$$

Then by the *s-m-n* theorem there is a total recursive function s such that

$$\phi_{s(x)}(y) = \begin{cases} 0 & \textit{if } y = x \\ \uparrow & \textit{otherwise.} \end{cases}$$

By the Kleene recursion theorem, there is an n such that

$$\phi_n(y) = \begin{cases} 0 & \textit{if } y = n \\ \uparrow & \textit{otherwise.} \end{cases}$$

On the other hand, the halting problem restricted to a fixed input n *is* an index set:

$$K_n = \{x : n \in W_x\}.$$

So is the set $I = \{n : W_n \text{ is infinite}\}$ which contains all indices of functions with infinite domains or the set T of all indices of total recursive functions. We can now show by a demonic argument that only trivial index sets are recursive.

Proposition 6.6 Rice's theorem

Let S be an index set. Then S is recursive $\Leftrightarrow S = \varnothing$ or $S = \omega$.

Proof: Suppose that for some $\mathcal{A}$, $S = \mathit{index}(\mathcal{A})$. Suppose, further, that $S \neq \varnothing$ and $S \neq \omega$. The everywhere undefined function $\varnothing$ is either in $\mathcal{A}$ or it is not. Let M_i be an arbitrary Turing machine.

Case 1: Suppose that the everywhere undefined function $\varnothing$ is in $\mathcal{A}$. Then since $S \neq \omega$, there is some $\psi \notin \mathcal{A}$ such that $\psi \neq \varnothing$. Intuitively, the demonic index d_i for M_i watches what M_i does on input d_i and produces output $\psi(y)$ as soon as M_i becomes sure that d_i is in S so that d_i is an index for $\psi \notin \mathcal{A}$ and M_i is wrong. Otherwise, d_i continues to stall forever so that d_i is an index for $\varnothing \in \mathcal{A}$ and M_i is again wrong for failing to recognize that $d_i \in S$. As in the case of the halting problem, we use the Kleene recursion theorem to construct such an index d_i. Define:

$$\delta(x, y) = \begin{cases} \psi(y) & \textit{if } M_i[x] \textit{ halts with } 1 \\ \mathcal{K} & \textit{otherwise.} \end{cases}$$

To compute this function, the demon need simply simulate the computation of

M_i on input x and pass along the result of simulating $\psi(y)$ if $M_i[x]$ returns 1. Applying the *s-m-n* theorem, we have a total recursive function s such that

$$\phi_{s(x)}(y) = \begin{cases} \psi(y) & \textit{if } M_i[x] \textit{ halts with } 1 \\ \uparrow & \textit{otherwise.} \end{cases}$$

By the Kleene recursion theorem, there is a d_i such that

$$\phi_{d_i}(y) = \begin{cases} \psi(y) & \textit{if } M_i[d_i] \textit{ halts with } 1 \\ \uparrow & \textit{otherwise.} \end{cases}$$

Thus, we have that $M_i(d_i)$ halts with $1 \Leftrightarrow d_i \notin S$, so M_i is not a decision procedure for S.

Case 2, in which the undefined function $\varnothing$ is *not* in $\mathcal{A}$ proceeds in a similar manner, except that 1 is replaced with 0 in the definition of δ. ■

We may also show by a demonic argument that a wide range of index sets are not r.e.

Proposition 6.7 The Rice-Shapiro theorem

$\forall \mathcal{A} \subseteq \mathcal{PR}$, if $\mathit{index}(\mathcal{A})$ is r.e.
then $\forall \phi \in \mathcal{PR}$, $\phi \in \mathcal{A} \Leftrightarrow \exists$ finite $\theta \in \mathcal{A}$ such that $\theta \subseteq \phi$.

Proof: Suppose that the consequent of the proposition is false of some $\phi \in \mathcal{PR}$. Let M_i be an arbitrary Turing machine. Case 1: Suppose first that the ($\Rightarrow$) side of the biconditional is false. Then $\phi \in \mathcal{A}$ but for each finite $\theta \subseteq \phi$, $\theta \notin \mathcal{A}$. Then to fool M_i, a demonic index d_i ought to proceed (by watching $M_i(d_i)$) so that d_i is an index for some finite $\theta \subseteq \phi$ if $M_i(d_i)$ halts (i.e., becomes sure that d_i is in S) and so that d_i is an index for ϕ otherwise. This can be arranged as follows:

$$\psi(x, y) = \begin{cases} \phi(y) & \textit{if } M_i(x)\uparrow \textit{ within } y \textit{ steps} \\ \uparrow & \textit{otherwise.} \end{cases}$$

