Strong Separation of Learning Classes

John Case
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716
email: case@cis.udel.edu
Tel: (302)-831-2711

Keh-Jiann Chen
Institute for Information Sciences
Academica Sinica
Taipei, 15, Taiwan
Republic of China

Sanjay Jain
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716
email: sjain@cis.udel.edu
Tel: (302)-831-1947

March 11, 2007

Abstract

Suppose $\mathbf{LC}_1$ and $\mathbf{LC}_2$ are two machine learning classes each based on a criterion of success. Suppose, for every machine which learns a class of functions according to the $\mathbf{LC}_1$ criterion of success, there is a machine which learns this class according to the $\mathbf{LC}_2$ criterion. In the case where the converse does *not* hold $\mathbf{LC}_1$ is said to be *separated* from $\mathbf{LC}_2$. It is shown that for many such separated learning classes from the literature a much *stronger* separation holds: $(\forall \mathcal{C} \in \mathbf{LC}_1)(\exists \mathcal{C}' \in (\mathbf{LC}_2 - \mathbf{LC}_1))[\mathcal{C}' \supset \mathcal{C}]$. It is also shown that there is a pair of separated learning classes from the literature for which the stronger separation just above does not hold. A philosophical heuristic toward the design of artificially intelligent learning programs is presented with each strong separation result.

1 Introduction

Technically the present paper presents a strengthening of some of the formal learning class hierarchy results mostly from Case and Smith (1983). It is hoped that the particular technical results presented herein can provide one of many possible vehicles to suggest a few philosophical heuristics toward the design of learning programs in artificial intelligence. To that end, we present, with most of our results, brief, corresponding interpretations.

In the present section we'll briefly overview some of the technical results and corresponding interpretations, but first it is helpful to explain what learning classes are and why they are interesting.

We are concerned with the algorithmic learning of programs for functions. The reader may find it interesting that the learning of programs for functions is interpreted in Blum and Blum (1975), Case and Smith (1983) and Fulk (1985) as being about inductively inferring explanations for scientific phenomena. The present paper mostly does not pursue further that interpretation.

We consider learning programs for functions each of whose possible inputs and outputs admits of a finite computer representation. Such inputs and outputs clearly can be numerically coded by numbers in N ($\stackrel{\text{def}}{=} \{0, 1, 2, 3, \ldots\}$). We will, then, consider the learning of programs for functions which map N into N. We could also consider the learning of programs for *partial* functions (Rogers (1967)) mapping N into N (functions with domain a *subset* of N), but, in the present paper, we do not.

Imagine, if you will, an algorithmic device or machine $\mathbf{M}$ receiving as input *all and only* the ordered pairs from the graph of a total function f mapping N into N. Lets say $\mathbf{M}$ receives the ordered pairs in the order

$$(0, f(0)), (1, f(1)), (2, f(2)), \ldots \ . \tag{1}$$

The ordered pairs from the graph of f in any other order could be algorithmically preprocessed into the order in (1), so it will suffice for consideration of mere algorithmicity to consider the order in (1). Further imagine, again if you will, that $\mathbf{M}$ may, as it's receiving the data about f in (1), output, from time to time, a computer program (in some programming system) that $\mathbf{M}$, at each such time, conjectures is a possible program for f. The quality of these conjectures may be expected to depend on the cleverness of $\mathbf{M}$ and the difficulty of f. One way to *define* the *success* of such an $\mathbf{M}$ on such an f (Blum and Blum (1975); Gold (1967)) is to require for success that at *some* time $\mathbf{M}$ receiving the data in (1) about f conjectures some program i such that

$$\mathbf{M} \text{ receiving } \textit{subsequent} \text{ data re } f \text{ never conjectures a program} \neq i, \tag{2}$$

and

$$i \ \textit{does} \text{ compute } f. \tag{3}$$

We say, following the terminology in Case and Smith (1983), that in (and only in) this case $\mathbf{M}$ $\mathbf{Ex}$-*identifies* f. It is interesting to consider the extent to which a machine $\mathbf{M}$ can be a general purpose learner, to what extent it can, say, $\mathbf{Ex}$-identify each function f in a large class of functions. It is easy to argue, for example, that a suitable machine $\mathbf{M}$ employing standard interpolation techniques (Kopal (1961)) can $\mathbf{Ex}$-identify each polynomial (of one variable with coefficients in N). It is known (Gold (1967); Blum and Blum (1975)) that *no* single machine $\mathbf{M}$, however clever, can $\mathbf{Ex}$-identify *each* computable function f. $\mathbf{Ex}$ is defined to be the class of all classes $\mathcal{C}$ of computable functions such that *some* machine $\mathbf{M}$ $\mathbf{Ex}$-identifies *each* function in $\mathcal{C}$. $\mathbf{Ex}$ provides a convenient set theoretic summary of the power of learning (programs for functions) according to the criterion of success, $\mathbf{Ex}$-identification. For example, the class of all polynomials (of one variable and with coefficients in N) is in $\mathbf{Ex}$, and the class of all computable functions is not.

$\mathbf{Ex}$ is our first example of a *learning class*.

