

# Machine Induction Without Revolutionary Changes in Hypothesis Size

John Case

Department of Computer and Information Sciences  
University of Delaware  
Newark, DE 19716, USA  
Email: case@cis.udel.edu

Sanjay Jain

Department of Information Systems and Computer Science  
National University of Singapore  
Singapore 119260, Republic of Singapore  
Email: sanjay@iscs.nus.sg

Arun Sharma

School of Computer Science and Engineering  
The University of New South Wales  
Sydney, NSW 2052, Australia  
Email: arun@cse.unsw.edu.au

## Abstract

This paper provides a beginning study of the effects on inductive inference of paradigm shifts whose absence is approximately modeled by various formal approaches to forbidding large changes in the size of programs conjectured.

One approach, called *severely parsimonious*, requires all the programs conjectured on the way to success to be nearly (i.e., within a recursive function of) minimal size. It is shown that this very conservative constraint allows learning infinite classes of functions, but *not* infinite *r.e.* classes of functions.

Another approach, called *non-revolutionary*, requires all conjectures to be nearly the same size as one another. This quite conservative constraint is, nonetheless, shown to permit learning some infinite *r.e.* classes of functions.

Allowing up to one extra *bounded size* mind change towards a final program learned certainly doesn't appear revolutionary. However, somewhat surprisingly for scientific (inductive) inference, it is shown that there are classes learnable *with* the non-revolutionary constraint (respectively, with severe parsimony), up to  $(i + 1)$  mind changes, and no anomalies, which classes *cannot* be learned with no size constraint, an unbounded, finite number of anomalies in the final program, but with no more than  $i$  mind changes. Hence, in some cases, the possibility of one extra mind change is considerably more liberating than removal of very conservative size shift constraints. The proofs of these results are also combinatorially interesting.

# 1 Introduction

The present paper is a beginning study of the phenomenon of paradigm shift (see Kuhn [26]) in the context of machine inductive learning/inference. The issues are difficult to formalize directly mathematically, so instead of tackling them directly, we take an indirect approach and investigate the effects on induction of disallowing *certain formalizable kinds of* paradigm shift. A number of mathematical formulations that capture some interesting notions of paradigm shift are proposed. The induction task we chose for this investigation is the identification of computer programs for computable functions from their graphs.<sup>1</sup>

A paradigm shift is usually associated with a significant *conceptual* change in the hypothesis conjectured by the “learner.” Conceptual change (significant or otherwise) is difficult to formalize rigorously. A possible, though overly simple expectation is that a significant conceptual change in hypotheses is also accompanied by a significant change in the *size* of the hypotheses. Thus, keeping learners from conjecturing hypotheses of extreme variation in size may be seen as disallowing a kind of paradigm shift. This forms the basis of our beginning attempt herein to model induction without paradigm shifts.

*Learning machines* may be thought of as Turing machines computing a mapping from “finite sequences of data” into computer programs. A typical variable for learning machines is  $\mathbf{M}$ . A *function learning machine* may be thought of as a learning machine that at any given time, takes an initial segment of the graph of a function as input<sup>2</sup> and outputs (the index of) a computer program in some fixed acceptable programming system [36, 28, 38]. We now describe what it means for such a machine to learn a function.

Let  $N$  denote the set of natural numbers. Let  $\mathcal{R}$  denote the set of all computable functions (i.e., the set of partial computable functions that are total) with arguments and values from  $N$ . The following definition is essentially Gold’s [22] criterion for successful identification of functions by learning machines.

**Definition 1** [22]

- (a)  $\mathbf{M}$  **Ex**-identifies  $f \in \mathcal{R}$  just in case  $\mathbf{M}$ , successively fed initial segments of  $f$  of increasing length, converges to an index of a program for  $f$ . In this case we say that  $f \in \mathbf{Ex}(\mathbf{M})$ .
- (b) **Ex** denotes the collection of all classes  $\mathcal{S}$  of computable functions such that some machine **Ex**-identifies each function in  $\mathcal{S}$ .

**Ex** is a set theoretic summary of the capability of single machines to **Ex**-identify classes of functions. In the sequel, we will refer to “an index of a program” by “a program.”

As noted, in the present study, we will approximately model disallowing paradigm shift in terms of restrictions regarding the size of a machine’s conjectures on the graph of a function. It is instructive, then, to review studies of size restriction on hypotheses in the inductive inference literature. These studies have been motivated by a desire to model *Occam’s Razor*<sup>3</sup>—a heuristic about the desirability of parsimony in explanations.

---

<sup>1</sup>Several papers in computational learning theory, which investigate identification of computer programs for computable functions, have already provided explicit insights into inductive inference in science (see, for example, [33, 6, 15, 12, 2, 11, 21, 27]).

<sup>2</sup>For each of the criteria of successful learning studied in this paper, it is easy to show that, *without loss of generality*, the graphs of the (total) function inputs can be received in standard order.

<sup>3</sup>*Entia non sunt multiplicanda, prater necessitatem*: attributed to the medieval philosopher William of Occam. However, W. M. Thorburn [41] raises some doubts as to whether William of Occam ever used the above expression, and raises the possibility that it may have originated in the work of Duns Scotus. According to Thorburn [42], the terminology, Occam’s Razor, seems to have first appeared in 1852 in the work of Sir William Hamilton. Moody [31] provides a study of the philosophy of William of Occam. ‘Occam’ is sometimes spelled ‘Ockham’.

