Infinitary Self Reference in Learning Theory [1]

John Case
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716
USA

Email: case@cis.udel.edu
Phone: 1-(302) 831-2711

[1]This paper has now appeared as Case (1994).

Abstract

Kleene's Second Recursion Theorem provides a means for transforming any program p into a program $e(p)$ which first creates a quiescent self copy and then runs p on that self copy together with any externally given input. $e(p)$, in effect, has complete (low level) self knowledge, and p represents how $e(p)$ uses its self knowledge (and its knowledge of the external world). Infinite regress is not required since $e(p)$ creates its self copy outside of itself. One mechanism to achieve this creation is a self replication trick isomorphic to that employed by single-celled organisms. Another is for $e(p)$ to look in a mirror to see which program it is. In 1974 the author published an infinitary generalization of Kleene's theorem which he called the Operator Recursion Theorem. It provides a means for obtaining an (algorithmically) growing collection of programs which, in effect, share a common (also growing) mirror from which they can obtain complete low level models of themselves and the other programs in the collection. These and other recursion theorems have found many applications in, among other domains, Gold style computational learning theory. Several examples are presented and explained with the intention to teach the use of infinitary (and other) recursion theorems in learning theory. There is also some discussion about the insights those proofs provide.

1 Introduction

Here is a brief outline of this paper.

In Section 2 we present useful background information about Gold style computational learning theory (see Gold (1967), Blum and Blum (1975), Klette and Wiehagen (1980), Case and Smith (1983), Angluin and Smith (1983), and Osherson, Stob and Weinstein (1986)) and some of the recursive function theory on which it depends.

In Section 3 we present carefully motivated informal and formal statements of Kleene's Second Recursion Theorem (Rogers (1967), Page 214) together with some simple, illustrative applications including one from Gold style computational learning theory. This examples are also used to show how self reference arguments give especial insight into the truths so proved.

In Section 4 we present carefully motivated informal and formal statements of the author's Operator Recursion Theorem (Case (1974)), an infinitary version of Kleene's Second Recursion Theorem. Three results from Gold style computational learning theory are proved using the Operator Recursion Theorem as a principal tool. The first was chosen for its simplicity and the remaining two for their aesthetic subtlety and illustration of two major trends in such proofs.

2 Preliminaries

N denotes the set of natural numbers, $\{0, 1, 2, 3, \ldots\}$.

φ denotes a fixed *acceptable* programming system for the partial computable functions: $N \to N$ (see Rogers (1958), Rogers (1967), Machtey and Young (1978), Riccardi (1980), Riccardi (1981), Royer (1987), and Marcoux (1989)). φ_p denotes the partial computable function computed by program p in the φ-system. All "naturally" occurring programming systems (e.g., those based on Turing machines, Lisp, Pascal, ...) are essentially acceptable, but non-acceptable systems (with self-interpreters) can be mathematically constructed (Rogers (1958)).

$\langle \cdot, \cdot \rangle$ denotes a fixed *pairing function* (a computable, bijective mapping: $N \times N \to N$) (see Rogers (1967)). $\langle \cdot, \cdot \rangle$ and its inverses are useful to simulate having multiple argument partial functions computable in the φ-system.

The set of all partial computable functions mapping N to N is denoted by $\mathcal{P}$. The set of all (total) computable functions mapping N to N is denoted by $\mathcal{R}$.

Φ denotes an arbitrary fixed Blum complexity measure (see Blum (1967a) and Hopcroft and Ullman (1979)) for the φ-system. Intuitively, $\Phi_p(x)$ denotes the amount of computational resources φ-program p uses on input x before, if ever, halting.

Imagine an algorithmic device (machine) M which takes as its input the graph of an arbitrary $\psi \in \mathcal{P}$, in any order, and, which, from time to time, as it is receiving its input, outputs a corresponding sequence of programs $p_0, p_1, p_2, \ldots$ from the φ system (see (1) just below).

$$p_0, p_1, p_2, \ldots \leftarrow [M] \leftarrow \text{graph of } \psi \text{ in some order.} \tag{1}$$

In the context of Gold style computational learning theory one hopes for some connection between the programs eventually output and ψ, but whether there is one will depend on the difficulty of ψ and the cleverness of M. Such M will be referred to as *learning machines*.

For σ a finite sequence of (pauses and) data points from $\psi \in \mathcal{P}$, $M(\sigma) \stackrel{\text{def}}{=}$ the last program output by M on σ from the time M begins to read σ until it finishes — we assume that reading time increases with the length of σ. For convenience we will suppose M initially outputs before it has seen any pauses or data points. In this way we make sure $M(\sigma)$ is always defined.

A set is said to have size $\leq * \stackrel{\text{def}}{\Leftrightarrow}$ it is finite. Suppose $a \in N \cup \{*\}$. '$=^a$' means '= except on a set of $\leq a$ counterexamples'.

Various criteria have been studied regarding what it means for a learning machine M to learn successfully (or infer) programs for ψ's $\in \mathcal{P}$. Here are some examples essentially from Blum and Blum (1975) and Case and Smith (1983), where the quantifier '$\stackrel{\infty}{\forall}$' means 'for all but finitely many'. '$\stackrel{\infty}{\exists}$' stands for 'there exists infinitely many'.

Definition 1

(a) We say M **Ex**^{a}-identifies $\psi \stackrel{\text{def}}{\Leftrightarrow}$ *M fed ψ in any order outputs a corresponding sequence of programs $p_0, p_1, p_2, \ldots$ such that $(\exists p)(\stackrel{\infty}{\forall} n)[p = p_n \ \wedge \ \varphi_p =^a \psi]$.*

(b) **Ex**^{a} *is defined to be the set of all sets $\mathcal{C}$ of partial computable functions such that some learning machine M* **Ex**^{a}*-identifies each partial function in $\mathcal{C}$.*

It is easy to argue, for example, that a suitable learning machine M employing standard interpolation techniques (see, for example, Kopal (1961)) can **Ex**^{0}-identify each polynomial (of one variable with coefficients in N), i.e., that a suitable M witnesses that the set of such polynomials is in **Ex**^{0}.

Here is another class of example criteria (from Barzdin (1974) and Case and Smith (1983)).