We sometimes refer to i conjectured by $\mathbf{M}$ on f after receiving some data about f and satisfying (2) above as the *final program* of $\mathbf{M}$ on f. Blum and Blum (1975) and Case and Smith (1983) in effect introduce the idea that perhaps suitably clever single machines could learn larger classes of functions if the criterion of success were loosened to allow final programs to be slightly erroneous. For $a \in N$, $\mathbf{M}$ is said (Case and Smith (1983)) to $\mathbf{Ex}^a$-*identify* f if and only if $\mathbf{M}$ on f conjectures a final program i and this i computes f correctly except on up to a arguments to f; on these arguments it may be incorrect. $\mathbf{Ex}^a$ is defined to be the class of all classes $\mathcal{C}$ of computable functions such that *some* machine $\mathbf{M}$ $\mathbf{Ex}^a$-identifies *each* function in $\mathcal{C}$. For each $a \in N$, $\mathbf{Ex}^a$ is a *learning class*. Clearly $\mathbf{Ex}^0 = \mathbf{Ex}$. Also, clearly $\mathbf{Ex}^a \subseteq \mathbf{Ex}^{a+1}$. It is shown in Case and Smith (1983) that

$$\mathbf{Ex}^{a+1} - \mathbf{Ex}^a \neq \emptyset. \tag{4}$$

This means that, for each $a \in N$, there is a class of computable functions $\mathcal{C}$ that *some* $\mathbf{M}$ can $\mathbf{Ex}^{a+1}$-identify, but which *no* $\mathbf{M}$ can $\mathbf{Ex}^a$-identify. Allowing the possibility of even one extra error in final programs enables the learning of a class that cannot be learned otherwise. An apparent philosophical heuristic toward the design of learning programs in artificial intelligence already from this result is that one should not be too perfectionistic about final programs and consider allowing errors in them since *each* error allowed may increase learning power. There is a possible problem with this heuristic. What if we care about learning all the functions in a particular class $\mathcal{C}$ known to be in, say, $\mathbf{Ex}^a$, but then we want to learn all the functions in $\mathcal{C}$ *together with even more functions besides* by the device of allowing an error in the final programs. (4) above does *not* guarantee the existence of a class $\mathcal{C}' \supseteq \mathcal{C}$ such that $\mathcal{C}' \in (\mathbf{Ex}^{a+1} - \mathbf{Ex}^a)$. In fact, it might be that, to get the extra learning power from (4) above, we have to give up learning a class $\mathcal{C}$ that we started with in $\mathbf{Ex}$. We may not want to adopt the apparent philosophical heuristic above if the cost of increasing learning power in one direction is a loss in another. Fortunately, in the present paper, for a wide variety of learning classes $\mathbf{LC}_1, \mathbf{LC}_2$ such that $\mathbf{LC}_1 \subseteq \mathbf{LC}_2$ *and* $(\mathbf{LC}_2 - \mathbf{LC}_1) \neq \emptyset$, we show the *strong separation* result that

$$(\forall \mathcal{C} \in \mathbf{LC}_1)(\exists \mathcal{C}' \in (\mathbf{LC}_2 - \mathbf{LC}_1))[\mathcal{C}' \supset \mathcal{C}], \tag{5}$$

where '$\supset$' denotes proper superset. $\mathbf{LC}_1 = \mathbf{Ex}^a$ and $\mathbf{LC}_2 = \mathbf{Ex}^{a+1}$ is an example.

For learning classes $\mathbf{LC}_1$ and $\mathbf{LC}_2$, from the literature, shown to be strongly separated in the present paper, we can and do propose a corresponding philosophical heuristic recommending loosening ones criterion of successful learning in a "direction" given by passing from the definition of $\mathbf{LC}_1$ to that of $\mathbf{LC}_2$.

Here is another example of a criterion of success we consider, this one first introduced in Barzdin (1974) based on a remark in Feldman (1972) and later introduced in Case and Smith (1983). This criterion does *not* require a final program. We say, following the terminology in Case and Smith (1983), that $\mathbf{M}$ $\mathbf{Bc}$-*identifies* f if and only if at *some* time $\mathbf{M}$ receiving the data in (1) about f conjectures some program i such that

$$i \text{ computes } f \tag{6}$$

and

$$\mathbf{M} \text{ receiving subsequent data re } f \text{ never conjectures a program not computing } f. \tag{7}$$

Note that (7) permits $\mathbf{M}$, receiving subsequent data re f, to conjecture lots of programs different from i, *but these programs must also compute* f. $\mathbf{Bc}$ is defined to be the class of all classes $\mathcal{C}$ of computable functions such that *some* machine $\mathbf{M}$ $\mathbf{Bc}$-identifies *each* function in $\mathcal{C}$. '$\subset$' denotes *proper* subset. In Case and Smith (1983) it is shown that, for $\mathbf{LC}_1 = \cup_{a \in N} \mathbf{Ex}^a$ and $\mathbf{LC}_2 = \mathbf{Bc}$, $\mathbf{LC}_1 \subset \mathbf{LC}_2$. In the present paper we, in effect, show that, for this choice of $\mathbf{LC}_1$ and $\mathbf{LC}_2$, the strong separation result (5) above actually holds. In this paper we also consider variants of $\mathbf{Bc}$-identification in which errors are allowed and show strong separation results, each accompanied by a corresponding philosophical heuristic. We also show that a particular pair of learning classes, $\mathbf{LC}_1 = \mathbf{Pex}$ and $\mathbf{LC}_2 = \mathbf{Tex}$, defined below and known to be separated (Case and Ngo Manguelle (1979)), is *not* strongly separated as in (5).

The reader may have noticed that the example heuristics explicitly given or hinted at above do not really tell the designer of learning programs in artificial intelligence exactly what to put in his or her design to achieve the resultant increase in learning power. It is not at all clear from the above what to actually *do* so as to make sure one is not being too narrow about what constitutes success. It turns out that each of our proofs of a strong separation result of the form (5) essentially presents an algorithm for passing from any machine $\mathbf{M}$ which $\mathbf{LC}_1$-identifies a class $\mathcal{C}$ to a machine $\mathbf{M}'$ which $\mathbf{LC}_2$-identifies a class $\mathcal{C}' \supset \mathcal{C}$, where $\mathcal{C}' \notin \mathbf{LC}_1$. Hence, we present algorithms for *example* ways to explicitly strictly improve any design in the direction of the advice from the corresponding philosophical heuristic. We also present some discussion about the potential eventual practicality of such algorithms (just before Proposition 15 below).