The study of size restriction on hypotheses requires that we clarify the notion of size of hypotheses. Since hypotheses conjectured by learning machines are computer programs, any size measure that satisfies Blum’s [8] axioms for size measures suffices. It is easy to see that one of the simplest size measures satisfying Blum’s axioms is the index of the program. We present our results in the context of this simple size measure. We, however, note that all the results described in the present paper hold for any Blum size measure. The choice of index of program as the size measure is motivated by its simplicity and its use in earlier studies of program size restrictions in machine induction.

Freivalds [17] was the first to consider a criterion of success in which a learning machine was required to conjecture the minimal size program for the function being identified. Unfortunately, this criterion of success turned out to be mathematically problematical since the collections of functions that could be identified according to this criterion were dependent on the acceptable programming system used to interpret a learner’s conjectures. More precisely, he showed that there are acceptable programming systems in which only finite classes of computable functions can be minimally identified and other acceptable systems in which infinite classes of computable functions can be so identified. Freivalds [17] relaxed the stringent minimality requirement to introduce a criterion of success called *nearly minimal identification* according to which a learner need converge only to a program for the function whose size is within a computable “fudge factor” of the minimal size program for the function. This criterion, called herein **ME<sub>x</sub>** and which is acceptable programming system *independent*<sup>4</sup>, turned out to be a useful notion and was extended in Chen [16] to include the case of anomalies in the final hypothesis (see [24] for extension of Freivalds’ and Chen’s work to language identification). However, it is easy to see that nearly minimal identification does not provide a suitable model for inference without paradigm shift because, although there is a size restriction placed on the final hypothesis, a learner is free to conjecture hypotheses of arbitrary size before the onset of convergence.<sup>5</sup> Addressing this objection yields our first model of induction without paradigm shift. This formulation, referred to as *severely parsimonious identification*, requires that a learner not only conjecture a final program that is within a computable fudge factor of minimal size, but all its conjectures on the function being identified are required to be within that computable fudge factor of minimal size. We clarify this notion in the next definition.

- Definition 2** (a) **M** is *severely parsimonious* on a collection of functions  $\mathcal{S}$  just in case there exists a computable function  $h$  such that, for each  $f \in \mathcal{S}$ , **M**, on initial segments of  $f$ , outputs only programs that are of size no greater than  $h(\text{minimal size program for } f)$ .
- (b) **SpEx** denotes the set of collections  $\mathcal{S}$  of functions such that there exists a machine that is severely parsimonious on  $\mathcal{S}$  and that **Ex**-identifies  $\mathcal{S}$ .

A somewhat stronger requirement in which the learner behaves severely parsimoniously on every computable function is referred to as *globally severely parsimonious* identification. The class **gSpEx** is defined to be the collection of sets of functions,  $\mathcal{S}$ , such that there exists a machine that is severely parsimonious on every computable function and that **Ex**-identifies  $\mathcal{S}$ . This latter, global version of severe parsimony turns out to be too restrictive since we are able to show that only finite collections of functions can be identified by globally severely parsimonious learners. On the other hand there are infinite collections of functions that can be identified by

---

<sup>4</sup>Furthermore, the presence of the computable fudge factor in nearly minimal identification *and in our new criteria below* together with Blum’s recursive relatedness result for his program size measures [8] nicely wash out any possible dependence of these criteria on the choice of Blum program size measure.

<sup>5</sup>It does quite importantly model the possible goal in science of eventually finding *parsimonious* explanations.

severely parsimonious learners. However, no *infinite r.e.* class of (total) computable functions is identifiable by severely parsimonious machines! Surprisingly, then, standard classes easily and naturally identifiable without the constraint of severe parsimony (for example, by the enumeration technique [22, 6]) are no longer identifiable with this constraint.

The simplicity of severely parsimonious identification notwithstanding, it suffers from a crucial drawback as a model of induction without paradigm shift. The following argument illuminates this point.

There is nothing that prevents a learner from going from a small size conjecture to a large size conjecture: The restriction is due to the size of the programs conjectured in relation to the minimal size program for the function being learned, and not due to the sizes of the various conjectured programs. To see this point, suppose on successive initial segments a learner outputs a very small size program and a very large size program. It is not clear if a paradigm shift (of the kinds we are considering) has taken place. One cannot be sure that such a paradigm shift has taken place since later parts of the function may be complex enough so that the size of minimal program for the function is huge, hiding all the earlier differences in the conjectures. Although the global version of severe parsimony avoids this problem, as noted above, it turns out to be too restrictive.

To address these concerns, we introduce another approach to modeling induction without paradigm shifts. This new approach considers learners that, on any computable function, conjecture programs which differ only “slightly” in size from each other. We make this idea precise with the help of some technical machinery. Suppose  $\mathbf{M}$  is a learning machine and  $f$  is a computable function. Then,  $\text{ProgSet}(\mathbf{M}, f)$  is defined to be the collection of programs that are output by  $\mathbf{M}$  on any initial segment of  $f$ .

**Definition 3** A learning machine  $\mathbf{M}$  is said to be *non-revolutionary* just in case there exists a computable function  $h$  such that for *each* computable  $f$ ,

$$\max(\text{ProgSet}(\mathbf{M}, f)) \leq h(\min(\text{ProgSet}(\mathbf{M}, f))).$$

In other words, the range of a non-revolutionary machine’s conjectures on any computable function is limited in terms of size, in fact there is a bound on how large the largest size conjecture can be in relation to the smallest size conjecture and this bound is uniform across every computable function.

**Definition 4**  $\mathbf{NrEx}$  denotes the set of collections of functions,  $\mathcal{S}$ , such that some non-revolutionary machine  $\mathbf{Ex}$ -identifies each function in  $\mathcal{S}$ .<sup>6</sup>

Clearly, non-revolutionary learners capture an interesting notion of induction without paradigm shifts.

