520
Notre Dame Journal of Formal Logic
Volume 28, Number 4, October 1987

Turing Projectability

TIMOTHY McCARTHY and STEWART SHAPIRO

1 Introduction Let N be the set of natural numbers. A function $g: N \to N$ is *effectively* (or *mechanically*) computable if there is an algorithm or effective procedure which, given (a representation of) an integer n as input, terminates after finitely many steps, and yields (a representation of) $g(n)$ as output. The Church–Turing thesis (CT) states that the effectively computable number-theoretic functions are precisely the Turing computable number-theoretic functions, or, equivalently, the recursive functions. In this paper, we develop a generalization of the notion of mechanical computability and use CT to argue for an analogous thesis concerning that generalization.[1]

It is an important aspect of the classical notion of effective or mechanical procedure that an effective procedure terminate for each natural number given as input.[2] We consider here a generalization of effective computability according to which a *nonterminating* effective process may be said to determine the values of a number-theoretic function. Our idea is to describe such a process in the following terms. There is, first, an effective procedure (in the ordinary sense) which, applied to any input m, generates a computation that proceeds in stages $\sigma_{1m}, \sigma_{2m}, \ldots$. Secondly, there is an effective function f which, applied to any stage σ_{nm} of the computation, yields a *projection*, or tentative value, for the computation at the stage n. The projection is subject to revision, in the sense that at a further stage σ_{km} $(k > n)$, it may happen that $f(\sigma_{km}) \neq f(\sigma_{nm})$. But we allow the projected value to change only *finitely* often, so that for each m there is a value p such that

$$\exists n \forall k > n \, f(\sigma_{km}) = p.$$

We say that the given pair of effective procedures is a *projective strategy*. The function g is said to be *projected* by that strategy if and only if for each m an n can be found such that

$$\forall k > n \, f(\sigma_{km}) = g(m).$$

In this case, we say that the strategy is *stable* for m at stage n.[3]

Received December 9, 1985; revised December 30, 1985

Intuitively, then, a projective strategy for g is an effective procedure which, given any number m as input, produces a *conjecture* for the value $g(m)$, but (possibly) keeps on working. Subsequently, the procedure may change the conjecture. But we require that the conjectured value change only finitely often. Eventually the value $g(m)$ is produced and is stable under further computation.

A function is *effectively projectable* if there is a strategy that projects it. A number-theoretic relation is said to be effectively projectable if its characteristic function is effectively projectable, and to be projected by any strategy that projects its characteristic function.

It is clear that all computable number-theoretic functions are effectively projectable. For, let a computable function $f: N \to N$ be given. Let M be a Turing machine computing f. A projective strategy for f is the following: for any $n \in N$, apply M to n in stages. At stages before the machine halts, project 0. At the stage at which the machine halts and thereafter, project the computed value for $f(n)$. However, the converse does not hold: not every effectively projectable function is effectively computable. For example, the characteristic function of any recursively enumerable set is effectively projectable. Recall that the characteristic function C_A of a set A is defined for each $n \in N$ by

$$C_A(n) = \begin{cases} 1 \text{ if } n \in A \\ 0 \text{ if } n \notin A. \end{cases}$$

The following is a projective strategy for C_A: given n, at stage 0, project the value 0 for $C_A(n)$. Enumerate A in stages $k > 0$. If n appears in the enumeration at some stage k, we project 1 at stages $\geq k$. Otherwise, we continue to project 0. Clearly, for each n, the correct value of $C_A(n)$ will eventually be produced by this process, and will be stable in the sense described.

If $h: N \times N \to N$ and $p, q \in N$, we write that $\lim_{k\to\infty} h(p,k) = q$ if $h(p,k) = q$ holds for all sufficiently large k. Using CT, we may obtain a simple characterization of the effectively projectable functions: a function $g: N \to N$ is effectively projectable iff there is a recursive function $h: N \times N \to N$ such that for each n we have

(*) $\lim_{k\to\infty} h(n,k) = g(n)$.

For if g satisfies this relation for a given recursive function h, we may take σ_{kn} above to be a computation of $h(n,k)$ for each k, n, and f to be the appropriate result-extracting function. Conversely, if g is effectively projectable, by CT there is a recursive function h which maps each pair $\langle n,k\rangle$ onto the projected value for $g(n)$ at stage k, and this h will satisfy (*) for each n. Thus, granted CT, the notion of 'effectively projectable set' is equivalent to Putnam's concept of a *trial and error predicate* [3]: a set A is a trial and error predicate if there exists a recursive binary function h such that $\lim_{k\to\infty} h(n,k) = C_A(n)$ holds for each $n \in N$.

In this paper, we shall give another characterization of the effectively projectable functions in terms of a generalization of the Turing model of computability, obtaining a theorem of Putnam as a corollary (Section 2). In Section 3 we will examine the relation of effective projectability to the problem of extrapolating deterministic regularities in inductive logic. The concluding sections con-

cern ways of restricting the definition of effective projectability which lead to more limited classes of functions.