Definition 2

(a) We say M **Bc**^{a}-identifies $\psi \stackrel{\text{def}}{\Leftrightarrow}$ *M fed ψ in any order outputs a corresponding sequence of programs $p_0, p_1, p_2, \ldots$ such that $(\stackrel{\infty}{\forall} n)[\varphi_{p_n} =^a \psi]$.*

(b) **Bc**^{a} *is defined to be the set of all sets $\mathcal{C}$ of partial computable functions such that some learning machine M* **Bc**^{a}*-identifies each partial function in $\mathcal{C}$.*

'$\subset$' denotes 'proper subset'. It is shown in Case and Smith (1983) (with special cases from Barzdin (1974) and Blum and Blum (1975)) that

Theorem 1

(a) $\mathbf{Ex}^0 \subset \mathbf{Ex}^1 \subset \mathbf{Ex}^2 \subset \cdots \subset \mathbf{Ex}^* \subset \mathbf{Bc}^0 \subset \mathbf{Bc}^1 \subset \mathbf{Bc}^2 \subset \cdots \subset \mathbf{Bc}^*$. *Moreover, the properness of each subset relation is witnessed by classes* $\subseteq \mathcal{R}$.

(b) **(Harrington)** $\mathcal{R} \in \mathbf{Bc}^*$.

Interesting extensions of Theorem 1 may be found in Case, Chen and Jain (1992).

In the next section we introduce a result of Kleene's which yields the existence of useful self referential (or self reflecting) programs.

3 Kleene Recursion Theorem

We begin with an informal statement of Kleene's Second Recursion Theorem (Rogers (1967), Page 214). In this informal statement, a *complete low level self model* of a program is just a copy of that program. Higher level self models might also include true sentences about the complete low level self model. (Case, Jain and Sharma (1992) and Baliga, Case, Jain and Suraj (1992), for example, are concerned with the *learning* of higher level information.) Of course the set of *all* such true sentences in languages at least as expressive as first order arithmetic Mendelson (1986) is not algorithmically generable. In this paper we restrict our attention to complete low level self models which, quite interestingly, *can* be obtained algorithmically.

Theorem 2 (Informal Kleene Recursion Theorem) *For any algorithmic thing one would want to compute about a complete low level self model, one can algorithmically find a program e which creates an* external, *complete low level self model and subsequently computes that thing about that self model. It is understood that e's computation may depend on its input from the external world too.*

The following is a formal statement of Kleene's Theorem for programs in the φ-system.

Theorem 3 (Kleene Recursion Theorem for φ) $(\forall p)(\exists e)(\forall x)[\varphi_e(x) = \varphi_p(\langle e, x\rangle)]$. *Moreover, example such φ-programs e can be algorithmically found from the p's on which they depend.*

In Theorem 3, p represents an arbitrary algorithmic thing one could want to compute about a complete low level self model. e is a corresponding program which computes that thing about its complete low level self model (and its input x from the external world). Intuitively, p is the *use* e makes of its low level self knowledge (and knowledge of its external input). Notationally, using '$\langle e, x\rangle$' as the input to p merely provides a convenient way of having the input to p code both the self copy of e and the external input x.

We omit the standard proof of Theorem 3 (Rogers (1967), Page 214), but note that Case (1974) contains an analysis of that proof showing a special case of it to be roughly isomorphic to von Neumann's construction of a self-reproducing automaton (see Neumann (1966), Myhill (1964), Codd (1968), and Burks (1970)). This latter is clearly isomorphic to the essential mechanism of self-reproduction employed by single-celled biological organisms. One can conceptualize the proof of Theorem 3 as providing the composition of a self copying operation followed by application of p to the self copy and the external input. The standard proof of Theorem 3 is also roughly isomorphic to that of Feferman's Lemma (Feferman (1960)) which captures the essence of Gödelian self referential *sentences* (Gödel (1986)). N.B. Theorem 3 should not be confused with the pseudo fixed-point variant popularized by Rogers in Rogers (1967), Page 180 (for discussion see Riccardi (1980), Riccardi (1981), Riccardi (1982), Royer (1987), Royer and Case (1986), Case (1991), and Royer and Case (1994)).

Figure 1 provides a graphical representation of Theorem 3. The robot pictured is facing an (expandable) board, a little to its right, and a mirror, a little to its left. It has a transparent front so that its complete underlying mechanism, flowchart, or program can be seen. The board serves for input, output, and intermediate computations. For the robot to obtain its complete low level self model, it has only to look in the mirror. It is depicted already having copied onto the board a portion of its "program" (corrected for mirror reversal). Note that the robot's self copy is projected *externally* to the robot itself. In this way infinite regress is not required for the robot to have *complete* low level self knowledge. The use this robot makes of its complete low level self knowledge is whatever was built into it, but it could have been anything algorithmic.

Just before the beginning of this section (Section 3), we suggested that self referential programs are useful. We present in the proof of the following theorem (Theorem 4) a very succinct application of Theorem 3. Theorem 4 is a variant of a theorem from Rogers (1967), Page 70. Theorem 4 says that the cardinality of finite sets is not computably bounded in the *size* of decision programs for those finite sets. Applications exist in learning theory too of self reference arguments applied to problems about program size (see, for example, Kinber (1977), Chen (1981), Chen (1982), Case, Jain and Sharma (1989), Case, Jain and Sharma (1990), and Case, Jain and Sharma (1992)), but we do not go into them here.

For $A \subseteq N$, we use χ_A to denote the characteristic function of A, the function which is 1 on A and 0 off A. $\mathrm{card}(A)$ denotes the cardinality of A.

Theorem 4 $(\forall$ *computable* $h)(\exists$ *φ-program e and finite set* $D \subseteq N \mid \varphi_e = \chi_D)[\mathrm{card}(D) > h(e)]$.

PROOF. Suppose h is computable. By Theorem 3 there is a self reflecting *φ-program* e such that

$$\varphi_e = \chi_{\{x|x \leq h(e)\}}. \tag{2}$$

e creates a self copy, computes h of that self copy (i.e., it computes $h(e)$), and then it decides membership in the finite set $\{x \mid x \leq h(e)\}$. Clearly, the cardinality of this finite set is $h(e)+1$, which *is* $> h(e)$.

$\square$

To obtain formally the program e in the proof of Theorem 4 just above, first let p be a φ-program such that, for all w, x,

$$\varphi_p(\langle w, x \rangle) = \chi_{\{x|x \leq h(w)\}}. \tag{3}$$

Such a p clearly exists since the right hand side of (3) trivially defines a computable function of arguments w and x. The existence of e satisfying (2) then follows by substitution into Theorem 3.