In the present paper we completely compare the power of all criteria considered (with and without anomalies allowed and/or mind change bounds [3, 15]), and below are some interesting highlights.

Of course we would (correctly) expect the non-revolutionary restriction to limit learning power, i.e., we expect and have (from Theorem 32) that  $(\mathbf{Ex} - \mathbf{NrEx}) \neq \emptyset$ . Surprisingly, though, this theorem says, moreover, that *some* classes can be learned with severe parsimony, no

---

<sup>6</sup>It is easily seen that, were we to require instead in the definition of  $\mathbf{NrEx}$  that some machine  $\mathbf{M}$   $\mathbf{Ex}$ -identifies each function in  $\mathcal{S}$ , where  $\mathbf{M}$ ’s non-revolutionary behavior is exhibited on all  $f \in \mathcal{S}$  (and not necessarily on all computable  $f$ ), the class  $\mathbf{NrEx}$  would be the same. Likewise, it would be the same were we to require the non-revolutionary behavior on all  $f$ , computable or otherwise.

anomalies and at most one mind change, which classes *cannot* be learned by a non-revolutionary machine even with no restrictions on mind changes and allowing an unbounded, finite number of anomalies in the final programs learned! *By contrast*, however, we see below from the proof of Corollary 17(a) and from Proposition 31 that **NrEx**, unlike **SpEx**, *does* contain some infinite r.e. classes of (total) computable functions; hence, the quite conservative non-revolutionary restriction is, in *some other and important cases*<sup>7</sup>, not as strongly deleterious to learning power as severe parsimony.

Of course the non-revolutionary constraint is quite conservative (as is severe parsimony). Allowing, say, up to one extra mind change toward a final program learned under the non-revolutionary constraint is seemingly trivially liberal—one little mind change *of bounded size* doesn't seem to make a revolution. *However*, by Theorem 35, somewhat surprisingly, there are classes learnable *with* the non-revolutionary constraint, up to  $(i + 1)$  mind changes, and no anomalies, which classes *cannot* be learned with no size constraint, an unbounded, finite number of anomalies in the final program, but with no more than  $i$  mind changes. Hence, in some cases, the possibility of one extra mind change is considerably more liberating than removal of the quite conservative non-revolutionary constraint. The proof of this result is combinatorially interesting.

Also, we have, by Corollary 34(a), that possible anomalies, like possible mind changes, in some cases, liberate more learning power than permitting revolutionary shifts in program size. This result, however, follows easily from prior results in the literature.

Theorems 26 and 24 provide results similar to those of the just above two paragraphs, but for the very conservative constraint of severe parsimony.

Though this paper is mostly concerned with function identification, the above notions have counterparts in language identification. We would like to note that most results carry over to the language learning context; however, the picture, for language learning without paradigm shift, is more complicated for vacillatory identification [10, 13, 25]. Future work will also consider slightly less conservative criteria in which the fudge factor functions  $h$  are allowed to be limiting-recursive [40, 14].

We now proceed formally. Section 2 introduces the notation and preliminary notions from inductive inference literature. Severely parsimonious identification is considered in Section 3. Non-revolutionary identification is studied in Section 4. Section 5 deals with issues arising out of behaviorally correct and vacillatory function learning, each without paradigm shift. Our proofs of many of the theorems in this paper involve complicated combinatorial arguments.

## 2 Notation and Preliminaries

Recursion-theoretic concepts not explained below are treated in [37].  $N$  denotes the set of natural numbers.  $*$  denotes a non-member of  $N$  and is assumed to satisfy  $(\forall n)[n < * < \infty]$ . We let  $e, i, j, k, l, m, n, p, r, s, t, x, y,$  and  $z$ , with or without decorations, range over  $N$ . We let  $a, b, c,$  and  $d$ , with or without decorations, range over  $N \cup \{*\}$ . We let  $P, S, X,$  with or without decorations, range over subsets of  $N$  and we let  $D$  range over finite subsets of  $N$ .  $\in, \subseteq, \subset, \supseteq, \supset,$  respectively denote membership, subset, proper subset, superset and proper superset relations for sets.  $\emptyset$  denotes emptyset.  $\text{card}(P)$  denotes the cardinality of  $P$ . So then, ' $\text{card}(P) \leq *$ ' means that  $\text{card}(P)$  is finite.  $\min(P)$  and  $\max(P)$  respectively denote the

---

<sup>7</sup>Infinite r.e. classes of total computable functions are archetypal examples of **Ex**-learnable classes [22, 6] and the object of further important study (for example, [4] regarding the complexity of their inference and [20] regarding their alleged ubiquity in learning).

minimum and maximum element in  $P$ . We take  $\min(\emptyset)$  to be  $\infty$  and  $\max(\emptyset)$  to be 0.

$\langle \cdot, \cdot \rangle$  denotes a 1-1 computable mapping from pairs of natural numbers onto natural numbers.  $\pi_1, \pi_2$  are the corresponding projection functions.  $\langle \cdot, \cdot \rangle$  is extended to  $n$ -tuples in a natural way.

$\eta$ , with or without decorations, ranges over partial functions. For  $a \in N \cup \{*\}$ ,  $\eta_1 =^a \eta_2$  means that  $\text{card}(\{x \mid \eta_1(x) \neq \eta_2(x)\}) \leq a$ . (If  $\eta_1$  and  $\eta_2$  are both undefined on input  $x$ , then, as is standard, we take  $\eta_1(x) = \eta_2(x)$ .)  $\text{domain}(\eta)$  and  $\text{range}(\eta)$  respectively denote the domain and range of the partial function  $\eta$ .