2 Turing projectability In this section, we shall set up a model of effective projectability analogous to the Turing model of computability. Informally, an *extended Turing machine* M is a Turing machine equipped with two tapes, a *projection tape* and a *computation tape*, each possessing a printer and a scanner (or, alternatively, a single printer-scanner that can move between the tapes). The machine states of M are of two sorts, *projection states* and *computation states*, but in any configuration M is in exactly one state. The instructions of M determine the action and the next state of M as a function of its current state and the symbol that it currently scans.[4]

By a *configuration* of an extended Turing machine M we mean a complete description of the content of the computation and projection tapes, the positions of the scanners, and the current computation or projection state. The instructions of M then effectively determine, for each configuration of M, the next configuration (if any). By a *computation* of M we mean a sequence of configurations $\langle \sigma_1, \ldots, \sigma_n \rangle$ such that $\forall i < n$, σ_{i+1} arises from σ_i by applying an instruction of M. Notice that, unlike the analogous situation with ordinary Turing machines, a computation of M may be incomplete in the sense that M may not have terminated at the last configuration in the sequence. Extended Turing machines and their configurations can be effectively coded via a Gödel numbering in the usual way, as can computations. In what follows, we freely identify syntactical objects with their Gödel numbers.

We introduce an analogue $T^*(z,y,x)$ of the Kleene T-predicate: for any e, n, $m \in N$, $T^*(e,n,m)$ holds iff m is the Gödel number of a computation of the extended machine (with Gödel number) e at input n. Notice that T^* is not completely analogous to the Kleene T-predicate in that $T^*(e,n,m)$ does not entail that m codes a terminating computation. Nevertheless, T^* is a recursive relation. We write $m\alpha k$ to mean that m and k are codes of sequences and m codes an initial subsequence of the sequence with code k. Of course, α is also a recursive relation. Notice that $T^*(e,n,k)$ and $m\alpha k$ entail that $T^*(e,n,m)$.

We define a partial recursive function $|\ |$ on N as follows. The domain of $|\ |$ is the set of Gödel numbers of sequences of extended machine configurations. If m is the Gödel number of such a sequence, then $|m|$ is the integer described on the projection tape of the last configuration of the sequence. If σ is a single configuration, $|\sigma|$ is defined to be $|\langle\sigma\rangle|$. If e is the Gödel number of a machine and m the Gödel number of a computation thereof, m is said to be *stable for* e iff

$$\forall n \forall x((m\alpha x \ \&\ T^*(e,n,x)) \rightarrow |x| = |m|).$$

Finally, a number-theoretic function f is said to be *projected by* an extended Turing machine M (with Gödel number e) iff, for each $n \in N$, there is a computation code m of M for input n such that m is stable for e and $|m| = f(n)$. A function is *Turing projectable* iff some extended Turing machine projects it.

Projectability and Turing projectability are related as are computability and

Turing computability. In fact, it follows from CT by means of a simple lemma that the effectively projectable number-theoretic functions are precisely the Turing projectable functions.

Lemma *A number theoretic function f is Turing projectable if and only if there are Turing computable functions f^* and g such that for each n, $f^*(n)$ is the index of a Turing machine and*

(1) $\exists m \forall k > m \; g(\{f^*(n)\}(k)) = f(n)$.

Proof: Let $f: N \to N$ and M be an extended Turing machine that projects f. For each n, let $f^*(n)$ be an index for an ordinary Turing machine that enumerates the computations of M at n in order of increasing length; if M applied to n halts in k steps, we set $\{f^*(n)\}(j) = \{f^*(n)\}(k)$ for all $j > k$. It is clear that $f^*(n)$ may be obtained effectively from n. Let g be $|\ |$. Then (1) holds; and both f^* and g are Turing computable.

Suppose now that f^* and g are Turing computable functions that fulfill (1) for each n. We give an intuitive description of an extended Turing machine M that projects f.

For any input n, the computation of M proceeds in stages. At stage 0, 0 is projected. At any stage $k > 0$, the following procedure is executed on the computation tape: using instructions for f^*, M first computes $f^*(n)$. Using the instructions of the universal Turing machine, M now computes $\{f^*(n)\}(k)$. Finally, using instructions for g, M computes $m_k = g(\{f^*(n)\}(k))$. By means of the projection instructions it then inscribes m_k on the projection tape. By (1), for each n, a k can be found for which $\forall p > k \; m_p = f(n)$; thus, at each n, the projected value of M is $f(n)$ at sufficiently large stages. Therefore, M projects f.

We may now obtain an analogue of CT for effective projectability. Suppose that $f: N \to N$ is effectively projectable. Recall that a projective strategy for f consists of a pair of effective functions (h,p) such that for each n, $h(n)$ determines a computation that proceeds in stages $\sigma_{n1}, \sigma_{n2}, \ldots$ and p maps each stage σ_{nj} effectively onto a projected value which is $f(n)$ at sufficiently large stages. From CT, we may assume h and p to be Turing computable, and that for each n, $h(n)$ is the index of a Turing machine that enumerates the stages σ_{nj}. Thus, taking $f^* = h$ and $g = p$ in the lemma satisfies (1) for each n, so that f is Turing projectable. Conversely, if f is Turing projectable, by the lemma there is a pair of Turing computable functions (f^*, g) such that (1) holds for each n. These functions determine an obvious projective strategy for f.

We now turn to a recursion-theoretic characterization of the class of Turing projectable functions. The characterization problem turns out to be quite simple:

Theorem 1 *A number-theoretic function is Turing projectable if and only if it is recursive relative to the halting problem for ordinary Turing machines.*

Proof: Suppose that $f: N \to N$ is projected by an extended Turing machine M. We show that for each $n \in N$, $f(n)$ may be computed relative to the halting problem for ordinary Turing machines. Consider the r.e. sequence $\{\sigma_k\}$ of com-

putations of M applied to n in order of increasing length. At stage k in the computation of $f(n)$, we compute σ_k and determine whether

(2) $\exists p > k \; |\sigma_p| \neq |\sigma_k|$.