One use of self referential programs is to provide very succinct, elegant, insightful proofs of theorems. Succinct and elegant should be clear in the case of the proof of Theorem 4 just above. It is useful to tease out the insight offered by this proof. One can think of the proof as involving a two move, two player game. (Lachlan (1970), Page 307, notes that his infinite game formulation of complex recursion theoretic constructions seems to interact well with the recursion theorem.) Player I plays h and player II plays e. Player II's goal is for e to decide a finite set containing more elements than $h(e)$. The proof provides a winning strategy for player II. Player II's move, e, uses its self knowledge to see what h will do to e itself. It can then "tunnel out" of player I's move, h, by deciding a set with (one) more than $h(e)$ elements in it. Many recursion theorem based proofs have this tunneling out character to them; self knowledge enables a program e to see how some process would transform e itself and then use that information to "defeat" that process. One reason, then, for the truth of Theorem 4 is that, for any computable h, a corresponding program e can figure out how to decide a finite set with $h(e)+1$ elements in it — a pretty simple reason for the truth of the theorem. Empirically, proofs employing self referential programs tend to be simplest known reasons for the truth of what is proven. In Western philosophy self knowledge is generally thought to be important and useful. *A* use is for succinct tunneling out from transformations of self.

Smith (1982a) is concerned with applications of functional programs which reflect on their own traces, an interesting *special case* of what programs may do with a complete low level self model.

One of the motives the author of the present paper had in directing the dissertations Riccardi (1980) and Royer (1987) was eventually to characterize mathematically the utility of (machine) self knowledge. In Theorem 3 the computable transformations between p's and suitable corresponding e's essentially determine a non-denotational control structure. A sample interesting result from Royer (1987) is that, as control structures, if-then-else and Kleene Recursion Theorem are complementary: neither can be constructed solely from the other, but together they yield an acceptable system, and, hence, by a result of the author's in Riccardi (1980), Riccardi (1981), and Royer (1987), together they enable the construction of all possible control structures. Therefore, the construction power of the control structure for decision branches is a *complement* of that of the control structure for self knowledge.

We next proceed to provide an application of machine self reference to learning theory, which application also features tunneling out.

Theorem 1 above has the following

Corollary 1 (Gold (1967) and Blum and Blum (1975))
$(\forall M)(\exists p \mid \varphi_p$ *is total*$)[M$ *fails to* $\mathbf{Ex}^0$*-identify* $\varphi_p]$.

It is shown in Case and Smith (1983) that the p's in Corollary 1 cannot be found algorithmically from the M's on which they depend; in fact,

Theorem 5 (Case and Smith (1983))
$\neg(\exists$ *computable* $f)(\forall M)[M$ *fails to* $\mathbf{Ex}^0$*-identify* $\varphi_{f(M)}]$.

What if algorithms were allowed to change their minds about their outputs finitely many times per input? Would, for example, Theorem 5 still hold? We answer this with Theorem 6 below, but first we provide the definition of *lim-computable* function to capture enough of the informal idea of allowing algorithms to change their minds.

Definition 3 *For* $g : (N \times N) \to N$ *and* $x \in N$, $g(x, \infty) \stackrel{\mathrm{def}}{=} \lim_{t\to\infty} g(x, t)$.

Definition 4 $f : N \to N$ *is* lim-computable $\stackrel{\mathrm{def}}{\Leftrightarrow}$ $(\exists$ *computable* $g : (N \times N) \to N)(\forall x)[f(x) = g(x, \infty)]$.

Intuitively, in Definition 4, $g(x, t)$ is the output at discrete time t of the mind changing algorithm for f (acting on input x). $(\forall x)[f(x) = g(x, \infty)]$, for g computable, means, then, that, for all x, for all but finitely many times t, the output of the mind changing algorithm on input x is $f(x)$.

It is easy to show that there is a lim-computable function h such that ($\forall$ computable g)$(\stackrel{\infty}{\forall} x)[h(x) > g(x)]$. Hence, the lim-computable functions go way beyond the computable ones; in fact, they have been known since Post (Shapiro (1971)) to characterize the functions computable with an oracle for the halting problem (see also Shoenfield (1959), Gold (1965), Putnam (1965), Shoenfield (1971), and Soare (1987)). Nonetheless, we have the following

Theorem 6 $\neg(\exists$ lim-computable $f)(\forall M)[M$ *fails to* $\mathbf{Ex}^0$*-identify* $\varphi_{f(M)}]$.

PROOF. Suppose for contradiction otherwise, where, for computable g, $f = g(\cdot, \infty)$. By a version of Theorem 2 *which applies to the system of learning machines* (instead of to φ), there is a learning machine E such that E, on any input partial function ψ, ignores ψ, *creates a self copy*, and successively computes and outputs $g(E,0), g(E,1), g(E,2), \ldots$. Clearly, the final program that E eventually outputs (over and over again) is $f(E)$. Therefore, when E is fed $\varphi_{f(E)}$, its final program, $f(E)$, *does* compute $\varphi_{f(E)}$, a contradiction. Hence, E defeats the (lim-computable) transformation f.

$\square$

It is natural to ask if Theorem 4 above can be extended to lim-computable functions. We have the following

Theorem 7 $(\forall$ *lim-computable* $h)(\exists$ φ*-program* e *and finite set* $D \subseteq N \mid \varphi_e = \chi_D)[\mathrm{card}(D) > h(e)]$.

PROOF. Suppose h is lim-computable. Hence, for some computable g, $h = g(\cdot, \infty)$. By Theorem 3, there is a self reflecting program e such that

$$\varphi_e = \chi_{\{x \mid \mathrm{card}(\{w<x \mid \varphi_e(w)=1\}) \leq g(e,x)\}}. \tag{4}$$

e, on x, creates a self copy and uses that self copy

(a) to make a recursive call on itself to see how many elements $w < x$ it has already decided are in the set it decides *and*

(b) to compute $g(e,x)$, an approximation to $h(e)$.

If e sees that, on input x, it hasn't put more than $g(e,x)$ elements $w < x$ into the set it decides, it puts x in that set, and, otherwise, it puts x in the complement. For all sufficiently large x, $g(e,x)$ will be $h(e)$. Hence, e will eventually put at least one more than $h(e)$ elements in the set it decides, yet it won't put in more elements than $1 + \max(\{g(e,x) \mid x \in N\}) < \infty$. Hence, e defeats h.

$\square$

Amtoft, Nikolajsen, Träff and Jones (1989) and Jones (1992) deal with *efficient* implementation of recursive calls by self reflecting programs.