$f, g, h$ , with or without decorations, range over  $\mathcal{R}$ .  $\mathcal{C}$  and  $\mathcal{S}$ , with or without decorations, range over subsets of  $\mathcal{R}$ .  $\psi$  ranges over *acceptable* programming systems for the partial computable functions:  $N \rightarrow N$ .  $\psi_i$  denotes the partial function computed by the  $i$ -th program in the  $\psi$  programming system. For the sake of brevity, we refer to the acceptable programming system  $\psi$  as simply the  $\psi$ -system.  $\varphi$  denotes a *fixed* acceptable programming system.  $\varphi_i$  denotes the partial computable function computed by program  $i$  in the  $\varphi$ -system. We let  $\Phi$  be an arbitrary Blum complexity measure [7] associated with the acceptable programming system  $\varphi$ ; such measures exist for any acceptable programming system [7]. For a given partial computable function  $\eta$ , we define  $\text{MinProg}(\eta)$  to denote  $\min(\{i \mid \varphi_i = \eta\})$ .

## 2.1 Function Identification in the Limit

We first describe function learning machines. We assume, without loss of generality, that the graph of a function is fed to a machine in canonical order. For  $f \in \mathcal{R}$  and  $n \in N$ , we let  $f[n]$  denote the finite initial segment  $\{(x, f(x)) \mid x < n\}$ . Clearly,  $f[0]$  denotes the empty segment.  $\text{SEG}$  denotes the set of all finite initial segments,  $\{f[n] \mid f \in \mathcal{R} \wedge n \in N\}$ . We let  $\sigma$ , with or without decorations, range over  $\text{SEG}$ .

**Definition 5** [22] A *function learning machine* is an algorithmic device that computes a mapping from  $\text{SEG}$  into  $N \cup \{?\}$ .

Intuitively, “?” above denotes the case when the machine may not wish to make a conjecture. Although it is not necessary to consider learners that issue “?” for identification in the limit, it becomes useful when the number of mind changes a learner can make is bounded. In this paper, we assume, without loss of generality, that once a function learning machine has issued a conjecture on some initial segment of a function, it outputs a conjecture on all extensions of that initial segment. This is without loss of generality, because a machine wishing to emit “?” after making a conjecture can instead be thought of as repeating its old conjecture. We let  $\mathbf{M}$ , with or without decorations, range over learning machines.

Since the set of all finite initial segments,  $\text{SEG}$ , can be coded onto  $N$ , we can view these machines as taking natural numbers as input and emitting natural numbers or ?’s as output. The next definition describes function identification in the limit. We also consider the case in which the final program is allowed to have anomalies. Some notation about anomalous programs is in order. Recall that for  $a \in N \cup \{*\}$ , a partial recursive function  $\eta$ , and a recursive function  $f$ , we say that  $\eta =^a f$  (read:  $\eta$  is an  $a$ -variant of  $f$ ) just in case  $\text{card}(\{n \mid \eta(n) \neq f(n)\}) \leq a$ . If  $\eta(x)$  is defined and  $\eta(x) \neq f(x)$ , then  $\eta$  is said to be *convergently different from  $f$  at  $x$* . It is helpful to think of a program  $i$  for  $\eta$  as an “anomalous explanation” for  $f$ , that is, an explanation with a finite number of anomalies (in fact,  $\leq a$  anomalies) in its predictions of values of  $f$ . In this case  $i$  is referred to as an  $a$ -error program for  $f$ . Finally, we say that  $\mathbf{M}(f)$  converges to  $i$  (written:  $\mathbf{M}(f)\downarrow = i$ ) iff  $(\forall n) [\mathbf{M}(f[n]) = i]$ ;  $\mathbf{M}(f)$  is undefined if no such  $i$  exists.

**Definition 6** [22, 6, 15] Let  $a, b \in N \cup \{*\}$ . Let  $f \in \mathcal{R}$ .

- (a)  $\mathbf{M} \mathbf{Ex}_b^a$ -identifies  $f$  (written:  $f \in \mathbf{Ex}_b^a(\mathbf{M})$ ) just in case there exists an  $a$ -error program  $i$  for  $f$  such that  $\mathbf{M}(f) \downarrow = i$  and  $\text{card}(\{n \mid ? \neq \mathbf{M}(f[n]) \neq \mathbf{M}(f[n+1])\}) \leq b$  (i.e.,  $\mathbf{M}$  makes no more than  $b$  mind changes on  $f$ ).
- (b)  $\mathbf{M} \mathbf{Ex}_b^a$ -identifies  $\mathcal{S}$  iff  $\mathbf{M} \mathbf{Ex}_b^a$ -identifies each  $f \in \mathcal{S}$ .
- (c)  $\mathbf{Ex}_b^a = \{\mathcal{S} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{S} \subseteq \mathbf{Ex}_b^a(\mathbf{M})]\}$ .

The relationship between the above criteria is summarized in the following theorem.

**Theorem 7** [15, 6]

- (a) Let  $b \in N \cup \{*\}$ . Then,  $\mathbf{Ex}_b = \mathbf{Ex}_b^0 \subset \mathbf{Ex}_b^1 \subset \mathbf{Ex}_b^2 \subset \dots \subset \mathbf{Ex}_b^*$ .
- (b) Let  $a \in N \cup \{*\}$ . Then,  $\mathbf{Ex}_0^a \subset \mathbf{Ex}_1^a \subset \mathbf{Ex}_2^a \subset \dots \subset \mathbf{Ex}_*^a$ .
- (c)  $(\forall a, b, c, d \in N \cup \{*\})[\mathbf{Ex}_b^a \subseteq \mathbf{Ex}_d^c \iff (a \leq c) \wedge (b \leq d)]$ .