2 Preliminaries

2.1 Notation

N denotes the set of natural numbers, $\{0, 1, 2, 3, \ldots\}$. Unless otherwise specified, $e, i, j, m, n, r, s, t, x, y, z$, with or without decorations (decorations are subscripts, superscripts and the like) range over N. $*$ denotes a non-member of N and is assumed to satisfy $(\forall n)[n < * < \infty]$. a and b with or without decorations, range over $N \cup \{*\}$. $\uparrow$ denotes undefined. S, with or without decorations, ranges over subsets of N. $\text{card}(S)$ denotes the cardinality of the set S. $\max(\cdot), \min(\cdot)$ denote the maximum and minimum of their respective set arguments. We assume that $\max(\emptyset) = 0$ and $\min(\emptyset) =\uparrow$.

η and θ range over *partial* functions with arguments and values from N. $\eta(x)\downarrow$ denotes that $\eta(x)$ is defined; $\eta(x)\uparrow$ denotes that $\eta(x)$ is undefined. f, g, p and q with or without decorations range over *total* functions with arguments and values from N. For $n \in N$ and partial functions η and θ, $\eta =^n \theta$ means that $\text{card}(\{x \mid \eta(x) \neq \theta(x)\}) \leq n$; $\eta =^* \theta$ means that $\text{card}(\{x \mid \eta(x) \neq \theta(x)\})$ is finite. $\text{domain}(\eta)$ and $\text{range}(\eta)$ denote the domain and range of the function η, respectively.

$\langle i, j \rangle$ stands for an arbitrary, computable, one to one encoding of all pairs of natural numbers onto N (Rogers (1967)) (we assume that $\langle i, j \rangle \geq \max(\{i, j\})$ and that $\langle \cdot, \cdot \rangle$ is increasing in both its arguments). π_1 and π_2 denote the corresponding inverses: for all x, y, $\pi_1(\langle x, y \rangle) = x$ and $\pi_2(\langle x, y \rangle) = y$.

φ denotes a fixed *acceptable* programming system for the partial computable functions: $N \rightarrow N$ (Rogers (1958); Rogers (1967); Machtey and Young (1978)). (Case showed the acceptable systems are *characterized* as those in which every control structure can be constructed (Riccardi (1980); Royer (1987)); Royer and later Marcoux examined complexity analogs of this characterization (Riccardi (1980); Riccardi (1981); Royer (1987); Marcoux (1989)). Essentially all "naturally" occurring programming systems are acceptable, but non-acceptable systems can be mathematically constructed (Rogers (1958)).) φ_i denotes the partial computable function: $N \rightarrow N$ computed by program i in the φ-system. Thanks to the numerical coding of programs onto N, we identify φ-programs with natural numbers. Φ denotes an arbitrary Blum complexity measure (Blum (1967a); Hopcroft and Ullman (1979)) for the φ-system. Intuitively, $\Phi_i(x)$ is a measure of the resource (for example, time) used by φ-program i on input x. The set of all total recursive functions of one variable is denoted by $\mathcal{R}$. $\mathcal{C}$, with or without decorations, ranges over subsets of $\mathcal{R}$.

A function f is said to be *zero-extension of* $\eta \overset{\text{def}}{\Leftrightarrow}$

$$f(x) = \begin{cases} \eta(x) & x \in \text{domain}(\eta); \\ 0 & \text{otherwise.} \end{cases}$$

ZERO* denotes the set $\{f \mid (\overset{\infty}{\forall} x)[f(x) = 0]\}$.

The quantifiers '$\overset{\infty}{\forall}$' and '$\overset{\infty}{\exists}$', essentially from Blum (1967a), mean 'for all but finitely many' and 'there exist infinitely many', respectively.

2.2 Fundamental Function Inference Paradigms

Let $f[n]$ denote the finite initial segment $((0, f(0)), (1, f(1)), \ldots, (n-1, f(n-1)))$. Let INIT $= \{f[n] \mid f \in \mathcal{R} \wedge n \in N\}$. We let $\mathbf{M}$, with or without decorations, range over learning machines (also called *Inductive Inference Machines*, IIMs (Blum and Blum (1975))).

IIMs have been used in the study of machine identification of programs for recursive functions as well as for algorithmic learning of grammars for languages (Blum and Blum (1975); Case and Smith (1983); Chen (1981); Fulk (1985); Gold (1967); Osherson, Stob and Weinstein (1986); Wiehagen (1978); Angluin and Smith (1983); Klette and Wiehagen (1980); Case (1986)). For language learning, direct analogs of the strong separations of the present paper provably do not hold. Whether suitably and interestingly amended versions do hold remains to be investigated.

For any IIM $\mathbf{M}$, behaving as described in Section 1, we define an associated function $\mathbf{M}_F$: INIT $\rightarrow N$ as follows. Consider $\mathbf{M}$ receiving the graph of f as input, in the order given in (1). Let prog_0 be a program for everywhere 0 function. Let $\mathbf{M}_F(f[n]) \overset{\text{def}}{=} \text{prog}_0$ if $\mathbf{M}$ has not output any program by the time it receives $(n, f(n))$; otherwise let $\mathbf{M}_F(f[n]) \overset{\text{def}}{=}$ the last program output by $\mathbf{M}$ by the time it receives $(n, f(n))$.

From now on, for expository convenience, we will use the '$\mathbf{M}$' to mean either '$\mathbf{M}$' itself or '$\mathbf{M}_F$' with context dictating which is intended.

We now extend the domain of $\mathbf{M}$, the function.

$\mathbf{M}(f)$ is defined to have the value i (written: $\mathbf{M}(f)\downarrow = i$) $\Leftrightarrow$ $(\overset{\infty}{\forall} n)[\mathbf{M}(f[n]) = i]$. $\mathbf{M}(f)$ *is undefined* (written: $\mathbf{M}(f)\uparrow$) $\overset{\text{def}}{\Leftrightarrow}$ no such i exists.

By means of the next five definitions we *formally* specify all the learning classes mentioned or hinted at in Section 1.