The present author is intrigued that succinct self reference arguments can be used to prove results about relative program succinctness such as Theorems 2 and 4. For more such results, also not directly in the context of learning theory, see Blum (1967b), Meyer and Bagchi (1972), Meyer (1972), Royer and Case (1986), and Royer and Case (1994).

4 Infinitary Self Reference

The Operator Recursion Theorem (Case (1974)) has been used to prove many results in Gold style computational learning theory; for example, see Case and Smith (1978), Smith (1982b), Case and Smith (1983), Daley (1984), Daley (1988), Case, Jain and Sharma (1989), Fulk and Jain (1989), Freivalds, Smith and Velauthapillai (1989), Case, Jain and Sharma (1990), Jain and Sharma (1990), Case, Jain and Sharma (1992), Case, Chen and Jain (1992), Jain and Sharma (1992b), Fulk and Jain (1990), Case, Jain and Sharma (1994), Baliga, Case, Jain and Suraj (1992), Baliga, Jain and Sharma (1992), Case, Jain and Ngo Manguelle (1994), Case (1992), and Baliga, Case and Jain (1993).

Theorem 8 (Informal Operator Recursion Theorem) *For any algorithmically given algorithmic things one would want the programs in an infinite r.e. sequence of* pairwise distinct *programs to compute about their complete low level self models* and those of the other programs in the sequence*, one can algorithmically find an r.e. sequence of pairwise distinct programs each of which computes the desired algorithmic thing about itself and the others. It is understood that the computation of each program in the sequence may depend on its input from the external world too.*

Intuitively, one can think of Theorem 8 as providing (algorithmically) growing sequences of pairwise distinct programs $e_0, e_1, e_2, \ldots$ sharing a common (also growing) mirror in which each can eventually see (without obstruction) itself and any of the others. The use each e_n makes of the complete low level knowledge of $e_0, e_1, e_2, \ldots, e_n, e_{n+1}, \ldots$ is whatever was built into it, but it could have been anything uniformly algorithmic in n.

In the proof of Theorem 9 below we present a world's simplest application of Theorem 8 before we present its formal statement a little further below in Theorem 11. The reader may see that there are clearly several ways in which to eliminate the use of Theorem 8 in the proof of Theorem 9, but we believe the proof given below is at least slightly conceptually cleaner than obvious alternatives, and it does quite simply and nicely illustrate Theorem 8.

In what follows, it is useful to think of partial functions as identified with the sets of ordered pairs from their graphs. Then, for example, it will make sense to compare partial functions by '$\subseteq$' or '$\subset$'.

A footnote in Blum and Blum (1975) claims that, if f is a computable function $\mathbf{Ex}^0$-identified by a learning machine M, then

$$\operatorname{card}(\{\varphi_p \subseteq f \mid \varphi_p \text{ is } \mathbf{Ex}^0\text{-identified by } M\}) < \infty \tag{5}$$

Actually,

Theorem 9 *There are computable f and learning machine M so that (5) above fails.*

Proof. By Theorem 8, there is an infinite r.e. sequence of (pairwise distinct) programs $e_0, e_1, e_2, \ldots$ such that, for any n, x,

$$\varphi_{e_n}(x) = \begin{cases} e_x, & \text{if } x \geq n; \\ \text{undefined}, & \text{if } x < n. \end{cases} \tag{6}$$

Clearly, if we write e for the *function* which on input n has value e_n, we have

$$\cdots \subset \varphi_{e_2} \subset \varphi_{e_1} \subset \varphi_{e_0} = e. \tag{7}$$

Let M on any partial function ψ output $\psi(n)$, where n is the least number it has seen so far in the domain of ψ. Clearly this M $\mathbf{Ex}^0$-identifies each partial function in $\{\ldots, \varphi_{e_2}, \varphi_{e_1}, \varphi_{e_0} = e\}$ an infinite collection of subfunctions of e (including e). Hence, (5) fails for $f = e$.

□

If M $\mathbf{Ex}^0$-identifies f, the subfunctions of f which are $\mathbf{Ex}^0$-identified by M *are* restricted, but according to the following theorem (Theorem 10). F in this theorem plays the role of a spoiler. This theorem is, for example, a direct corollary of Theorem 1 from Case (1992).

Theorem 10 *Suppose $I \in \{\mathbf{Ex}^0, \mathbf{Bc}^0\}$ and that M I-identifies f. Then*

$$(\exists \textit{ finite } F \subset f)[\{\varphi_p \textit{ I-identified by } M \mid F \subseteq \varphi_p \subset f\} = \emptyset]. \tag{8}$$

Regarding a formal statement of the Operator Recursion Theorem, we need some way to express the preassigned algorithmic dependence of each *program* e_n on $e_0, e_1, e_2, \ldots$ (i.e., on the *function* e), on n, and on its externally given input x. Recursive operators provide a convenient medium for such expression.

Definition 5 Θ *is a* recursive operator $\overset{\text{def}}{\Leftrightarrow}$

(a) Θ maps the set of all partial functions into itself, and

(b) *there is an algorithm which converts any listing of any input* ψ *to* Θ *into some corresponding listing of the output* $\Theta(\psi)$.

Rogers (1967) provides an excellent and extensive discussion of recursive operators and their characterization. Here are three example recursive operators.

Example 1 *Each of* Θ_0, Θ_1, *and* Θ_2 *defined below are clearly recursive operators.*

(a) $\Theta_0(\psi) = \psi \circ \psi$

(b) $\Theta_1(\psi)(y) = \psi(y+1)$

(c)

$$\Theta_2(\psi)(\langle n, x\rangle) = \begin{cases} \psi(x), & \text{if } x \geq n; \\ \text{undefined}, & \text{if } x < n. \end{cases} \tag{9}$$

From Case (1974) we have the following result. When e is a function, we write e_n in place of $e(n)$.

Theorem 11 (Operator Recursion Theorem for φ)
$(\forall$ *recursive operators* $\Theta)(\exists$ *a 1-1 computable function* $e)(\forall n, x)[\varphi_{e_n}(x) = \Theta(e)(\langle n, x\rangle)]$. *Moreover, programs for example such 1-1 computable* e*'s can be algorithmically found from programs for the* Θ*'s on which they depend.*

Proofs of Theorem 11 and variations on the theme can be found in Case (1974).

If one applies Theorem 11 to the recursive operator Θ_2 from (9) in Example 1 above, one obtains a 1-1 computable function e such that, for any n, x, (6) above holds.