This function is computable in light of Church's thesis since we can effectively simulate $M_i(x)$, counting each step as we go until y steps are used and then check whether M_i produces an output by that time. If not, we simulate $\phi(y)$ and pass along the output, if any. If so, we go into an infinite loop. Applying the *s-m-n* theorem and the recursion theorem in the usual way yields

$$\phi_{d_i}(y) = \begin{cases} \phi(y) & \textit{if } M_i(d_i)\uparrow \textit{ within } y \textit{ steps} \\ \uparrow & \textit{otherwise.} \end{cases}$$

Hence, $M_i(d_i)\downarrow \Rightarrow d_i$ is an index for some finite initial segment θ of $\phi \Rightarrow d_i \notin S$, and $M_i(d_i)\uparrow \Rightarrow d_i$ is an index for $\phi \Rightarrow d_i \in S$, so $S \neq W_i$.

In case 2, the ($\Leftarrow$) side of the biconditional is false, so we have that there is a $\phi \in \mathcal{PR} - \mathcal{A}$ and there is a finite $\theta \subseteq \phi$ such that $\theta \in \mathcal{A}$. In this case, we want the demonic index d_i to be for ϕ if $M_i(d_i)$ becomes sure that d_i is in S and to be for θ otherwise. Define:

$$\psi(x, y) = \begin{cases} \phi(y) & \textit{if } M_i(x)\downarrow \textit{ or } y \in dom(\theta) \\ \uparrow & \textit{otherwise.} \end{cases}$$

To compute ψ, first check whether $y \in dom(\theta)$, using a lookup table. Then check whether $M_i(x)$ halts. If so, simulate $\phi(y)$ and return the result. Applying the *s-m-n* and recursion theorems, we obtain d_i such that

$$\phi_{d_i}(y) = \begin{cases} \phi(y) & \textit{if } M_i(d_i)\downarrow \textit{ or } y \in dom(\theta) \\ \uparrow & \textit{otherwise.} \end{cases}$$

Now we again have that $M_i(d_i)\downarrow \Rightarrow d_i$ is an index for $\phi \Rightarrow d_i \notin S$ and $M_i(d_i)\uparrow \Rightarrow d_i$ is an index for $\theta \Rightarrow d_i \in S$, so $S \neq W_i$. ■

This result was proved by means of a demonic argument, but it has an epistemological flavor in its own right. Suppose we have a fixed, 1–1 encoding of pairs of natural numbers into numbers other than 0. 0 is reserved as a special symbol. Let $\varepsilon \in \mathcal{N}$ be a data stream. We say that ε is *for* ϕ just in case ε lists code numbers for all and only the pairs $(x, \phi(x))$ such that $\phi(x)$ is defined, together, possibly, with some 0s added. The funny business with 0 ensures that there is a data stream for the everywhere undefined function $\varnothing$, which otherwise would have none, since there is no pair in $\varnothing$ to present. Notice that other functions may have gratuitous 0s in their data streams as well, so that seeing a 0 is not a dead giveaway that the data stream will be for $\varnothing$. Let *data*(ϕ) be the set of all data streams for ϕ. If $\mathcal{A} \subseteq \mathcal{PR}$, let *data*($\mathcal{A}$) be the union of all *data*(ϕ) such that $\phi \in \mathcal{A}$. Finally, let $h_{\mathcal{A}}$ denote the empirical hypothesis that the data stream will be in *data*($\mathcal{A}$), and assume that correctness is truth. Now we have:

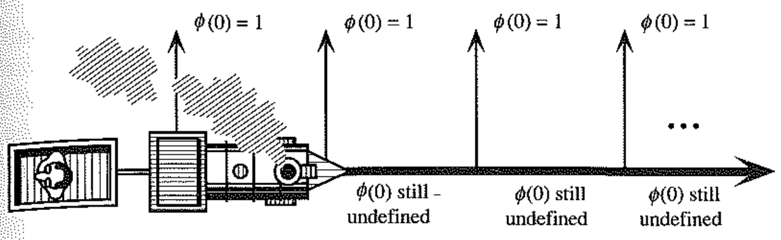
Corollary 6.8

(a) *If $\mathcal{A} \subseteq \mathcal{PR}$ and index($\mathcal{A}$) is r.e. then data($\mathcal{A}$) is data($\mathcal{PR}$)-open.*