Definition 1 (Gold (1967); Blum and Blum (1975); Case and Smith (1983)) Recall that a ranges over $N \cup \{*\}$. For all a,

(a) $\mathbf{M}$ $\mathbf{Ex}^a$*-identifies* a recursive function f (written: $f \in \mathbf{Ex}^a(\mathbf{M})$) iff both $\mathbf{M}(f)\downarrow$ and $\varphi_{\mathbf{M}(f)} =^a f$.

(b) $\mathbf{Ex}^a = \{\mathcal{C} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{C} \subseteq \mathbf{Ex}^a(\mathbf{M})]\}$.

Definition 2 (Case and Smith (1983)) For all a,

(a) $\mathbf{M}$ $\mathbf{Bc}^a$-*identifies* a recursive function f (written: $f \in \mathbf{Bc}^a(\mathbf{M})$) iff $(\overset{\infty}{\forall} n)[\varphi_{\mathbf{M}(f[n])} =^a f]$.

(b) $\mathbf{Bc}^a = \{\mathcal{C} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{C} \subseteq \mathbf{Bc}^a(\mathbf{M})]\}$.

We usually write $\mathbf{Ex}$ for $\mathbf{Ex}^0$ and $\mathbf{Bc}$ for $\mathbf{Bc}^0$. (Barzdin (1974)) essentially introduced the notion of $\mathbf{Bc}^0$. Theorem 3 just below states some of the basic hierarchy results about the $\mathbf{Ex}^a$ and $\mathbf{Bc}^a$ classes.

Theorem 3 *For all n,*

(a) $\mathbf{Ex}^n \subset \mathbf{Ex}^{n+1}$*;*

(b) $\bigcup_{n \in N} \mathbf{Ex}^n \subset \mathbf{Ex}^*$*;*

(c) $\mathbf{Ex}^* \subset \mathbf{Bc}$*;*

(d) $\mathbf{Bc}^n \subset \mathbf{Bc}^{n+1}$*;*

(e) $\bigcup_{n \in N} \mathbf{Bc}^n \subset \mathbf{Bc}^*$*; and*

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

Parts (a), (b), (d), and (e) are due to Case and Smith (1983). John Steel first observed that $\mathbf{Ex}^* \subseteq \mathbf{Bc}$ and the diagonalization in part (c) is due to Harrington and Case (Case and Smith (1983)). Part (f) is due to Harrington (Case and Smith (1983)). Blum and Blum (1975) first showed that $\mathbf{Ex} \subset \mathbf{Ex}^*$. Barzdin (1974) first showed $\mathbf{Ex} \subset \mathbf{Bc}$.

The following definition was based on Popper's principle (Popper (1968)) that scientific conjectures should be refutable.

Definition 4 (Case and Ngo Manguelle (1979)) $\mathbf{M}$ is *Popperian* iff for all f and n, $\varphi_{\mathbf{M}(f[n])}$ is a total function.

We write $f \in \mathbf{Pex}(\mathbf{M})$ to denote the fact that $f \in \mathbf{Ex}(\mathbf{M})$ and $\mathbf{M}$ is Popperian.

Definition 5 (Case and Ngo Manguelle (1979)) $\mathbf{Pex} = \{\mathcal{C} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{C} \subseteq \mathbf{Ex}(\mathbf{M}) \wedge \mathbf{M}$ is Popperian $]\}$.

The following definition requires $\mathbf{M}$ to output programs for total functions on the initial segments of functions it identifies, but not necessarily otherwise.

Definition 6 (Case and Ngo Manguelle (1979))

(a) $\mathbf{M}$ $\mathbf{Tex}$-*identifies* f (written: $f \in \mathbf{Tex}(\mathbf{M})$) iff $\mathbf{M}$ $\mathbf{Ex}$-identifies f and $(\forall n)[\varphi_{\mathbf{M}(f[n])}$ is total $]$.

(b) $\mathbf{Tex} = \{\mathcal{C} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{C} \subseteq \mathbf{Tex}(\mathbf{M})]\}$.

Theorem 7 (Case and Ngo Manguelle (1979)) $\mathbf{Pex} \subset \mathbf{Tex} \subset \mathbf{Ex}$.

3 Strong Separation

First we consider the **Ex** hierarchy.

Theorem 8 $(\forall n)(\exists \mathcal{C})(\forall \mathbf{M})(\exists \mathbf{M}')[[\mathcal{C} \not\in \mathbf{Ex}^n] \wedge [\mathcal{C} \subseteq \mathbf{Ex}^{n+1}(\mathbf{M}')] \wedge [(\mathbf{Ex}^n(\mathbf{M}) - \mathcal{C}) \subseteq \mathbf{Ex}^n(\mathbf{M}')]]$.

Corollary 9 $(\forall n)(\forall \mathcal{C} \in \mathbf{Ex}^n)(\exists \mathcal{C}' \supseteq \mathcal{C})[\mathcal{C}' \in \mathbf{Ex}^{n+1} - \mathbf{Ex}^n]$.

As mentioned in Section 1, the advice suggested by Corollary 9 is to consider allowing even one (more) error in final programs. The proof of Theorem 8 provides one way to algorithmically augment any learning machine to take advantage of this advice.

Proof of Theorem 8. Fix n. Let $\mathbf{M}_0, \mathbf{M}_1, \ldots$ denote a recursive enumeration of all inductive inference machines. We will construct a recursive sequence of programs, $p(0), p(1), p(2), \ldots$ such that, for all i, $\varphi_{p(i)}(0) = i$ and $\mathrm{card}(\{x \mid \varphi_{p(i)}(x)\uparrow\}) \leq n+1$, and, moreover, $\{f \mid \varphi_{p(i)} \subseteq f\} \not\subseteq \mathbf{Ex}^n(\mathbf{M}_i)$. We take $\mathcal{C} = \{f \mid (\exists i)[\varphi_{p(i)} \subseteq f]\}$ (thus $\mathcal{C} \not\in \mathbf{Ex}^n$). Given $\mathbf{M}$ we first show how to construct $\mathbf{M}'$ satisfying the statement of the theorem (assuming the existence of p and $\mathcal{C}$ as defined above). Let $\mathbf{M}'(f[0]) = 0$. For $m > 0$, let $\mathbf{M}'(f[m]) = p(f(0))$ if $\neg[(\exists x < m)[\Phi_{p(f(0))}(x) \leq m \wedge \varphi_{p(f(0))}(x) \neq f(x)]]$; $\mathbf{M}'(f[m]) = \mathbf{M}(f[m])$ otherwise. It is easy to see that $\mathbf{M}'$ satisfies the statement of the theorem. It remains now to construct p as claimed above. The construction of p is a variant of the construction in Case and Smith (1983) proving that $(\mathbf{Ex}^{n+1} - \mathbf{Ex}^n \neq \emptyset)$.