We present a more complicated application of infinitary self reference in learning theory in the proof of the following theorem (joint with Harrington) from Case and Smith (1983). The present proof contains some simplification suggested by Sanjay Jain. This theorem implies one of the separations indicated in Theorem 1 above.

Theorem 12 *Let* $\mathcal{S} = \{f \in \mathcal{R} \mid (\stackrel{\infty}{\forall} n)[\varphi_{f(n)} = f]\}$. *Then*

(a) $(\exists M)(\forall f \in \mathcal{S})[M\ \mathbf{Bc}^0$*-identifies* $f]$, *yet*

(b) $(\forall M)(\exists f \in \mathcal{S})[M$ *fails to* $\mathbf{Ex}^*$*-identify* $f]$.

That the highly self referential $\mathcal{S}$ from Theorem 12 satisfies clause (a) is immediate. $\mathcal{S}$ was chosen explicitly to make it so. Tunneling out is involved in the proof of clause (b), but it doesn't depend so much on the self-other reference; the self-other reference is necessary to ensure obtaining an f which lands in $\mathcal{S}$. As we will see, it also helps the programs coordinate their efforts so that one of them defeats M. By Theorems 5 and 6 above, we know that, from M, one can't algorithmically, with or without mind changes, create a program defeating M. In the proof just below of Theorem 12, suitable programs from Theorem 11, in effect, cooperate, collaborate, or conspire so that at least one of them computes an f (in $\mathcal{S}$) which defeats M. The more important role of self-other reference in the proof of Theorem 12 is in working with $\mathcal{S}$ which captures *a minimal essence* of $\mathbf{Bc}^0$: if anything is going to separate $\mathbf{Bc}^0$ from $\mathbf{Ex}^*$, $\mathcal{S}$ ought to do it. The use of $\mathcal{S}$ succinctifies the proof of Theorem 12 by making half the proof utterly immediate, while, at the same time, not really complicating the other half. Many of the infinitary and other self reference arguments in Gold style computational learning theory have this character. These are, by the way, existence results for which we would like to have witnessing examples which are natural. It is argued in Case (1988); Case (1992) that self referential examples witnessing existence may presage natural examples witnessing the same.

Proof (Theorem 12). It remains to show clause (b). Suppose M is given. To make sure the f we construct which M fails to $\mathbf{Ex}^*$-identify is in $\mathcal{S}$, we'll put a lot of programs for this f in its range. By Theorem 11, there is an r.e. sequence of pairwise distinct programs $e_0, e_1, e_2, \ldots$ whose I/O behavior is described in a construction by stages below. Intuitively, in stage s, e_0 plays an aggressor role. It watches its relatively passive helpers in that stage, the *distinct* programs e_{2s+1} and e_{2s+2}, which compute two different partial computable functions, to see if M is *sensitive* to the difference between $\varphi_{e_{2s+1}}$ and $\varphi_{e_{2s+2}}$. If so, e_0 will exploit M's sensitivity to force M to change its mind. Were e_0 to succeed in forcing M to change it mind infinitely often, e_0 would reach its goal of computing the desired f. If, past some point, M remains *in*sensitive to the difference between $\varphi_{e_{2s+1}}$ and $\varphi_{e_{2s+2}}$, then e_{2s+1} and e_{2s+2} will reach their goal of having at least one of them compute the desired f.

$\varphi^s_{e_n}$ will denote the finite part of φ_{e_n} defined *before* stage s. For partial functions ψ, $\psi[x]$ denotes ψ restricted to $\{w \mid w < x\}$.

Go to stage 0.

begin stage s;
  set x^s = the least number $\notin$ the domain of $\varphi^s_{e_0}$;
  set $\varphi_{e_{2s+1}}[x^s]$ and $\varphi_{e_{2s+2}}[x^s]$ both $= \varphi_{e_0}[x^s]$;
  set $y = x^s$;
  go to substage 0;
  begin substage t;
    set $\varphi_{e_{2s+i}}(y) = e_{2s+i}$, $i = 1, 2$;
    set $y = y + 1$;

if $M(\varphi_{e_{2s+1}}[y]) \neq M(\varphi_{e_{2s+2}}[y])$

  then $(*$ M *is* sensitive $*)$

    set j = the least i such that $M(\varphi^s_{e_0}) \neq M(\varphi_{e_{2s+i}}[y])$;

    $(*$ i exists since $M(\varphi^s_{e_0})$ can't = both $M(\varphi_{e_{2s+1}}[y])$ and $M(\varphi_{e_{2s+2}}[y])$ $*)$

    set $\varphi_{e_0}[y] = \varphi_{e_{2s+j}}[y]$;

    $(*$ then M is forced to change its mind on φ_{e_0} $*)$

    have e_{2s+j} *emulate* e_0 on all arguments $\geq y$;

    $(*$ *then* e_{2s+j}, in the range of φ_{e_0}, *will be another program for* φ_{e_0} $*)$

    go to *stage* $s+1$

  else $(*$ M is *in*sensitive $*)$

    go to *sub*stage $t+1$

endif

end *sub*stage t

end *stage* s.

Case (1). Each stage s is entered and subsequently terminates. Therefore, φ_{e_0} is total. Furthermore, $\varphi_{e_0} \in \mathcal{S}$ since any e_{2s+j} put into the range of φ_{e_0} is, in this case, explicitly made to compute φ_{e_0}. Moreover, M on φ_{e_0} changes its output infinitely often. Hence, M doesn't $\mathbf{Ex}^*$-identify φ_{e_0}.

Case (2). Some stage s is entered but never subsequently terminates. In this case, $\varphi_{e_{2s+1}}$ and $\varphi_{e_{2s+2}}$ differ at infinitely many arguments and are both in $\mathcal{S}$. Also,

$$(\forall y \geq x^s)[M(\varphi_{e_{2s+1}}[y]) = M(\varphi_{e_{2s+2}}[y])]. \tag{10}$$

Hence, M is *in*sensitive in its final program(s) to the infinite difference between $\varphi_{e_{2s+1}}$ and $\varphi_{e_{2s+2}}$; therefore, it can't possibly $\mathbf{Ex}^*$-identify both of them.

$\square$ (Theorem 12)

Next we motivate our remaining application, in this paper, of Theorem 11.