## 2.2 Nearly Minimal Identification

Since our study of induction without paradigm shift is intimately related to identification of succinct programs, we next define the notion nearly minimal identification considered by Freivalds [17] and extended by Chen [16].

**Definition 8** [17, 19, 16] Let  $a, b \in N \cup \{*\}$ .

- (a)  $\mathbf{M} \mathbf{MEx}_b^a$ -identifies  $f$ , with recursive fudge factor  $h$ , (written:  $f \in \mathbf{MEx}_b^a(\mathbf{M}, h)$ ) iff  $\mathbf{M} \mathbf{Ex}_b^a$ -identifies  $f$  and  $\mathbf{M}(f) \leq h(\text{MinProg}(f))$ .
- (b)  $\mathbf{M} \mathbf{MEx}_b^a$ -identifies  $\mathcal{S}$ , iff there exists a recursive fudge factor  $h$  such that,  $\mathcal{S} \subseteq \mathbf{MEx}_b^a(\mathbf{M}, h)$ .
- (c)  $\mathbf{MEx}_b^a = \{\mathcal{S} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathbf{M} \mathbf{MEx}_b^a\text{-identifies } \mathcal{S}]\}$ .

We write  $\mathbf{MEx}$  for  $\mathbf{MEx}_*^0$  and  $\mathbf{MEx}^a$  for  $\mathbf{MEx}_*^a$ .

The following is a theorem relating  $\mathbf{MEx}$ -identification and  $\mathbf{Ex}$ -identification.

**Theorem 9** [16, 23] Let  $a, b, c, d \in N \cup \{*\}$  and  $n \in N$ . Then

- (a)  $\mathbf{MEx}_b^a \subseteq \mathbf{Ex}_d^c \iff (a \leq c) \wedge (b \leq d)$ .
- (b)  $\mathbf{Ex}_n^a \subset \mathbf{MEx}_*^a$ .
- (c)  $\mathbf{Ex}_0^0 - \mathbf{MEx}_n^* \neq \emptyset$ .
- (d)  $\mathbf{Ex}^0 - \mathbf{MEx}^n \neq \emptyset$ .
- (e)  $\mathbf{Ex}^* = \mathbf{MEx}^*$ .

Before we begin our discussion of machine induction without revolutionary paradigm shifts, we present the following lemma which is useful in many of our diagonalization results.

**Lemma 10** [32] *There exists an r.e. sequence  $\mathbf{M}_0, \mathbf{M}_1, \dots$ , of learning machines such that for all criterion of identification,  $\mathbf{I}$ , discussed in this paper, for all  $\mathcal{S} \in \mathbf{I}$ , there exists an  $i$  such that  $\mathcal{S} \in \mathbf{I}(\mathbf{M}_i)$ .*



The reader should note that the above lemma holds not only for  $\mathbf{I} \in \{\mathbf{Ex}_b^a, \mathbf{MEx}_b^a\}$  but for all the other criteria defined in the sequel. The advantage of the above lemma is that in diagonalization arguments it suffices to show that none of the learning machines in the sequence  $\mathbf{M}_0, \mathbf{M}_1, \mathbf{M}_2, \dots$  is successful.

### 3 Severely Parsimonious Identification

As noted in the introductory section, the constraint of severe parsimony requires a learning machine to emit programs of size within a computable “fudge” factor of the minimal size program. We formalize this notion in the next definition. In doing so we also consider the following two modifications on the constraint of severe parsimony as described in the introduction.

- (a) Allowing the final programs to contain errors.
- (b) Introducing a bound on the number of mind changes before the onset of convergence.

**Definition 11** Let  $a, b \in N \cup \{*\}$ .

- (a)  $\mathbf{M}$  is *severely parsimonious* on  $\mathcal{S} \subseteq \mathcal{R}$  just in case there exists a recursive function  $h$  such that, for each  $f \in \mathcal{S}$ , for each  $n \in N$ ,  $\mathbf{M}(f[n]) \leq h(\text{MinProg}(f))$ .
- (b)  $\mathbf{M}$  is *globally severely parsimonious* just in case  $\mathbf{M}$  is severely parsimonious on  $\mathcal{R}$ .
- (c)  $\mathbf{SpEx}_b^a = \{\mathcal{S} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathbf{M} \text{ is severely parsimonious on } \mathcal{S} \text{ and } \mathbf{M} \text{ Ex}_b^a\text{-identifies } \mathcal{S}]\}$ .
- (d)  $\mathbf{gSpEx}_b^a = \{\mathcal{S} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathbf{M} \text{ is globally severely parsimonious and } \mathbf{M} \text{ Ex}_b^a\text{-identifies } \mathcal{S}]\}$ .

Intuitively,  $\mathbf{SpEx}_b^a$  denotes the class of sets of functions,  $\mathcal{S}$ , such that some machine that is severely parsimonious on  $\mathcal{S}$   $\text{Ex}_b^a$ -identifies  $\mathcal{S}$ . On the other hand  $\mathbf{gSpEx}_b^a$  denotes the class of sets of functions that can be identified by a globally severely parsimonious machine. It is easy to see that for all  $a, b \in N \cup \{*\}$ ,

$$\mathbf{gSpEx}_b^a \subseteq \mathbf{SpEx}_b^a.$$

Clearly,  $\mathbf{SpEx}$  is  $\mathbf{SpEx}_*^0$ . Also, by  $\mathbf{SpEx}^a$  we mean  $\mathbf{SpEx}_*^a$ . We first establish the following result which shows that, if a machine  $\mathbf{M}$  is severely parsimonious on an r.e. collection of functions  $\mathcal{S}$ , then  $\mathbf{M}$  outputs only finitely many programs on initial sequences drawn from functions in  $\mathcal{S}$ . This result is used to show that  $\mathbf{gSpEx}$  is a trivial class.