Since (2) is a Σ_1 predicate, this can be determined recursively relative to the halting problem. If the answer is yes, we proceed to stage $k + 1$. If the answer is no, we set $f(n) = |\sigma_k|$ and stop. Since M projects f, we must eventually reach a stage k_n such that $\forall p \geq k_n \, f(n) = |\sigma_p|$, so that the indicated procedure terminates after k_n stages, and gives the correct result.

Conversely, suppose that f is computable relative to the halting problem. We give a projective strategy for f. Recall that solutions to cases of the halting problem assign truth values to sentences of the form

(3) $\exists x \; T(e,m,x)$,

where $T(z,y,x)$ is the *ordinary* Kleene T-predicate ("x is a (convergent) computation of machine with code z at input y"). (Note that since (3) is Σ_1, there is a projective strategy for the relation $\{\langle e,m\rangle$: e is the Gödel number of a Turing machine that halts at input $m\}$.)

To project the value of $f(n)$, we give a computation that proceeds in stages. At each stage, a projected value for $f(n)$ will be effectively determined. At stage 0, project 0. We divide each stage $k > 0$ into three parts Ak, Bk, and Ck. At substages Ak, the truth values of the sentences $T(e,m,p)$ for all e, m, $p < k$ are computed. At substage Bk, k steps in the computation of $f(n)$ relative to the halting problem are executed. When a truth-value of a sentence of the form (3) is required, assign it the value F unless a computation verifying an instance of this sentence was executed at substage Ak, in which case the value T is assigned. At substage Ck, determine if a value for $f(n)$ was produced at stage Bk. If so, project this value for stage k. If not, project the value projected at stage $k - 1$.

We claim that the procedure just described is a projective strategy for f. Since f is recursive relative to the halting problem, for any n there are finitely many pairs $\langle e_1,m_1\rangle,\ldots,\langle e_z,m_z\rangle$ of integers such that the relative recursive procedure for f determines the correct value of $f(n)$ given the truth-values of the statements (3) for $e = e_1$, $m = m_1$, $O < 1 \leq z$. Let $\langle e_{j1},m_{j1}\rangle,\ldots,\langle e_{jv},m_{jv}\rangle$ be precisely the pairs $\langle e_i,m_i\rangle$ such that $\exists x T(e_i,m_i,x)$ holds. For each s, $O < s \leq v$, let $N_s = \min\{p$: $T(e_{js},m_{js},p)\}$, and $M_s = \max\{e_{js},m_{js},N_s\}$. Then a computation verifying (3) for $e = e_{js}$, $m = m_{js}$ will appear by stage M_s. Let p be the maximum value of the M_s and q be the number of steps in the complete computation of $f(n)$ relative to the halting problem. Then our proposed strategy will produce the correct value of $f(n)$ at all steps after $\max\{p,q\}$. Thus f is effectively projectable and hence, by CT, Turing projectable. This completes the proof of Theorem 1.

We now obtain a theorem of Putnam ([3], Theorem 1):

Corollary *A set of natural numbers is a trial and error predicate if and only if it is* Δ_2.

Proof: A predicate is Δ_2 if and only if it is recursive relative to the halting problem. A set of natural numbers is a trial and error predicate if and only if its char-

acteristic function is Turing projectable. The corollary now follows from Theorem 1.

3 Generalized computability and inductive logic Inductive logicians have discussed the possibility of a *perfect learning strategy*, i.e., a procedure for forming belief that will correctly extrapolate any (mechanical) regularity (cf. [1], [4]). In order to provide a suitable mathematical representation of this question, it has been customary to convert it into the question of whether there is a uniform strategy for solving the following class of problems:

(I) Given a sufficiently large finite initial segment of a mechanically generated sequence $\{n_1\}$ of natural numbers, to extrapolate the rest of the sequence.

For any $\alpha: N \to N$, $n \in N$, let $\alpha^*(n)$ code the finite sequence $\langle \alpha(0), \ldots, \alpha(n-1) \rangle$ in N. A strategy for solving the problems (I) for a given class R of functions is representable by a number-theoretic function Ψ such that for each function of $\alpha \in R$ there is an integer n_α such that

(P) $\forall n > n_\alpha \Psi(\alpha^*(n)) = \alpha(n)$.

If (P) holds for some choice of n_α, we say that Ψ *extrapolates* α. Informally, Ψ may be thought of as describing the mathematician's pastime of considering a finite initial segment of a sequence and trying to guess the next element. The assertion "Ψ extrapolates α" is the claim that Ψ describes a process that eventually produces the correct result; i.e., given a large enough initial subsequence, the process will always conjecture the (correct) next element.