(b) *If $\mathcal{A} \subseteq \mathcal{PR}$ and index($\mathcal{A}$) is r.e. then $h_{\mathcal{A}}$ is ideally verifiable with certainty given data($\mathcal{PR}$).*

Proof: (a) By the Rice-Shapiro theorem, we have that for each $\phi \in \mathcal{PR}$, $\phi \in \mathcal{A} \Leftrightarrow$ there is a finite $\theta \in \mathcal{A}$ such that $\theta \subseteq \phi$. Then we have for each $\varepsilon \in data(\mathcal{PR})$,

$$\varepsilon \in data(\mathcal{A}) \Leftrightarrow \exists \textit{ finite } \theta \in \mathcal{A}, \exists n \in \omega \; [\forall m \in dom(\theta)\; \theta(m) = y \Leftrightarrow \exists k \leq n\; \varepsilon_k \textit{ codes } (m, y)].$$



$\phi(0) = 1$
$\phi(0) = 1$
$\phi(0) = 1$
$\phi(0) = 1$
...
$\phi(0)$ still undefined
$\phi(0)$ still undefined
$\phi(0)$ still undefined

Figure 6.13

All the quantifiers occurring within the square brackets are bounded, and hence correspond to *finite* unions or intersections. The relation ε_k *codes*(m, y) is clopen in $\mathcal{N}$. Finally, the only remaining quantifiers are existential quantifiers over countable domains (since there are at most countably many finite θ), which correspond to countable unions. (b) follows from (a) and proposition 4.6. ■

What these corollaries tell us is that demonic arguments against ideal scientists with background knowledge *data*($\mathcal{PR}$) can be used to show that a broad class of purely fromal questions do not admit of computable positive tests. In other words, the failure of index sets violating the Rice-Shapiro condition to be r.e. is an instance of inductive underdetermination. The two phenomena involve exactly the same topological structure. By way of illustration, consider the problem $K_0 = \{x: 0 \in W_x\}$. Let α be a scientific method facing the hypothesis that the data stream is in *data*($\overline{K_0}$). The demon simply feeds α the data stream 0, 0, 0, . . . until α emits the certainty mark '!' followed by 1. Thereafter, the demon feeds *code*(0, 1) forever after (*Fig. 6.13*). If α ever says '!' immediately followed by 1, then the data stream is for the function $\{(0, 1)\}$, which is defined at 0 and hence is not in $\overline{K_0}$. But if the method never becomes sure, the data stream is for the everywhere undefined function, which is in $\overline{K_0}$. This is an instance of Sextus's ancient argument against inductive generalization. But in light of corollary 6.8, it shows a purely computational fact, namely, that $\overline{K_0}$ is not r.e.

7. Some Disanalogies

The purpose of this chapter has been to illustrate the strong structural and philosophical analogy between standard results in the theory of computability and classical skeptical arguments for local underdetermination, so that degrees of uncomputability may be thought of as levels of underdetermination. The analogy between ideal inquiry and Turing computability is not exact, however. First, the analogy has been illustrated here only for index sets or for closely related sets like $\overline{K}$. Second, the converse of proposition 6.7 fails because it is easy to construct non-r.e. index set problems that generate inductive inference problems that are ideally verifiable with certainty.[9] It also turns out that demonic arguments cease to establish the uncomputability of index sets at level 3 because the data set corresponding to any index set is always $\Delta[data(\mathcal{PR})]^B_3$. This follows simply from the fact that $\mathcal{PR}$ is countable. This is also the best general upper bound available given just background knowledge $data(\mathcal{PR})$.

Proposition 6.9

(a) *For each countable* $\mathcal{S}$, $data(\mathcal{A}) \cap data(\mathcal{S}) \in \Delta[data(\mathcal{S})]^B_3$.

(b) $\exists \mathcal{A} \exists \mathcal{S} \subseteq \mathcal{PR}$ *such that*
$$data(\mathcal{A}) \cap data(\mathcal{PR}) \notin \Sigma[data(\mathcal{S}))]^B_2 \cup \Pi[data(\mathcal{S})]^B_2,$$