**Theorem 12** *Suppose  $\mathcal{S}$  is an r.e. class of recursive functions and  $\mathbf{M}$  is severely parsimonious on  $\mathcal{S}$ . Then  $\text{card}(\{\mathbf{M}(f[n]) \mid f \in \mathcal{S} \wedge n \in N\}) < \infty$ .*

PROOF. Let  $\mathcal{S}$  be an r.e. class of recursive functions. Suppose by way of contradiction that  $\mathbf{M}$  is severely parsimonious on  $\mathcal{S}$  and  $\text{card}(\{\mathbf{M}(f[n]) \mid f \in \mathcal{S} \wedge n \in N\}) = \infty$ . Then for all  $i$ , there exists an  $f \in \mathcal{S}$  and an  $n \in N$  such that  $\mathbf{M}(f[n]) > i$ .

Let  $h$  be such that, for all  $f \in \mathcal{S}$  and  $n \in N$ ,  $\mathbf{M}(f[n]) \leq h(\text{MinProg}(f))$ . Without loss of generality we can assume that  $h$  is an increasing function. Now, by the implicit use of Kleene’s recursion theorem [37], there exists an  $e$  such that  $\varphi_e$  may be defined as follows.

**begin** {Definition of  $\varphi_e$ }

Search for an  $f \in \mathcal{S}$  and  $n \in N$  such that  $\mathbf{M}(f[n]) > h(e)$ .

{This search is possible because  $\mathcal{S}$  is r.e. and because of the supposition.}

If and when such an  $f, n$  are found, let  $\varphi_e = f$ .

**end** {Definition of  $\varphi_e$ }

It is easy to see that  $\varphi_e \in \mathcal{S}$  and for some  $n$ ,  $\mathbf{M}(\varphi_e[n]) > h(e)$ . A contradiction. Thus, if  $\mathbf{M}$  is severely parsimonious on  $\mathcal{S}$ , then  $\text{card}(\{\mathbf{M}(f[n]) \mid f \in \mathcal{S} \wedge n \in N\}) < \infty$ .  $\blacksquare$

The construction in the just previous proof is quite like of Blum's [8] in his proof that programs in an acceptable system can be vastly more succinct than corresponding programs from a subrecursive system such as loop programs [30] for the primitive recursive functions (see also [29, 39]).

As a corollary to the above theorem we have.

**Corollary 13** *Suppose  $\mathcal{S}$  is an infinite r.e. class of recursive functions. Then,  $\mathcal{S} \notin \mathbf{SpEx}$ .*

As a corollary to proof of Theorem 12 we have that, if  $\mathbf{M}$  is severely parsimonious on  $\mathcal{R}$ , then range of  $\mathbf{M}$  consists of only finitely many conjectures. Thus,

**Corollary 14**  $(\forall \mathcal{S} \subseteq \mathcal{R})[\mathcal{S} \in \mathbf{gSpEx} \iff \text{card}(\mathcal{S}) < \infty]$ .

Thus, globally parsimonious machines can **Ex**-identify only finitely many functions.<sup>8</sup> The non-global version of severe parsimony, however, is not so restrictive since it can potentially identify classes that are not r.e. This is the subject of the next theorem.

**Theorem 15**  $(\exists \mathcal{S} \subseteq \mathcal{R})[\text{card}(\mathcal{S}) = \infty \wedge \mathcal{S} \in \mathbf{SpEx}_0^0]$ .

PROOF. We will construct an acceptable programming system  $\psi$  such that an infinite class of functions is  $\mathbf{SpEx}_0^0$  identifiable in the programming system  $\psi$ . Since the class  $\mathbf{SpEx}_b^a$  is acceptable programming system independent, we have our result. Such constructions will be used in other proofs in this paper. Freivalds was the first to use such constructions [17] (see also, [18]).

Define  $\psi$  as follows.

Let  $\psi_{3i} = \varphi_i$ . Note that this makes  $\psi$  acceptable. For  $j$  not divisible by 3, let  $\psi_j$  be defined as follows:  $\psi_j(x) = j$ , for all  $x$ . Let  $\mathcal{S} = \{\psi_j \mid j \text{ is not divisible by 3 and } \text{MinProg}(\psi_j) = j\}$ . It is easy to see that  $\mathcal{S}$  is an infinite class.

Define a machine  $\mathbf{M}$  as follows: for  $n > 0$ ,  $\mathbf{M}(f[n]) = f(0)$ . It is easy to see that  $\mathbf{M}$  witnesses  $\mathcal{S} \in \mathbf{SpEx}_0^0$ .  $\blacksquare$

An immediate corollary of the above two theorems follows:

**Corollary 16**  $\mathbf{gSpEx} \subset \mathbf{SpEx}$ .

Intuitively, the reason for the validity of the above corollary is that for global parsimony one requires a machine to be parsimonious on all initial segments; this restricts the machine to emit only finitely many distinct conjectures. However, for successful **SpEx**-identification a machine need not be parsimonious on all initial segments; it needs to be parsimonious only on initial segments of functions from the class being identified. This leaves open the possibility that a machine may parsimoniously identify a class of functions that contains no infinite r.e. class as a subset. This is what is exploited in the proof of Theorem 15 above.

---

<sup>8</sup>Even for identification with anomalies globally parsimonious machines cannot identify much since all the functions identified by the machines must be within the specified error bound of functions computed by finitely many programs.