It might be suggested that if Ψ is to describe a learning strategy for a class R of sequences, then there must be a uniform *bound* on the number of steps required for the strategy to extrapolate a sequence in R; or, at least, that there be a uniform bound on the number of times the procedure can be defeated by a sequence in R. However, although these are desirable features of a learning strategy, they are not plausibly necessary. It frequently occurs that a class K of possible physical systems can be described by laws of a certain form without there being an a priori bound on the complexity of the laws (in terms of some natural complexity measure, e.g., length). There is no such bound because there is no a priori bound on the complexity of the *systems* under consideration. Consider the problem of extrapolating such a law from data presented by a system in K. A natural inductive strategy would be to try to fit the data by laws in order of increasing complexity (with respect to a suitable measure of complexity; for this example, let us suppose it to be decidable whether a given finite array of data is generated by a law of the relevant form). For any integer n, there is a system which is not described by any law of complexity $\leq n$, and so the procedure will not terminate in fewer than n steps for the system. Moreover, if we suppose that the data for this system are presented in stages $\sigma_1, \sigma_2, \ldots$ such that, for each $k \leq n$, the data of σ_k are accommodated by a law of complexity k, but by no law of complexity $<k$, this system will defeat the procedure at least n times. Nevertheless, the procedure seems unexceptionable. We are justified in continuing to use it in the face of repeated defeats, for our theory of the sys-

tem in question implies that it will eventually succeed. The fact that it is defeatable any finite number of times is attributable not to a defect in the strategy, but to the complexity of the systems to which it is applied.

We are interested in inductive strategies for the class of all mechanically generated sequences. Granted (CT), then, we may focus on providing such a strategy for the class of all recursive functions. It is easily shown that no recursive function Ψ has the property (P) for each recursive function α (for a proof, see [4]). In fact, by essentially the same method one can prove a more general result. A collection T of number-theoretic functions is said to be a *Turing cone* iff

$$\forall f \in T \, \forall g (g \leq_T f \rightarrow g \in T),$$

where '$\leq_T$' denotes Turing reducibility. We then have:

Theorem 2 *No element of a Turing cone T extrapolates each element of T.*

Proof: Let a Turing cone T be given, and consider any $f \in T$. We construct a $g \in T$ that f does not extrapolate. g is described recursively as follows:

(4) $g(0) = 0$
(5) $g(n + 1) = f(g^*(n + 1)) + 1.$

Clearly, g is recursive relative to f, so that $g \in T$. If f extrapolates g, there exists an n_g such that

$$g(n_g + 1) = f(g^*(n_g + 1)).$$

But taking $n = n_g$ in (5) we have

$$g(n_g + 1) = f(g^*(n_g + 1)) + 1,$$

a contradiction. Thus f cannot extrapolate g.

However, we will show that there is a *projective strategy* that uniformly solves the problems (I) for each recursively generated sequence $\{n_i\}$.

Theorem 3 *There is a Turing projectable function ϕ: $N \rightarrow N$ such that ϕ extrapolates any recursive function.*

Proof: In virtue of Theorem 1, it suffices to construct a function ϕ such that ϕ extrapolates any recursive function and ϕ is recursive relative to the halting problem for ordinary Turing machines. This is done as follows. Let $e_0, e_1, \ldots$ be a recursive enumeration of Turing machine indices. We assume that the indicated coding of finite sequences from N is onto N, and for any $n \in N$ we let $\langle n_0, \ldots, n_{k(n)} \rangle$ be the sequence with code n. We give a procedure by which $\phi(n)$ may be computed relative to the halting problem. Find the first index e_{i_n} such that $\{e_{i_n}\}(j)$ converges to n_j for each $j \leq k(n)$, and converges for $j = k(n) + 1$. Set:

$$\phi(n) = \{e_{i_n}\}(k(n) + 1).$$

ϕ is clearly defined for each n and computable relative to the halting problem.

For later purposes we note that ϕ is projected by the following extended

Turing machine M. For any input n, the computation of M proceeds in stages. At each stage m, M does m steps in the computation of the functions $\{e_1\}, \ldots, \{e_m\}$ at all inputs $\leq k(n) + 1$ on its computation tape. If at stage m, M finds an index e_i such that $\{e_i\}(j) = n_j$ for each $j \leq k(n)$ and for which $\{e_i\}$ converges at $k(n) + 1$, then it projects $\{e_i\}(k(n) + 1)$ for the least such i. Otherwise, M projects 0.

We now claim that ϕ extrapolates any recursive function f. Let e_v be the first number in the enumeration $e_1, e_2, \ldots$ which is an index of f. Let n_f be the least integer n such that $\forall i < v$ either $\{e_i\}(j)$ does not converge for some $j \leq n$ or $f^*(n) \neq \{e_i\}^*(n)$. Then $\phi(f^*(j)) = \{e_v\}(j) = f(j)$ for each $j \geq n_f$, so that ϕ extrapolates f.

On the other hand, there is no Turing projectable function that extrapolates each *Turing projectable* function. This is an immediate consequence of Theorems 1 and 2, and the fact that the collection of functions recursive relative to the halting problem is a Turing cone.

The interest of Theorem 3 depends upon the extent to which extended Turing machines are relevant to an acceptable model of inductive procedure. Here we argue for a limited sort of relevance. An extrapolating procedure for a function f is applied to finite initial segments of the graph of f. These are regarded as empirical data on the basis of which the procedure frames predictions about the further behavior of the graph of f. Given a sufficiently large finite initial segment as input, the procedure produces a rule which describes that initial segment. Call this rule a *covering law* for that segment.

The conjectured covering law is subject to revision under two circumstances: First, some new data—a larger initial segment of the graph of f—may be found which is inconsistent with the covering law. In this case, the procedure searches for an alternative covering law for the largest known initial segment. We think of these initial segments as derived from external experience.