Suppose $\mathcal{S} \subseteq \mathcal{R}$. Not surprisingly, being able to decide, with a mind changing algorithm, the *equivalence* of programs for functions in $\mathcal{S}$ is linked to being able to learn programs for the functions in $\mathcal{S}$. See, for example, Freivalds (1975), Kinber (1975), Chen (1981), Chen (1982), Freivalds, Kinber and Wiehagen (1984), Case, Jain and Sharma (1989), Jain and Sharma (1991), Jain and Sharma (1992b), and Jain and Sharma (1992a) for such linkages. Deciding the equivalence of programs is clearly equivalent to being able to find canonical, equivalent programs for them. Next is defined a formal notion called *limiting standardizability* capturing the latter idea. '$\downarrow$' means 'converges' or 'is defined to be a finite number'.

Definition 6 $\mathcal{S}$ *is* limiting standardizable *(written:* $\mathcal{S} \in \mathrm{LS}$*)* $\stackrel{\mathrm{def}}{\Leftrightarrow}$ $(\exists$ *recursive* $G)(\forall p \mid \varphi_p \in \mathcal{S})[G(p,\infty)\!\downarrow \ \wedge\ \varphi_{G(p,\infty)} = \varphi_p \ \wedge\ (\forall q \mid \varphi_q = \varphi_p)[G(q,\infty)\!\downarrow = G(p,\infty)]]$.

The following theorem is due to Kinber (see Kinber (1975)). The proof below, however, is due jointly to the author and Jim Royer. It is a relatively succinct, tunneling out construction employing infinitary self-other reference, and it bears resemblance to a proof of a related result in Jain and Sharma (1992b).

Theorem 13 $\mathcal{R} \notin \mathrm{LS}$.

PROOF. Suppose for contradiction that $\mathcal{R} \in \mathrm{LS}$ as witnessed by a computable function G. By Theorem 11, there is an infinite r.e. sequence of φ-programs, $e_0, e_1, e_2, \ldots$ with respective I/O behaviors determined by the following construction in stages. In this construction, e_0 plays the role of an aggressor trying to force φ_{e_0} total but $\neq \varphi_{G(e_0,\infty)}$ whenever it appears that $G(e_0,\infty)$ might $= G(e_{s+1},\infty)$; each e_{s+1} brought into play assumes a passive role of computing a total function and initially agreeing with φ_{e_0}; a new e_{s+1} is brought into play each time e_0 explicitly makes $\varphi_{e_0}(w)\!\downarrow \neq \varphi_{G(e_0,x)}(w)$, for some new w and x.

$\varphi^s_{e_0}$ will denote the finite part of φ_{e_0} defined *before* stage s.

Go to stage 0.

begin stage s;
- set $w^s =$ the least number $\notin$ the domain of $\varphi^s_{e_0}$;
- set $\varphi_{e_{s+1}} = (\varphi^s_{e_0} \cup \{(x,0) \mid w^s \leq x\})$;
- (∗ then $\varphi_{e_{s+1}}$ is total, and $\varphi^s_{e_0} \subset \varphi_{e_{s+1}}$ ∗)
- go to substage 0;
- **begin** substage t;
  - set $x = w^s$;
  - **while** $[G(e_0,x) = G(e_{s+1},x) \ \wedge\ \Phi_{G(e_0,x)}(w^s) > x]$ **do**
    - set $x = x+1$
  - **endwhile**; (∗ $G(e_0,x) \neq G(e_{s+1},x) \ \vee\ \Phi_{G(e_0,x)}(w^s) \leq x$ ∗)
  - **if** $[G(e_0,x) \neq G(e_{s+1},x)]$
    - **then**
      - for each $w \leq x \mid w^s \leq w$ set $\varphi_{e_0}(w) = 0$;
      - set $w^s = x+1$;
      - go to substage $t+1$
    - **else** (∗ $G(e_0,x) = G(e_{s+1},x) \ \wedge\ \Phi_{G(e_0,x)}(w^s) \leq x$ ∗)
      - set $\varphi_{e_0}(w^s) = (1 \dot{-} \varphi_{G(e_0,x)}(w^s))$;
      - go to stage $s+1$

endif

end substage t

end stage s.

Case (1). Each stage s is entered and subsequently terminates. Then φ_{e_0} is total, yet $(\overset{\infty}{\exists}\ w)(\exists x \geq w)[\varphi_{e_0}(w){\downarrow} \neq \varphi_{G(e_0,x)}(w){\downarrow}]$. Therefore, $\varphi_{G(e_0,\infty)} \neq \varphi_{e_0}$ total, a contradiction.

Case (2). Some stage s is entered but never subsequently terminates. Since, for this s, $\varphi_{e_{s+1}}$ is total, $G(e_{s+1},\infty){\downarrow}$ and $\varphi_{G(e_{s+1},\infty)} = \varphi_{e_{s+1}}$.

Subcase (2.1). $G(e_0,\infty){\downarrow} = G(e_{s+1},\infty)$. Hence, $\varphi_{G(e_0,\infty)} = \varphi_{G(e_{s+1},\infty)}, = \varphi_{e_{s+1}}$, which is total. Therefore, stage s eventually terminates, a contradiction to Case (2).

Subcase (2.2). Not Subcase (2.1). Then φ_{e_0} is made $= \varphi_{e_{s+1}}$, which is total. Hence, $G(e_0,\infty){\downarrow} = G(e_{s+1},\infty)$, a contradiction to this subcase.