Proof: (*a*) *Let* $\mathcal{A}, \mathcal{S} \subseteq \mathcal{P}$ be given, with $\mathcal{S}$ countable. Let $\varepsilon \in \mathcal{N}$. Then we have, for each $\varepsilon \in data(\mathcal{S})$:

$$\begin{aligned}
&\varepsilon \in data(\mathcal{A})\\
&\quad \Leftrightarrow \exists \phi \in \mathcal{A} \cap \mathcal{S} \text{ such that } \forall x, y \in \omega, \phi(x) = y \Leftrightarrow \exists z\ \varepsilon_z \text{ encodes } (x, y)\\
&\quad \Leftrightarrow \exists \phi \in \mathcal{A} \cap \mathcal{S} \text{ such that } \forall x, y[(\phi(x) = y \;\&\; \exists z \text{ such that } \varepsilon_z \text{ encodes}\\
&\qquad (x, y)) \text{ or } (\phi(x) \neq y \;\&\; \forall z, \varepsilon_z \text{ does not encode } (x, y))]\\
&\quad \Leftrightarrow \exists \forall [\exists \cdots \text{ or } \forall \cdots]\\
&\quad \Leftrightarrow \exists \forall \forall \exists \cdots\\
&\quad \Leftrightarrow \exists \forall \exists \cdots
\end{aligned}$$

Since the quantifiers all range over countable sets, they correspond to countable intersections and unions, so $data(\mathcal{A}) \cap data(\mathcal{S}) \in \Sigma[data(\mathcal{S})]^B_3$. Also, for each $\varepsilon \in data(\mathcal{S})$,

$$\begin{aligned}
&\varepsilon \in data(\mathcal{A}) \Leftrightarrow \forall \phi \in \mathcal{S} - \mathcal{A},\\
&\exists x, y \in \omega \text{ such that } \phi(x) = y \text{ and } \forall z\ \varepsilon_z \text{ does not encode } (x, y) \text{ or}\\
&\exists x \text{ such that } \phi(x)\uparrow \text{ and } \exists z, w \text{ such that } \varepsilon_z \text{ encodes } (x, w).
\end{aligned}$$

By regrouping the quantifiers, it follows that

$$data(\mathcal{A}) \cap data(\mathcal{S}) \in \Pi[data(\mathcal{S})]^B_3.$$

(b) Let $\mathcal{A}_{even} = \{\phi\colon \exists n \forall \text{ even } m \geq n, \phi(m) = 0\}$ and let $\mathcal{A}_{odd} = \{\phi\colon \exists n \forall \text{ odd } m \geq n, \phi(m) = 0\}$. Let $\mathcal{A} = \mathcal{A}_{even} \cap \overline{\mathcal{A}_{odd}}$. Now proceed with a demonic argument along the lines of the one given in the example at the end of section 6 to show that $data(\mathcal{A})$ is neither $\Sigma[data(\mathcal{PR})]^B_2$ nor $\Pi[data(\mathcal{PR})]^B_2$. ■

[9] Cf. exercise 6.6.

Propositions 6.7 and 6.9 leave open an interesting question. Can we always show that an index set is not Σ^A_2 by means of an ideal demonic argument? That is:

Question 6.10

If $\mathcal{A} \subseteq \mathcal{PR}$ *and* $index(\mathcal{A})$ *is* Σ^A_2 *then is it the case that*
$$data(\mathcal{A}) \cap data(\mathcal{PR}) \text{ is } \Sigma[data(\mathcal{PR})]^B_2? \qquad ■$$

It is easy to show that if we restrict our attention to those Σ^A_2 sets that are r.e. unions of co-r.e. index sets, then the answer is affirmative (exercise 6.7). But since a Σ^A_2 index set could conceivably be an r.e. union of co-r.e. sets that are not index sets, it is not clear that the answer is affirmative in general.

Exercises

6.1. Prove the analogue of proposition 6.1 for the other notions of computational success introduced in section 6.2.

6.2. Define the *limiting halting problem* as follows:

$$K_{\text{lim}} = \{x\colon M_x \text{ started on input } x \text{ outputs a sequence that stabilizes to } 1.\}$$

Show that $K_{\text{lim}} \in \Sigma^A_2 - \Pi^A_2$, both by means of depicting $\overline{K_{\text{lim}}}$ as the counterdiagonal of a table and by means of a demonic argument using Kleene's recursion theorem.

6.3. Use corollary 6.8(a), together with a demonic argument, to show that $\{i\colon W_i$ is infinite$\}$ is not r.e.

6.4. Use a demonic argument based on Kleene's recursion theorem together with proposition 6.3 and the results of exercise 6.1 to prove that $I = \{i\colon W_i$ is infinite$\}$ is not Π^A_3. Follow the strategy of the arguments presented after proposition 6.4.

6.5. Show the converses of corollary 6.8 are false.

6.6. Complete the proof of proposition 6.9(b).

6.7. Settle question 6.10 in the affirmative in the restricted case of r.e. unions of co-r.e. index sets.