Secondly, however, the conjecture may be changed in the face of further *theoretical* experience, independently of further external experience. The procedure may produce a covering law that provides a better explanation of the *given* data. In this case, a finite initial segment of f is given, together with a covering law for it. Subsequently, an alternative law is found which better explicates the given initial segment, without a longer initial segment being given. For example, it may be found that a *simpler* algorithm (with respect to a suitable simplicity measure) generates the given initial segment; or that that segment may be described by an alternative law that is a logical consequence of a previously acquired theory. (Recall from Church's theorem that it is not generally *decidable* whether a law *is* a consequence of a given theory.)

A conjectured hypothesis, then, may be revised either in the face of recalcitrant data or in the aftermath of a theoretical reevaluation. To some extent, the present model of inductive procedure in terms of the projectable function ϕ reflects both aspects of conjecture revision. On the suggested model, a finite initial segment σ of the function to be extrapolated is recorded on the computation tape of the machine M projecting ϕ. The machine searches for a covering law for σ in the form of a partial recursive function, which becomes a basis for the prediction of further values. The conjectured covering law is subject to revision in two ways. First, a longer initial segment σ^+ may appear which is

incompatible with the conjecture. The procedure then searches for a Turing machine index that generates σ^+. Second, even if new data do not appear, the conjecture is subject to revision due to the nonterminating character of the computation. Once a covering law is found for σ, the machine will continue to search for another covering law with a smaller index. If we imagine the initially given ordering of Turing machine indices to reflect a prior ordering of hypotheses in terms of theoretical acceptability (say, by means of a simplicity measure), the second class of revisions represents a corresponding sort of theoretical reevaluation.

4 Projectability in finite space In this and the following sections, we shall consider some restrictions on the definition of projectability which determine more limited classes of functions. In some cases, these restrictions will lead us back to the class of Turing computable functions.

We first consider limitations on the computational space used by an extended Turing machine. Let $p = \langle \sigma_1, \ldots, \sigma_m \rangle$ be a sequence of configurations of an extended Turing machine M. By the *space used by* p, we mean the total number of loci on the tapes of M which either contain symbols or which occur between loci which contain symbols in some σ_i. If M is an extended Turing machine and n is a natural number, let $P_M(n,0)$ be the initial configuration of M at input n, and for any (n,m), let $P_M(n,m+1)$ be the configuration following $P_M(n,m)$, if any; otherwise, it is $P_M(n,m)$. By $sp_M(n,m)$ we denote the space used by the sequence of configurations $\langle P_M(n,0), \ldots, P_M(n,m) \rangle$. The *space used by M at n* is defined to be the maximum over all natural numbers m of the values $sp_M(n,m)$, if this maximum exists, and is otherwise undefined. We say that M is a *finite space machine* if this value is defined for each n. In short, a finite space machine is an extended Turing machine which uses only a finite amount of space for each input.

We begin with the following observation. One may think of an *ordinary* Turing machine as an extended Turing machine, one whose projection at any stage coincides with the content of its tape at that stage.[5] Thus the definitions of stability and projection make sense for ordinary Turing machines. In particular, an ordinary Turing machine configuration is stable just in case no printing or erasing instructions are executed in succeeding configurations, and an ordinary Turing machine M projects a function $f\colon N \to N$ iff for each input $n \in N$, M eventually enters a stable configuration with $f(n)$ on its tape. Thus, if an ordinary Turing machine projects a function, it executes only finitely many printing or erasing instructions for each input. Therefore, an ordinary Turing machine that projects a function is a finite space machine in the sense of the last paragraph.

The definition of projectability for ordinary Turing machines thus relaxes the usual definition of Turing computability by dropping the requirement that the machine halt when it has produced the "result" of the computation. The next theorem says that this relaxation does not change the class of computable functions.

Theorem 4 *Let $f\colon N \to N$. Then f is projectable by an ordinary Turing machine iff f is (Turing) computable.*

Proof: Right to left is, of course, immediate. Assume, then, that f: $N \to N$ is projected by an ordinary Turing machine M. It suffices to show that the collection of stable configurations for M is recursive. For in this case, to compute $f(n)$ we apply M to n to obtain a recursive enumeration $\sigma_1, \sigma_2, \ldots$ of configurations, testing each for stability. Since M projects f, we must eventually reach an m such that σ_m is a stable configuration for M. We then output $|\sigma_m|$ for the least such m. So we need only prove:

Lemma 5 *Given an ordinary Turing machine M and a configuration σ for M, it is decidable whether σ is stable for M.*

Proof: Given M and σ, consider the following algorithm:

Apply M to the configuration σ to obtain a sequence of configurations $\{\sigma_i\}$; i.e., $\sigma_0 = \sigma$ and for any n σ_{n+1} is the configuration following σ. Execute the following instructions, in order, at each stage in the computation:

(1) If a printing or erasing instruction is executed, output "NO" and stop.
(2) If a halting state is assumed, output "YES" and stop.
(3) If for some n and $m > n$ $\sigma_m = \sigma_n$, then output "YES" and stop.
(4) If for some n and $m > n$:
  (i) σ_m and σ_n indicate the same state;
  (ii) in σ_n the machine is scanning a space to the right of any space on which a printed symbol appears;
  (iii) in σ_m the machine is scanning a space to the right of the space scanned in σ_n;
  (iv) for any i, $n < i < m$, in σ_i the machine is scanning a space to the right of any space containing a printed symbol;
  then output "YES" and stop.
(5) Like (4), with "left" in place of "right".