$\square$

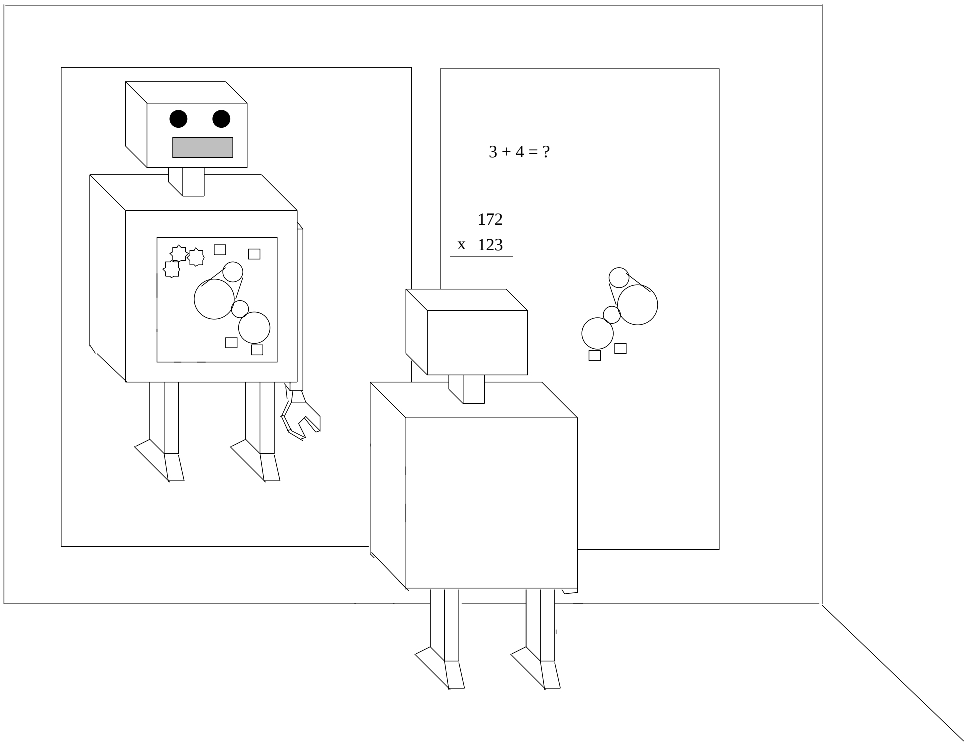

Figure 1. Self Reflecting Robot

References

Amtoft, T., Nikolajsen, T., Träff, J. L., and Jones, N. (1989). Experiments with implementations of two theoretical constructions. In *Logic at Botik*, volume 363 of *Lecture Notes in Computer Science*, (pp. 119–133). Springer-Verlag, Berlin.

Angluin, D. and Smith, C. (1983). A survey of inductive inference: Theory and methods. *Computing Surveys*, *15*, 237–289.

Baliga, G., Case, J., and Jain, S. (1993). Language learning with some negative information. In P. Enjalbert, A. Finkel, K. W. (Ed.), *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science*, volume 665 of *Lecture Notes in Computer Science*, (pp. 672–681). Springer-Verlag, Berlin. Journal version to appear in *Journal of Computer and System Sciences*.

Baliga, G., Case, J., Jain, S., and Suraj, M. (1992). Machine learning of higher order programs. In Nerode, A. and Taitslin, M. (Eds.), *Proceedings of the Second International Symposium on Logical Foundations of Computer Science – Tver '92*, volume 620 of *Lecture Notes in Computer Science*, (pp. 9–20). Springer-Verlag, Berlin. Journal version in press for *Journal of Symbolic Logic*.

Baliga, G., Jain, S., and Sharma, A. (1992). Learning from multiple sources of inaccurate data. In Jantke, K. (Ed.), *Proceedings of the Third International Workshop on Analogical and Inductive Inference*, volume 642 of *Lecture Notes in Artificial Intelligence*, (pp. 108–128). Springer-Verlag, Berlin.

Barzdin, J. (1974). Two theorems on the limiting synthesis of functions. *In Theory of Algorithms and Programs, Latvian State University, Riga*, *210*, 82–88.

Blum, L. and Blum, M. (1975). Toward a mathematical theory of inductive inference. *Information and Control*, *28*, 125–155.

Blum, M. (1967a). A machine independent theory of the complexity of recursive functions. *Journal of the ACM*, *14*, 322–336.

Blum, M. (1967b). On the size of machines. *Information and Control*, *11*, 257–265.

Burks, A. W. (Ed.). (1970). *Essays on Cellular Automata*. Univ. of Illinois Press.

Case, J. (1991). Effectivizing inseparability. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, *37*, 97–111.

Case, J. (1974). Periodicity in generations of automata. *Mathematical Systems Theory*, *8*, 15–32.

Case, J. (1988). The power of vacillation. In Haussler, D. and Pitt, L. (Eds.), *Proceedings of the Workshop on Computational Learning Theory*, (pp. 133–142). Morgan Kaufmann Publishers, Inc. Journal version submitted.

Case, J. (1992). The power of vacillation in language learning. Technical Report 93-08, University of Delaware.

Case, J. (1994). Infinitary self-reference in learning theory. *Journal of Experimental and Theoretical Artificial Intelligence*, *6*, 3–16.

Case, J., Chen, K. J., and Jain, S. (1992). Strong separation of learning classes. *Journal of Experimental and Theoretical Artificial Intelligence*, *4*, 281–293.

Case, J., Jain, S., and Ngo Manguelle, S. (1994). Refinements of inductive inference by Popperian and reliable machines. *Kybernetika*, *30*, 23–52.

Case, J., Jain, S., and Sharma, A. (1989). Convergence to nearly minimal size grammars by vacillating learning machines. In Rivest, R., Haussler, D., and Warmuth, M. (Eds.), *Proceedings of the Second Annual Workshop on Computational Learning Theory, Santa Cruz, California*, (pp. 189–199). Morgan Kaufmann Publishers, Inc. Journal version in press for *Journal of Computer and System Sciences*.

Case, J., Jain, S., and Sharma, A. (1990). Anomalous learning helps succinctness. In Arikawa, S., Goto, S., Ogsuga, S., and Yokomori, T. (Eds.), *Proceedings of the First Workshop on Algorithmic Learning Theory*, (pp. 282–288). Japanese Society for Artificial Intelligence. Tokyo, Japan. Journal article under review.

Case, J., Jain, S., and Sharma, A. (1992). On learning limiting programs. *International Journal of Foundations of Computer Science*, *3*(1), 93–115.

Case, J., Jain, S., and Sharma, A. (1994). Complexity issues for vacillatory function identification. *Information and Computation*. To appear.

Case, J. and Smith, C. (1978). Anomaly hierarchies of mechanized inductive inference. In *Symposium on the Theory of Computation*, (pp. 314–319).

Case, J. and Smith, C. (1983). Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, *25*, 193–220.

Chen, K. (1981). *Tradeoffs in Machine Inductive Inference*. PhD thesis, Computer Science Department, SUNY at Buffalo.

Chen, K. (1982). Tradeoffs in inductive inference of nearly minimal sized programs. *Information and Control*, *52*, 68–86.

Codd, E. F. (1968). *Cellular Automata*. Academic Press.

Daley, R. (1984). On the error correcting power of pluralism in bc-type inductive inference. *Theoretical Computer Science*, *24*, 95–104.

Daley, R. (1988). Transformation of probabilistic learning strategies into deterministic learning strategies. In Haussler, D. and Pitt, L. (Eds.), *Proceedings of the Workshop on Computational Learning Theory*, (pp. 157–163). Morgan Kaufmann Publishers, Inc.

Feferman, S. (1960). Arithmetization of metamathematics in a general setting. *Fundamenta Mathematicae*, *49*, 35–92.