By a suitably padded version of s-m-n theorem (Rogers (1967)) there exists a recursive p, such that the (partial) function $\varphi_{p(i)}$ may be defined as follows. We describe the function $\varphi_{p(i)}$. $\varphi_{p(i)}(0) = i$. Let x_s denote the least x such that $\varphi_{p(i)}(x)$ has not been defined before stage s. Go to stage 0.

Definition of $\varphi_{p(i)}$

Begin stage s

1. For each $y \in N$, let f_y denote the function defined as follows:

$$f_y(x) = \begin{cases} \varphi_{p(i)}(x) & x < x_s; \\ y & x_s \leq x \leq x_s + n; \\ 0 & \text{otherwise.} \end{cases}$$

2. Dovetail steps 3 and 4 until, if ever, step 3 succeeds. If and when step 3 succeeds, go to step 5.
3. Search for a y and m, with $m > x_s$, such that $\mathbf{M}_i(f_y[m]) \neq \mathbf{M}_i(f_y[x_s])$.
4. Let $x = x_s + n + 1$.

```
        repeat
            $\varphi_{p(i)}(x) = 0$; $x = x + 1$.
        forever
    5.  If and when 3 above succeeds, let $y, n$ be as found in step 3. Let $\varphi_{p(i)}(x) = y$ for
          $x_s \leq x \leq x_s + n$. For $x$ such that $x_s + n < x < m$ and $\varphi_{p(i)}(x)$ has not been defined
          till now, let $\varphi_{p(i)}(x) = 0$.
    6.  Go to stage $s + 1$.
    End stage $s$
End of definition of $\varphi_{p(i)}$
```

We now have to prove that card$(\{x \mid \varphi_{p(i)}(x)\uparrow\}) \leq n+1$ and $(\exists f)[\varphi_{p(i)} \subseteq f \wedge f \notin \mathbf{Ex}^n(\mathbf{M}_i)]$. For this we consider the following cases:

Case 1: Infinitely many stages are executed.

In this case clearly $\varphi_{p(i)}$ is total and $\mathbf{M}_i(\varphi_{p(i)})\uparrow$.

Case 2: Some stage s starts but never finishes.

In this case, for each $y \in N$, let f_y be as defined in step 1 in Stage s. Clearly $\{x \mid \varphi_{p(i)}(x)\uparrow\} = \{x_s, \ldots, x_s + n\}$, and, for all y, $\varphi_{p(i)} \subseteq f_y$. Also for all y, $\mathbf{M}_i(f_y) = \mathbf{M}_i(\varphi_{p(i)}[x_s])$. However, $\mathbf{M}_i(\varphi_{p(i)}[x_s])$ can compute an n variant of at most $(n+1)$ f_y's. Thus there exists an $f_y \supseteq \varphi_{p(i)}$ such that $\mathbf{M}_i$ does not $\mathbf{Ex}^n$-identify f_y.

From the above two cases it follows that $\mathcal{C} \notin \mathbf{Ex}^n$. ▌

Similarly it can be shown that

Theorem 10 $(\exists \mathcal{C})(\forall \mathbf{M})(\exists \mathbf{M}')[[\mathcal{C} \notin \bigcup_i \mathbf{Ex}^i] \wedge [\mathcal{C} \subseteq \mathbf{Ex}^*(\mathbf{M}')] \wedge [(\forall n)[(\mathbf{Ex}^n(\mathbf{M}) - \mathcal{C}) \subseteq \mathbf{Ex}^n(\mathbf{M}')]]]$.

Corollary 11 $(\forall \mathcal{C} \in \bigcup_i \mathbf{Ex}^i)(\exists \mathcal{C}' \supseteq \mathcal{C})[\mathcal{C}' \in \mathbf{Ex}^* - \bigcup_i \mathbf{Ex}^i]$.

The advice suggested by Corollary 11 is to consider allowing an unbounded, finite number of errors in final programs. A worked out proof of Theorem 10 would provide one way to algorithmically augment any learning machine to take advantage of this advice. We should caution the reader that not every learning application would be able to tolerate an *unbounded*, finite number of errors.

Theorem 12 below can be proved using a combination of the idea used to prove Theorem 13 below and the proof in Case and Smith (1983) to show that $\mathbf{Bc} - \mathbf{Ex}^* \neq \emptyset$.

Theorem 12 $(\forall \mathcal{C} \in \mathbf{Ex}^*)(\exists \mathcal{C}' \supseteq \mathcal{C})[\mathcal{C}' \in \mathbf{Bc} - \mathbf{Ex}^*]$.

The advice suggested by Theorem 12 is to not necessarily require a final program. Furthermore, this is more powerful advice than that given after Corollary 11 above. A worked out proof of

Theorem 12 would provide one way to algorithmically augment any learning machine to take advantage of this advice. We should caution the reader that in some cases the increased learning power *must* come from the improved machine outputting infinitely many different programs (Case and Smith (1983)), and, hence, in such cases, the *size* of these programs (Blum (1967b)) will grow without bound. There are two answers to this. One is that the human race may go extinct before the program sizes get too large to handle, so it might not matter. The other is that, while requiring the number of different programs output to be *finite* does *not* increase learning power (Barzdin and Podnieks (1973); Case and Smith (1983)), it *does* increase learning power if one also bounds a suitably sensitive measure of the computational complexity of the learning machine itself (Case, Jain and Sharma (1991)).