We claim that: (i) if this algorithm halts, then it gives the correct results; and (ii) the algorithm halts for any configuration σ.

Ad(i): If the algorithm halts because of instruction (1) or (2), then the result is clearly correct. Now suppose that the algorithm halts because of instruction (3). Then when applied to σ, M will eventually pass through the sequence σ_n, $\sigma_{n+1}, \ldots \sigma_m$ without executing a printing or erasing instruction. Since $\sigma_n = \sigma_m$, the sequence will enter an infinite loop. Thus, σ is stable. If the algorithm halts due to instruction (4), the machine passes through a sequence $\sigma_n, \sigma_{n+1}, \ldots, \sigma_m$ in which only blank spaces are encountered, which puts the machine in the state of σ_n and places the scanner to the right of its position in σ_n. The machine will then follow indefinitely an analogous sequence of configurations without executing a printing or erasing instruction. Thus σ is stable for M in this case as well. If the algorithm halts because of instruction (5), the result is similar.

Ad(ii): We consider four cases. Case (I): the initial configuration σ is not stable for M. Then when applied to σ the machine will eventually print or erase a symbol, in which case instruction (1) will be executed. Case (II): M applied to σ halts without executing a printing/erasing instruction. Then instruction (2) will eventually be executed. Case (III): M applied to σ neither halts nor executes

a printing/erasing instruction, and for each n there is an $m > n$ such that σ_m indicates the machine to be scanning a symbol. Notice that with the tape contents fixed, there are only finitely many configurations a given machine may assume in which it scans a (nonblank) symbol. Thus eventually a configuration will be repeated and instruction (3) will be executed. Case (IV): the machine neither halts nor executes a printing/erasing instruction, and there exists an n such that for each $m > n$ σ_m indicates that the machine is scanning a blank. Thus in $\sigma_{n+1}, \sigma_{n+2}, \ldots$ the scanner either (i) stays within some finite block of blank spaces or (ii) tracks indefinitely to the right (possibly with backtracking) without encountering a symbol, or (iii) tracks indefinitely to the left without encountering a symbol. Since the total number of states is finite, if (i) holds, instruction (3) will be executed; if (ii) holds, instruction (4) will be executed; and if (iii) holds, instruction (5) will be executed. This completes the proof of Lemma 5.

It is tedious but straightforward to extend Lemma 5 to finite space machines in general. The reason is that such machines also produce only finitely many configurations in which a (nonblank) symbol is scanned. It follows that the collection of all stable configurations of any finite space machine is recursive. So in analogy to Theorem 4, we have:

Theorem 6 *Suppose that f: $N \to N$ is projectable by a finite space machine. Then f is Turing computable.*

Thus the finite space restriction on projectability leads us back to precisely the class of computable functions.

5 Projectability with convergence bounds If an extended Turing machine M projects a function f: $N \to N$, at each input n, M produces a computation which eventually results in a stable projection for $f(n)$. One can then ask, for each input, how many steps are required for a stable value to be achieved. Say that a function g: $N \to N$ is a *stability measure for M* if for each n, M applied to n reaches a stable value in $g(n)$ steps. More formally, g is a stability measure for M iff, for each $n \in N$, there is a computation $\langle \sigma_1, \ldots, \sigma_k \rangle$ of M at input n such that $k \leq g(n)$ and σ_k is stable for M. Notice that an extended machine has a stability measure only if there is a function that it projects; the first observation of this section is that in such a case the machine has a *projectable* stability measure.

Theorem 7 *Let M be an extended Turing machine that projects a number-theoretic function. Then there is a projectable stability measure for M.*

Proof: Consider the following projective strategy. Given any n as input, apply M to n. Let $\sigma_1, \sigma_2, \ldots$ be the resulting sequence of configurations for M. We specify a projection $G_n(m)$ of the strategy at stage m by induction on m: $G_n(0) = 0$. For each m,

$$G_n(m+1) = \begin{cases} G_n(m) \text{ if } |\sigma_{m+1}| = |\sigma_m| \\ m + 1 \text{ otherwise.} \end{cases}$$

Thus, at stages where M changes its projection, G conjectures the number of steps that M has run up to that point. Let $g(n)$ be the least integer m such that the computation of M at input n produces a stable configuration in m steps. Then g is a stability measure for M, and for each n we have

$$\forall m > g(n) G_n(m) = g(n),$$

so that the suggested strategy projects g.

On the other hand, it is immediate that a function is Turing reducible to any stability measure for any extended machine that projects it:

Theorem 8 *Let f: $N \to N$, let M project f, and let g be a stability measure for M. Then f is recursive relative to g.*

Proof: Let f, M and g satisfy the hypothesis. Let n be given. To compute $f(n)$ relative to g, do $g(n)$ steps in the computation of M applied to n. Let $\sigma_1, \ldots, \sigma_{g(n)}$ be the corresponding sequence of configurations for M. Output $|\sigma_{g(n)}|$. Since g is a stability measure for M, we have $f(n) = |\sigma_{g(n)}|$ for each n, so that this procedure, which is clearly effective relative to g, computes f.