Freivalds, R. (1975). Minimal Gödel numbers and their identification in the limit. *Lecture Notes in Computer Science*, *32*, 219–225.

Freivalds, R., Kinber, E., and Wiehagen, R. (1984). Connections between identifying functionals, standardizing operations, and computable numberings. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, *30*, 145–164.

Freivalds, R., Smith, C., and Velauthapillai, M. (1989). Trade-off among parameters affecting inductive inference. *Information and Computation*, *82*, 323–349.

Fulk, M. and Jain, S. (1989). Learning in the presence of inaccurate information. In Rivest, R., Haussler, D., and Warmuth, M. K. (Eds.), *Proceedings of the Second Annual Workshop on Computational Learning Theory, Santa Cruz, California*, (pp. 175–188). Morgan Kaufmann Publishers, Inc.

Fulk, M. and Jain, S. (1990). Approximate inference and scientific method. In Arikawa, S., Goto, S., Ogsuga, S., and Yokomori, T. (Eds.), *Proceedings of the First Workshop on Algorithmic Learning Theory*, (pp. 256–265). Japanese Society for Artificial Intelligence. Tokyo, Japan.

Gödel, K. (1986). On formally undecidable propositions of Principia Mathematica and related systems I. In S. Feferman (Ed.), *Kurt Gödel. Collected Works. Vol. I* (pp. 145–195). Oxford Univ. Press.

Gold, E. (1965). Limiting recursion. *Journal of Symbolic Logic*, *30*, 28–48.

Gold, E. (1967). Language identification in the limit. *Information and Control*, *10*, 447–474.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory Languages and Computation*. Addison-Wesley Publishing Company.

Jain, S. and Sharma, A. (1990). Language learning by a team. In Paterson, M. S. (Ed.), *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, (pp. 153–166). Springer-Verlag.

Jain, S. and Sharma, A. (1991). Program size restrictions in inductive learning. In *AAAI Symposium Series, Symposium: Machine Learning of Natural Language and Ontology*. Journal version accepted by *Theoretical Computer Science*.

Jain, S. and Sharma, A. (1992a). Characterizing language learning by standardizing operations. *Journal of Computer and System Sciences*. To Appear.

Jain, S. and Sharma, A. (1992b). Learning with the knowledge of an upper bound on program size. *Information and Computation*. To appear; conference version presented at *Workshop on Computational Learning Theory and Natural Learning Systems*, Princeton, N.J., September 1990.

Jones, N. (1992). Computer implementation and applications of Kleene's s-m-n and recursion theorems. In Moschovakis, Y. (Ed.), *Logic From Computer Science*, volume 21 of *Mathematical Science Research Institute Publications*, (pp. 243–263). Springer-Verlag.

Kinber, E. (1975). On comparison of limit identification and limit standardization of general recursive functions. *Uch. zap. Latv. univ.*, *233*, 45–56.

Kinber, E. (1977). On a theory of inductive inference. *Lecture Notes in Computer Science*, *56*, 435–440.

Klette, R. and Wiehagen, R. (1980). Research in the theory of inductive inference by GDR mathematicians – A survey. *Information Sciences*, *22*, 149–169.

Kopal, Z. (1961). *Numerical Analysis*. Chapman and Hall Ltd., London.

Lachlan, A. (1970). On some games which are relevant to the theory of recursively enumerable sets. *Annals of Mathematics*, *91*, 291–310.

Machtey, M. and Young, P. (1978). *An Introduction to the General Theory of Algorithms*. North Holland, New York.

Marcoux, Y. (1989). Composition is almost as good as s-1-1. In *Proceedings, Structure in Complexity Theory–Fourth Annual Conference*. IEEE Computer Society Press.

Mendelson, E. (1986). *Introduction to Mathematical Logic* (Third ed.). Brooks-Cole, San Francisco.

Meyer, A. (1972). Program size in restricted programming languages. *Information and Control*, *21*, 382–394.

Meyer, A. and Bagchi, A. (1972). Program size and economy of description. In *Symposium on the Theory of Computation*.

Myhill, J. (1964). Abstract theory of self-reproduction. In M. D. Mesarović (Ed.), *Views on General Systems Theory* (pp. 106–118). J. Wiley and Sons, New York.

Neumann, J. V. (1966). *Theory of Self–Reproducing Automata*. Univ. Illinois Press. Edited and completed by A. W. Burks.

Osherson, D., Stob, M., and Weinstein, S. (1986). *Systems that Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, Cambridge, Mass.

Putnam, H. (1965). Trial and error predicates and the solution to a problem of Mostowski. *Journal of Symbolic Logic*, *30*, 49–57.

Riccardi, G. (1980). *The Independence of Control Structures in Abstract Programming Systems*. PhD thesis, SUNY Buffalo.

Riccardi, G. (1981). The independence of control structures in abstract programming systems. *Journal of Computer and System Sciences*, *22*, 107–143.

Riccardi, G. (1982). The independence of control structures in programmable numberings of the partial recursive functions. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, *48*, 285–296.

Rogers, H. (1958). Gödel numberings of partial recursive functions. *Journal of Symbolic Logic*, *23*, 331–341.

Rogers, H. (1967). *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York. Reprinted, MIT Press, 1987.

Royer, J. (1987). *A Connotational Theory of Program Structure*. Lecture Notes in Computer Science 273. Springer Verlag.

Royer, J. and Case, J. (1986). Progressions of relatively succinct programs in subrecursive hierarchies. Technical Report 86-007, Computer Science Department, University of Chicago.

Royer, J. and Case, J. (1994). *Subrecursive Programming Systems: Complexity and Succinctness*. Progress in Theoretical Computer Science. Birkhäuser Boston. In press.

Shapiro, N. (1971). Review of "Limiting recursion" by E.M. Gold and "Trial and error predicates and the solution to a problem of Mostowski" by H. Putnam. *Journal of Symbolic Logic*, *36*, 342.

Shoenfield, J. (1959). On degrees of unsolvability. *Annals of Mathematics*, *69*, 644–653.

Shoenfield, J. (1971). *Degrees of Unsolvability.* North-Holland.

Smith, B. (1982a). Reflection and semantics in a procedural language. Technical Report MIT-TR-272, MIT Laboratory for Computer Science.

Smith, C. (1982b). The power of pluralism for automatic program synthesis. *Journal of the ACM*, *29*, 1144–1165.

Soare, R. (1987). *Recursively Enumerable Sets and Degrees.* Springer-Verlag.