We now turn to the **Bc** hierarchy. Corollary 14 solves an open problem in Chen (1981).

Theorem 13 $(\forall n)(\exists \mathcal{C})(\forall \mathbf{M})(\exists \mathbf{M}')[[\mathcal{C} \not\in \mathbf{Bc}^n] \wedge [\mathcal{C} \subseteq \mathbf{Bc}^{n+1}(\mathbf{M}')] \wedge [(\mathbf{Bc}^n(\mathbf{M}) - \mathcal{C}) \subseteq \mathbf{Bc}^n(\mathbf{M}')]]$.

Corollary 14 $(\forall n)(\forall \mathcal{C} \in \mathbf{Bc}^n)(\exists \mathcal{C}' \supseteq \mathcal{C})[\mathcal{C}' \in \mathbf{Bc}^{n+1} - \mathbf{Bc}^n]$.

The advice suggested by Corollary 14 is similar to that suggested by Corollary 9: consider allowing even one (more) error in programs conjectured. The proof of Theorem 13 provides one way to algorithmically augment any learning machine to take advantage of this advice.

PROOF OF THEOREM 13. Fix n. Let $\mathbf{M}_0, \mathbf{M}_1, \ldots$ denote a recursive sequence of all inductive inference machines. We will construct a recursive sequence of pairwise distinct programs $p(0), p(1), \ldots$ such that, for each i, the following three conditions are satisfied.

(A) $(\exists j)[\varphi_{p(\langle i,j\rangle)}$ is total $]$.

(B) $(\forall j, k \mid j \leq k)[\text{domain}(\varphi_{p(\langle i,k\rangle)}) = \emptyset \vee \text{domain}(\varphi_{p(\langle i,j\rangle)}) \subseteq \text{domain}(\varphi_{p(\langle i,k\rangle)})]$.

(C) $\{f \in \mathcal{R} \mid f(0) = i \wedge (\forall j)[\text{card}(\{x \mid \varphi_{p(\langle i,j\rangle)}(x)\downarrow \neq f(x)\}) \leq n+1]\} \not\subseteq \mathbf{Bc}^n(\mathbf{M}_i)$.

Let $\mathcal{C}_i = \{f \in \mathcal{R} \mid f(0) = i \wedge (\forall j)[\text{card}(\{x \mid \varphi_{p(\langle i,j\rangle)}(x)\downarrow \neq f(x)\}) \leq n+1]\}$.

Note that conditions (A) and (B) imply that, for each i,

$[(\overset{\infty}{\forall} k)[\text{domain}(\varphi_{p(\langle i,k\rangle)}) = \emptyset \vee \varphi_{p(\langle i,k\rangle)}$ is total $] \bigwedge$

$[\text{card}(\{k \mid \text{domain}(\varphi_{p(\langle i,k\rangle)}) \neq \emptyset\}) < \infty \Rightarrow \varphi_{p(\langle i,\max(\{k \mid \text{domain}(\varphi_{p(i,k)}) \neq \emptyset\})\rangle)}$ is total $]]$.

Thus, for all f,

$$f \in \mathcal{C}_i \Rightarrow (\overset{\infty}{\forall} m)[\varphi_{p(i,\max(\{k<m \mid \Phi_{p(\langle i,k\rangle)}(0)<m\}))} =^{n+1} f]. \quad (8)$$

We take $\mathcal{C} = \bigcup_{i \in N} \mathcal{C}_i$. We will construct p as claimed above later. Given $\mathbf{M}$, we first construct $\mathbf{M}'$ as claimed in the statement of the theorem. Define $\mathbf{M}'$ as follows. Let

$$Sat(m, f) = (\exists k < m)[\text{card}(\{x < m \mid \Phi_{p(\langle f(0),k\rangle)}(x) < m \wedge \varphi_{p(\langle f(0),k\rangle)}(x) \neq f(x)\}) > n + 1] \quad (9)$$

$\mathbf{M}'(f[0]) = 0$. For $m > 0$, if $\neg Sat(m, f)$, then $\mathbf{M}'(f[m]) = p(\langle f(0), r\rangle)$, where $r = \max(\{k < m \mid \Phi_{p(\langle i,k\rangle)}(0) < m\})$; $\mathbf{M}'(f[m]) = \mathbf{M}(f[m])$ otherwise.

Note that, if $f \in \mathcal{C}$, then by (8), for all but finitely many m, $\mathbf{M}'(f[m])$ is a program for an $n+1$ variant of f. Thus $\mathcal{C} \subseteq \mathbf{Bc}^{n+1}(\mathbf{M}')$. If $f \notin \mathcal{C}$, then, for all but finitely many m, $Sat(m, f)$ is true, and thus, for all but finitely many m, $\mathbf{M}'(f[m]) = \mathbf{M}(f[m])$. It follows that $\mathbf{Bc}^n(\mathbf{M}) - \mathcal{C} \subseteq \mathbf{Bc}^n(\mathbf{M}')$.

We now construct p as claimed above. p can be constructed by easy modification of the construction used in the proof in Case and Smith (1983) of the fact that $\mathbf{Bc}^{n+1} - \mathbf{Bc}^n \neq \emptyset$. For completeness we give the details of the construction. Our construction of p, in addition to satisfying conditions (A) and (C) above also satisfies a strengthened version of (B).

(B$'$) $(\forall k)[\varphi_{p(\langle i,k\rangle)}$ is total $] \bigvee$

$(\exists k)[\varphi_{p(\langle i,k\rangle)}$ is total $\wedge (\forall k' < k)[\text{domain}(\varphi_{p(\langle i,k'\rangle)}) = \text{domain}(\varphi_{p(\langle i,0\rangle)})] \wedge (\forall k' > k)[\text{domain}(\varphi_{p(\langle i,k'\rangle)}) = \emptyset]]$.