Corollary 9 *If an extended Turing machine has an effective stability measure, then it projects a recursive function.*

Conversely, it is clear that any recursive function is projected by an extended Turing machine with a recursive stability measure. Thus the condition that a function is projectable by an extended machine with a computable stability measure is equivalent to the computability of that function. However, the following theorem shows that it is not true that any extended machine that projects a recursive function can be associated with a recursive stability measure:

Theorem 10 *There is an extended machine M that projects a computable function such that there exists no computable stability measure for M.*

Proof: We construct an extended Turing machine M that projects the constantly zero function. Let M_0 be an extended machine that projects a nonrecursive function f. Given an input n, M proceeds in stages as follows: on its computation tape, M simulates the computation of M_0 at n, producing the configurations $\sigma_1, \sigma_2, \ldots$ of M_0. For each k, if $|\sigma_k|$ differs from $|\sigma_{k+1}|$, M prints 1 on its projection tape, and if $|\sigma_k| = |\sigma_{k+1}|$ it prints 0. That is, M conjectures 1 when M_0 changes its conjecture, and conjectures zero at all other times.

Since M_0 projects f, at any input n, M will eventually be stable with a projection of zero. Let g be any stability measure for M. Then g is nonrecursive. For it is clear that one could compute a stability measure for M_0 relative to g. Thus, if g were recursive, M_0 would possess a recursive stability measure. By Corollary 9, then, f would be recursive, contradicting the assumption.

It is natural at this point to ask whether, if f: $N \to N$ is a projectable function, there *exists* an extended machine projecting f which is associated with a stability measure recursive relative to f. The following result answers this question in the negative:

Theorem 11 *Let f: $N \to N$ be a projectable function. Then there is an extended Turing machine M and a stability measure g for M such that M pro-*

jects f and g is recursive relative to f iff f is Turing equivalent to a recursively enumerable set.

Proof: For the left to right direction, let M be an extended Turing machine projecting a function f and let g be a stability measure for M. Suppose that g is recursive relative to f. Let g^* be the stability measure for M that maps any n onto the least integer m such that, when M is given n as input, M is stable in m steps. Clearly, g^* is recursive relative to g. Since g is recursive relative to f, g^* is recursive relative to f. Also, by Theorem 8, f is recursive relative to g^*. Thus, f is Turing equivalent to g^*. Now let

$$A = \{\langle n,m\rangle\text{: at input } n,\ M \text{ is } \textit{not} \text{ stable in } m \text{ steps}\}.$$

Note that A is r.e. For, to enumerate A, we systematically apply M to each input. If, for an input n and stage p, M either halts or changes its conjecture at p, we output $\langle n,m\rangle$ for each $m < p$. Finally, we claim that A is Turing equivalent to g^*, and hence to f. To compute g^* relative to A proceed as follows: at any argument n and stage m, determine whether $\langle n,m\rangle \in A$. If not, output m and halt. Otherwise, go to stage $m + 1$. Conversely, to determine whether any $\langle n,m\rangle$ belongs to A, one need only determine whether $m < g^*(n)$, and this relation is clearly recursive relative to g^*.

For the right to left direction, assume that f is Turing equivalent to an r.e. set P. We construct an extended Turing machine that projects f along the lines of the proof of Theorem 1. Let an input n be given. Divide each stage k in the computation into substages Ak, Bk, and Ck. At substage Ak, do k steps in the enumeration of P. At substage Bk, do k steps in the computation of $f(n)$ relative to P. When a truth-value of a sentence of the form "$m \in P$" is required, assign it the value F unless m appeared in the (partial) enumeration of P at substage Ak, in which case the value T is assigned. At substage Ck, determine whether a value for $f(n)$ was produced at stage Bk. If so, that value is projected. Otherwise, project 0 at stage k. As in Theorem 1, it may be shown that M projects f. Moreover, for each n there is a stage k_n such that at substage Bk_n only correct truth-values are assigned to sentences of the form "$m \in P$" and a value for $f(n)$ is determined. This value will be correct and M will be stable at stage k_n. The function h defined for each n by $h(n) = k_n$ is then a stability measure for M. Moreover, h is clearly recursive relative to P and hence, by hypothesis, relative to f.

Corollary 12 *There is a Turing projectable function f such that no extended machine projecting f possesses a stability measure recursive relative to f.*

Proof: Let f be any function recursive relative to the halting problem, but not equivalent to any r.e. set. By Theorem 1, f is Turing projectable. But by Theorem 11, no extended machine projecting f has a stability measure recursive relative to f.

6 Other conditions There are other natural restrictions on the definition of 'projectable function'. In this section, we briefly consider some restrictions which lead to classes of projectable functions properly intermediate between the recursive functions and the collection of all projectable functions.

The conditions considered above involving space and time may be viewed as constraints on the efficiency of projection. Another notion of efficiency for projective strategies is that there exists a uniform finite bound on the number of times a given generalized machine can *change* its projection. Let us say that a generalized machine is *uniform* if there is a number k such that at any input the machine reaches a stable projection with fewer than k changes in its projected value. Say that a function $f\colon N \to N$ is *uniformly projectable* if it is projected by some uniform machine. Under a somewhat different terminology, Putnam investigated the question of uniform projectability for characteristic functions (see [3], especially Theorem 2). He found that such a function is uniformly projectable if and only if the associated predicate belongs to Σ_1^*, the smallest set containing the r.e. predicates and closed under truth-functions. Thus the collection of such predicates falls short of the collection of all effectively projectable predicates.