### 3.1 Severe Parsimony and Nearly Minimal Identification

Because of the results in the previous section, we only investigate the non-global version of severe parsimony. Our first aim is to compare the effects of altering the parameters  $a$  and  $b$  in  $\mathbf{SpEx}_b^a$ . This is facilitated by looking at  $\mathbf{SpEx}_b^a$  in relation to nearly minimal identification, that is classes  $\mathbf{MEx}_b^a$ . It is easy to verify that for all  $a, b \in N \cup \{*\}$ ,

$$\mathbf{SpEx}_b^a \subseteq \mathbf{MEx}_b^a.$$

First, we would like to find out how does  $\mathbf{SpEx}^0$  compare with  $\mathbf{Ex}_0^0$ . To this end it is helpful to consider the class  $\mathcal{S}^0 = \{f \mid \varphi_{f(0)} = f\}$ . It can be shown that  $\mathcal{S}^0$  contains an infinite r.e. class as a subset. It is also easy to see that  $\mathcal{S}^0 \in \mathbf{Ex}_0^0$ . Thus, as an immediate consequence of Theorem 13 above we have Corollary 17(a) below which says that there are collections of functions that can be finitely identified, but which cannot be identified in the limit by severely parsimonious learners. Additionally, since Freivalds [17] (see also Chen [16]) showed that  $\mathcal{S}^0 \in \mathbf{MEx}^0$ , we also get Corollary 17(b).

#### Corollary 17

- (a)  $\mathbf{Ex}_0^0 - \mathbf{SpEx}^0 \neq \emptyset$ .
- (b)  $\mathbf{MEx}^0 - \mathbf{SpEx}^0 \neq \emptyset$ .

A natural question is, if the anomalous version of severe parsimony is considered, does Part (a) of the above corollary still hold (i.e., is  $\mathbf{Ex}_0^0 - \mathbf{SpEx}^* \neq \emptyset$ ?). To answer this question, we introduce the following technical definition.

**Definition 18** Let  $a \in N \cup \{*\}$ .

- (a) A finite set  $D$  *a-supports* a class of recursive functions  $\mathcal{S}$  just in case, for each  $f \in \mathcal{S}$ , there exists an  $i \in D$  such that  $\varphi_i =^a f$ .
- (b)  $\mathcal{S} \subseteq \mathcal{R}$  is *a-supportable* just in case there exists a finite set that *a-supports*  $\mathcal{S}$ .

Intuitively,  $D$  *a-supports*  $\mathcal{S}$  iff, for each  $f \in \mathcal{S}$ ,  $D$  contains a program for an  $a$ -variant of  $f$ . The next corollary is a counterpart of Corollary 13 for  $\mathbf{SpEx}^a$ . It follows immediately from Theorem 12.

**Corollary 19** Let  $a \in N \cup \{*\}$ . Suppose  $\mathcal{S}$  is r.e. and not *a-supportable*. Then  $\mathcal{S} \notin \mathbf{SpEx}^a$ .

It can be shown that  $\mathcal{S}^0$  contains r.e. classes of functions that are not *a-supportable* for any  $a \in N \cup \{*\}$ . Hence, an immediate consequence of Corollary 19 is that  $\mathcal{S}^0 \notin \mathbf{SpEx}^a$  for any  $a \in N \cup \{*\}$ . This yields the following corollary.

#### Corollary 20

- (a)  $\mathbf{Ex}_0^0 - \mathbf{SpEx}^* \neq \emptyset$ .
- (b)  $\mathbf{MEx}^0 - \mathbf{SpEx}^* \neq \emptyset$ .

We now consider whether Part (b) of the above corollary can be further strengthened when mind changes are introduced in the class  $\mathbf{MEx}^0$ . Also, the following proposition is immediate.

**Proposition 21** Let  $a \in N \cup \{*\}$ .  $\mathbf{MEx}_0^a \subseteq \mathbf{SpEx}_0^a$ .

Note that, since  $\mathbf{SpEx}_b^a \subseteq \mathbf{MEx}_b^a$ , it follows that  $\mathbf{MEx}_0^a = \mathbf{SpEx}_0^a$ .

A natural question is what happens to the above proposition if we allow up to one mind change in nearly minimal identification. That is, we would like to know, if by allowing extra mind changes in nearly minimal identification, can we get out of severely parsimonious identification. The answer to this question turns out to be affirmative as implied by the following theorem.

**Theorem 22**  $\mathbf{MEx}_1^0 - \mathbf{SpEx}^* \neq \emptyset$ .

PROOF. We assume that  $\mathbf{M}_0, \mathbf{M}_1, \mathbf{M}_2, \dots$  is an r.e. sequence of learning machines given in Lemma 10. We will construct an acceptable programming system  $\psi$  and a class  $\mathcal{C}_\psi$  such that, with respect to this programming system,  $\mathcal{C}_\psi$  is in  $\mathbf{MEx}_1^0 - \mathbf{SpEx}_*^*$ . (Since the inference classes  $\mathbf{MEx}_1^0$  and  $\mathbf{SpEx}_*^*$  are acceptable programming system independent, we will have our result.)

Let  $\mathcal{C}_\psi^0 = \{f \in \mathcal{R} \mid (\forall x)[f(x) = \text{MinProg}_\psi(f)]\}$ .

Let  $\mathcal{C}_\psi^1 = \{f \in \mathcal{R} \mid (\exists y > 0)[f(y) = \text{MinProg}_\psi(f) \wedge (\forall x < y)[f(x) = f(0) \neq f(y)] \wedge (\forall x > y)[f(x) = f(y)]\}$ .

Let  $\mathcal{C}_\psi = \mathcal{C}_\psi^0 \cup \mathcal{C}_\psi^1$ .