We now proceed to construct p, as claimed above. By the operator recursion theorem (Case (1974)) there exists a recursive, one-to-one, p, such that, for each i, the (partial) function $\varphi_{p(\langle i,\cdot\rangle)}$ may be described as follows.

Let $\varphi_{p(\langle i,0\rangle)}(0) = i$. Let x_s denote the least x such that $\varphi_{p(\langle i,0\rangle)}(x)$ has not been defined before stage s. Go to stage 1.

```
Begin stage $s$
1.  For each $x < x_s$, let $\varphi_{p(\langle i,s\rangle)}(x) = \varphi_{p(\langle i,0\rangle)}(x)$.
2.  Dovetail steps 3 and 4 until, if ever, step 3 succeeds. If and when step 3 succeeds, complete the
        then *current* iteration of the repeat loop in step 4, and then, go to step 5.
3.  Let $g$ be the zero extension of $\varphi_{p(\langle i,0\rangle)}[x_s]$. Search for $m > x_s$, and for a $y > m$ such that, for
        all $r \leq n$, $\varphi_{\mathbf{M}_i(g[m])}(y + r)\downarrow = 0$.
4.  Let $x = x_s$.
    **repeat**
        $\varphi_{p(\langle i,s\rangle)}(x) = 0$.
        $x = x + 1$.
    **forever**
5.  If and when step 3 succeeds, let $y$ be as found in step 3.
    5.1   Let $\varphi_{p(\langle i,0\rangle)}(y + r) = 1$, for $r \leq n$.
```

5.2 Let $m' = \max(y+n, x)$.

5.3 For each $z \leq m'$ such that $\varphi_{p(\langle i,0\rangle)}(z)$ has not been defined till now, let $\varphi_{p(\langle i,0\rangle)}(z) = 0$.

5.4 For each $z \leq m'$ such that $\varphi_{p(\langle i,s\rangle)}(z)$ has not been defined till now, let $\varphi_{p(\langle i,s\rangle)}(z) = 0$.

5.5 For $z > m'$, make $\varphi_{p(\langle i,s\rangle)}(z) = \varphi_{p(\langle i,0\rangle)}(z)$ whenever $\varphi_{p(\langle i,0\rangle)}(z)$ gets defined (i.e. $\varphi_{p(\langle i,s\rangle)}$ "follows" $\varphi_{p(\langle i,0\rangle)}$ from now on).

(Note that, because of step 5, $\varphi_{p(\langle i,s\rangle)} =^{n+1} \varphi_{p(\langle i,0\rangle)}$ and domain$(\varphi_{p(\langle i,0\rangle)})$ = domain$(\varphi_{p(\langle i,s\rangle)})$.)

6. Go to stage $s+1$.

End stage s

Case 1: Infinitely many stages are executed.

In this case, for all s, $\varphi_{p(\langle i,s\rangle)}$ is total. Also, for all s, $\varphi_{p(\langle i,s\rangle)} =^{n+1} \varphi_{p(\langle i,0\rangle)}$ (see comment at the end of step 5). Thus $\varphi_{p(\langle i,0\rangle)} \in \mathcal{C}_i$. Also $\varphi_{p(\langle i,0\rangle)} \notin \mathbf{Bc}^n(\mathbf{M}_i)$, since because of the success, in each stage s, of step 3 and the diagonalization in step 5.1, for infinitely many m, $\varphi_{\mathbf{M}_i(\varphi_{p(\langle i,0\rangle)}[m])} \neq^n \varphi_{p(\langle i,0\rangle)}$. Thus conditions (A)–(C) as stated above are satisfied.

Case 2: Some stage s starts but never finishes.

Clearly, $\varphi_{p(\langle i,s\rangle)}$ is total. Properties (A) and (B) are immediate from the construction (see comment at the end of step 5). Clearly, $\varphi_{p(\langle i,s\rangle)} \in \mathcal{C}_i$. (C) follows from the fact that, for all but finitely many m, for infinitely many x, $\varphi_{\mathbf{M}_i(\varphi_{p(\langle i,s\rangle)}[m])}(x) \neq 0$ (otherwise step 3 would succeed). ∎

It is useful at this juncture to consider the particular ways presented in the proofs so far to algorithmically improve learning. It would be easy to argue that *these* ways to achieve improvement provide mathematical existence proofs but do not necessarily provide improvement on a class of functions that is likely to come up in a natural learning scenario. The proof of Theorem 13 involves a complex self (and other) reference argument. With minor changes we could recast the proof of Theorem 8 as a self-reference argument based on some form of the Kleene recursion theorem (Rogers (1967), Page 214). As will be seen, the proof of Theorem 17 below involves a self-reference argument based on the Kleene recursion theorem. In each case the existence of the means for improvement depends on a self-reference argument (or something close enough for our purposes). It is plausibly argued in Case (1988) that self-referential examples witnessing existence are harbingers of natural examples witnessing same. Hence, there is some reason to expect that a search to find natural ways to improve learning along the lines suggested in the present paper would be fruitful.

From Theorem 3, $\mathcal{R} \in \mathbf{Bc}^*$; hence, we have

Proposition 15 $(\forall \mathcal{C} \in \bigcup_i \mathbf{Bc}^i)(\exists \mathcal{C}' \supseteq \mathcal{C})[\mathcal{C}' \in \mathbf{Bc}^* - \bigcup_i \mathbf{Bc}^i]$.

4 Some More Results on Strong Separation

Here is a case where strong separation does not hold.

Theorem 16 $\neg[(\forall \mathcal{C} \in \mathbf{Pex})(\exists \mathcal{C}' \supseteq \mathcal{C})[\mathcal{C}' \in \mathbf{Tex} - \mathbf{Pex}]]$.