A rather different notion of efficiency for projective strategies concerns the number of times a given projection is used. Let M be an extended machine. Say that M is *perfect* if at any input M projects any value at most once. If a function $f\colon N \to N$ is projected by a perfect machine, we say that f is *perfectly projectable.*

One class of perfectly projectable functions is of special interest. Let R be a partial ordering of the natural numbers. Let M be an extended Turing machine. Say that M is *monotonic in* R if for any argument the projected values of M appear in (strictly) increasing order with respect to R. Thus, for example, if R is the natural order ($\leq$) on N, then a machine is monotonic in R if for any successive projections n and m of M, we have $n < m$. Monotonicity in $\leq$ may be thought of as the requirement on an extended Turing machine that no erasing instruction be executed on the projection tape.[6] A number-theoretic function is *monotonically projectable in* R iff some machine monotonic in R projects it. It is clear that if a machine is monotonic in some order, then it is perfect provided that it projects a function.

Our first result circumscribes the perfectly projectable functions:

Theorem 13 *Let $f\colon N \to N$. If f is perfectly projectable, then f is Turing equivalent to a recursively enumerable set.*

Proof: Let M be a perfect extended machine projecting f. We show that f is Turing equivalent to an r.e. set. In view of Theorem 11, it suffices to show that M has a stability measure recursive relative to f. We may compute a stability measure relative to f as follows: for any n, count the number of steps until $f(n)$ is projected, then output that number. That this procedure computes a stability measure for M follows immediately from the fact that M is perfect.

In view of Theorems 1 and 13, there exist projectable functions f which are not perfectly projectable: take f to be any function recursive relative to the halting problem, but not Turing equivalent to any recursively enumerable set. Thus, since it is clear that any recursive function is perfectly projectable, the perfectly projectable functions are properly intermediate between the recursive functions and the class of all projectable functions.

Our final result shows that Theorem 13 cannot be reversed:

Theorem 14 *There is a function f: $N \to N$ such that f is equivalent to an r.e. set and f is not perfectly projectable.*

Proof: Let $A(n,k)$ say that n is the Gödel number of an extended machine M such that M at input n discards at least one projection by stage k. Define g: $N \times N \to N$ by

$$g(n,k) = \begin{cases} 0 \text{ if } A(n,k) \\ 1 \text{ otherwise.} \end{cases}$$

Define f by $f(n) = \lim_{k \to \infty} g(n,k)$. Then f is not projected by any perfect extended machine (on the convention that each machine begins with an initial projection of 0). But f is clearly Turing equivalent to the predicate $(\exists x)A(n,x)$, which is r.e., since $A(n,x)$ is recursive.

NOTES

1. Similar generalizations have been considered before. In particular, the notion of 'effectively projectable set' introduced in Section 1 of this paper is equivalent to the concept of 'trial and error predicate' in Putnam's [3]; see also Jeroslow [2].

2. For simplicity, we consider only total functions here, but most of the results and arguments presented throughout readily generalize to partial functions. Here, the relevant statement for partial functions is that the output of an effective or mechanical procedure is defined only for inputs on which it terminates.

3. Of course, we wish to allow the case in which $\{\sigma_{im}\}$ terminates with some stage σ_{km}. In this case, we set $\sigma_{im} = \sigma_{km}$ for all $i > k$.

4. As with ordinary Turing machines, virtually any reasonable variation on this theme will suffice. For definiteness, we propose the following: The *alphabet* of each extended Turing machine is the familiar universal one consisting of blanks (or zeros) and ones, with a natural number n represented by a string of n ones. Let $q_0, q_1, \ldots$ be the computation states and $r_0, r_1, \ldots$ the projection states. The "interaction" between the tapes is effected by two distinguished states q_0, r_0, and an instruction $(q_0,1,0,r_0)$: when M is in the state q_0 and scans a one on the computation tape, it erases the character and assumes the distinguished projection state r_0. This state causes the machine to perform a subroutine which erases the projection tape and throws M back into state q_0, scanning the original space on the computation tape. At this point, M transcribes part of the computation tape to the projection tape by proceeding from left to right, transcribing each '1' until a blank space is encountered.

5. In this case, one must slightly extend the notion of projection, to allow finite sequences of '1's and blanks to be projected values.

6. Incidentally, there is a similarity in spirit here to the *original* Turing machines (of Turing's seminal "On computable numbers, with an application to the Entsheidungs-problem" [5]). Those machines were thought of as devices for enumerating sets. Every second square of the tape is designated for output and no erasing instructions are executed on those squares.

REFERENCES

[1] Carnap, R., *The Continuum of Inductive Methods*, University of Chicago Press, Chicago, 1952.

[2] Jeroslow, R., "Experimental logics and Δ_2^0-theories," *Journal of Philosophical Logic*, vol. 4 (1975), pp. 253–267.

[3] Putnam, H., "Trial and error predicates and the solution to a problem of Mostowski," *The Journal of Symbolic Logic*, vol. 30 (1965), pp. 49–57.

[4] Putnam, H., " 'Degree of confirmation' and inductive logic," reprinted in his *Mathematics, Matter and Method*, Cambridge Unversity Press, Cambridge, 1975.

[5] Turing, A. M., "On computable numbers, with an application to the Entsheidungsproblem," *Proceedings of the London Mathematical Society*, series 2, vol. 42 (1936–1937), pp. 230–265.

T. McCarthy
Department of Philosophy
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

S. Shapiro
Department of Philosophy
The Ohio State University
Newark, Ohio 43055-9990