PROOF. Let $\mathcal{C} = \mathbf{ZERO}^* \in \mathbf{Pex}$. Any machine which **Tex**-identifies $\mathbf{ZERO}^*$ is a Popperian machine. ∎

Theorem 17 $(\exists \mathcal{C})(\forall \mathbf{M})(\exists \mathbf{M}')[[\mathcal{C} \notin \mathbf{Tex}] \wedge [\mathcal{C} \cup \mathbf{Tex}(\mathbf{M}) \subseteq \mathbf{Ex}(\mathbf{M}')]]$.

The advice suggested by the two corollaries just below to Theorem 17 is to not necessarily require that all the conjectures along the way to a final program compute total functions. The proof of Theorem 17 provides one way to algorithmically augment any learning machine to take advantage of this advice.

Corollary 18 $(\forall \mathcal{C} \in \mathbf{Tex})(\exists \mathcal{C}' \supseteq \mathcal{C})[\mathcal{C}' \in \mathbf{Ex} - \mathbf{Tex}]$.

Corollary 19 $(\forall \mathcal{C} \in \mathbf{Pex})(\exists \mathcal{C}' \supseteq \mathcal{C})[\mathcal{C}' \in \mathbf{Ex} - \mathbf{Pex}]$.

PROOF OF THEOREM 17. Given f, let $n_f = 1 + \min(\{x \mid \varphi_{f(0)}(x) \neq f(x) \vee \Phi_{f(0)}(x) \geq f(x+1)\})$ if $\{x \mid \varphi_{f(0)}(x) \neq f(x) \vee \Phi_{f(0)}(x) \geq f(x+1)\} \neq \emptyset$; n_f is undefined otherwise. Consider the following class of functions:

$\mathcal{C} = \{f \mid [\varphi_{f(0)} = f \wedge (\forall x)[\Phi_{f(0)}(x) < f(x+1)]] \vee [(\exists x)[\Phi_{f(0)}(x) \geq f(x+1) \vee \varphi_{f(0)}(x) \neq f(x)] \wedge [(\forall x \geq n_f)[f(x) = 0]]]\}$.

We first show that $\mathcal{C} \notin \mathbf{Tex}$. Suppose by way of contradiction that $\mathbf{M}$ **Tex**-identifies $\mathcal{C}$. Then by the Kleene recursion theorem there exists an e such that the following holds. Let x_s denote the least x such that $\varphi_e(x)$ is not defined before stage s. Let $\varphi_e(0) = e$. Go to stage 0.

```
Begin stage $s$
    If and when $\varphi_{\mathbf{M}(\varphi_e[x_s])}(x_s)\downarrow$, let $\varphi_e(x_s) = \varphi_{\mathbf{M}(\varphi_e[x_s])}(x_s) + \Phi_e(x_s - 1) + 1$ and go to stage $s+1$.
End stage $s$
```

Now consider the following cases.

Case 1: All stages halt.

In this case let $f = \varphi_e \in \mathcal{C}$. Now $\mathbf{M}(f)\uparrow$ or for all but finitely many x, $\varphi_{\mathbf{M}(f)}(x) \neq f(x)$ (by construction).

Case 2: Some stage s starts but never finishes.

Then let f be the zero extension of $\varphi_e[x_s]$. Clearly, $f \in \mathcal{C}$. But $\varphi_{\mathbf{M}(f[x_s])}$ is not total (otherwise stage s will halt!).

From the above cases we have that $\mathcal{C} \notin \mathbf{Tex}$. Now suppose $\mathbf{M}$ is given, we give $\mathbf{M}'$ satisfying the statement of the theorem. Let zeroext be such that, for all f, n: $\varphi_{\text{zeroext}(f[n])} =$ the zero extension of $f[n]$.

$$\mathbf{M}'(f[n]) = \begin{cases} f(0) & \text{if } (\forall x < n-1)[\Phi_{f(0)}(x) < f(x+1) \wedge \varphi_{f(0)}(x) = f(x)]; \\ \text{zeroext}(\varphi_e[n_f]) & \text{if } (\exists x < n-1)[\Phi_{f(0)}(x) \geq f(x+1) \vee \varphi_{f(0)}(x) \neq f(x)] \wedge \\ & \quad (\forall x \mid n_f \leq x < n)[f(x) = 0]; \\ \mathbf{M}(f[n]) & \text{otherwise.} \end{cases}$$

It is easy to see that $\mathbf{M}'$, $\mathbf{Ex}$-identifies $\mathcal{C} \cup \mathbf{Tex}(\mathbf{M})$. ▮

5 Acknowledgements

This work was supported by NSF grants MCS 8010728 and CCR 8713846. We would like to thank the anonymous referees for several helpful comments.

References

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

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

Barzdin, J. M. and Podnieks, K. (1973). The theory of inductive inference. In *Mathematical Foundations of Computer Science*.

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.

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

Case, J. (1986). Learning machines. In W. Demopoulos and A. Marras (Eds.), *Language Learning and Concept Acquisition*. Ablex Publishing Company.

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.

Case, J., Jain, S., and Sharma, A. (1991). Complexity issues for vacillatory function identification. In *Proceedings, Foundations of Software Technology and Theoretical Computer Science, Eleventh Conference, New Delhi, India. Lecture Notes in Computer Science 560*, (pp. 121–140). Springer-Verlag.

Case, J. and Ngo Manguelle, S. (1979). Refinements of inductive inference by Popperian machines. Technical Report 152, SUNY/Buffalo.

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, SUNY at Buffalo.

Feldman, J. (1972). Some decidability results on grammatical inference and complexity. *Information and Control*, *20*, 244–262.

Fulk, M. (1985). *A Study of Inductive Inference machines.* PhD thesis, SUNY at Buffalo.

Gold, E. M. (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.

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.

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.

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.

Popper, K. (1968). *The Logic of Scientific Discovery* (Second ed.). Harper Torch Books, New York.

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.

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.

Wiehagen, R. (1978). *Zur Therie der Algrithmischen Erkennung.* PhD thesis, Humboldt-Universitat, Berlin.