It is easy to see that, for each acceptable programming system  $\psi$ ,  $\mathcal{C}_\psi \in \mathbf{MEx}_1^0$ .

We will now construct an acceptable programming system  $\psi$  such that  $\mathcal{C}_\psi \notin \mathbf{SpEx}_*^*$ . Our intention is to make  $\mathcal{C}_\psi^0$  infinite, forcing any machine which  $\mathbf{SpEx}_*^*$ -identifies  $\mathcal{C}_\psi^0$  to output arbitrarily large programs on constant functions. This in turn will allow us to construct members of  $\mathcal{C}_\psi^1$  diagonalizing against such machines. We now proceed formally.

Let,  $\psi_{4^i} = \varphi_i$ . Note that this makes  $\psi$  acceptable.

For all  $i$ , for all  $j$ , such that  $4^{2i+1} < j < 4^{2i+2}$ , for all  $x$ , let  $\psi_j(x) = j$ .

Let  $\mathcal{S}_i = \{\psi_j \mid 4^{2i+1} < j < 4^{2i+2}\}$ . Note that for any  $i$ , at least  $\text{card}(\mathcal{S}_i) - (4^{2i+1} + 1)$  of the functions in  $\mathcal{S}_i$  are in  $\mathcal{C}_\psi^0$ . This ensures that any  $\mathbf{M}$  that witnesses  $\mathcal{C}_\psi \in \mathbf{SpEx}_*^*$  must output arbitrarily large programs on functions in  $\mathbf{CONST} = \{f \mid (\forall x)[f(x) = f(0)]\}$ . We will use this fact to define  $\psi_j$ , where  $4^{2i} < j < 4^{2i+1}$ . Our intention is to use these  $\psi_j$  to define functions in  $\mathcal{C}_\psi^1$  diagonalizing against machines which output arbitrarily large programs on functions in  $\mathbf{CONST}$  (programs  $j$ , such that  $4^{2\langle k,l \rangle} < j < 4^{2\langle k,l \rangle+1}$ , will be used to diagonalize against machine  $\mathbf{M}_k$ , with fudge factor  $\varphi_l$ ).

Let  $g_i$  denote the constant function  $\lambda x.[i]$ , i.e., for all  $x$ ,  $g_i(x) = i$ .

Let  $X_{k,l} = \{i > 4^{2\langle k,l \rangle+1} \mid \varphi_l(4^{2\langle k,l \rangle+1}) \downarrow \wedge (\exists n)[\mathbf{M}_k(g_i[n]) > \varphi_l(4^{2\langle k,l \rangle+1})]\}$ .

Intuitively,  $X_{k,l}$  is used to collect the functions  $g_i$ , such that  $\mathbf{M}_k$  on  $g_i$  outputs a large program (a program larger than  $\varphi_l(4^{2\langle k,l \rangle+1})$ ). Note that if  $\mathbf{M}_k$ , using recursive fudge factor  $\varphi_l$ , witnesses that  $\mathcal{C}_\psi^0 \in \mathbf{SpEx}_*^*$  then  $X_{k,l}$  must be infinite (note that  $\varphi_l$  must be total and thus  $\varphi_l(4^{2\langle k,l \rangle+1})$  is defined).

Fix  $j$ , such that  $4^{2\langle k,l \rangle} < j < 4^{2\langle k,l \rangle+1}$ . We now define  $\psi_j$ . Let  $i$  be  $(j - 4^{2\langle k,l \rangle})$ -th element in some fixed 1–1 recursive enumeration of  $X_{k,l}$  (if no such element exists, then  $\psi_j$  is everywhere undefined). Let  $n$  be such that  $\mathbf{M}_k(g_i[n]) > \varphi_l(4^{2\langle k,l \rangle+1})$ . Let  $\psi_j$  be defined as follows:  $\psi_j(x) = i$ , if  $x \leq n$ ;  $\psi_j(x) = j$  otherwise.

We now show that  $\mathcal{C}_\psi \notin \mathbf{SpEx}_*^*$ . So suppose by way of contradiction that  $\mathbf{M}_k$  witnesses  $\mathcal{C}_\psi \in \mathbf{SpEx}_*^*$ , where the recursive fudge factor is  $\varphi_l$ . Without loss of generality we assume that  $\varphi_l$  is increasing. Now let us consider the functions  $\psi_j$ , where  $4^{2\langle k,l \rangle} < j < 4^{2\langle k,l \rangle+1}$ . It is easy to see (by the construction of these  $\psi_j$ 's) that each of these  $\psi_j$ 's are total, distinct, and on each of them  $\mathbf{M}_k$  outputs a program larger than  $\varphi_l(4^{2\langle k,l \rangle+1})$ . Also, at least one of these  $\psi_j$ 's is in  $\mathcal{C}_\psi^1$  (since at most  $4^{2\langle k,l \rangle} + 1$  of these  $\psi_j$ 's do not belong to  $\mathcal{C}_\psi^1$ ). This contradicts the assumption

that  $\mathbf{M}_k$  witnesses  $\mathcal{C}_\psi \in \mathbf{SpEx}_*^*$  where the recursive fudge factor is  $\varphi_l$ . Thus, no such  $\mathbf{M}_k$  can exist and we have  $\mathcal{C}_\psi \notin \mathbf{SpEx}_*^*$ .  $\blacksquare$

As a corollary to the above results, we have:

**Corollary 23** *Suppose  $a, b, c, d \in N \cup \{*\}$ .  $\mathbf{MEx}_b^a \subseteq \mathbf{SpEx}_d^c \iff (b = 0) \wedge (a \leq c)$ .*

### 3.2 Anomaly Hierarchy for Severe Parsimony

Using Proposition 21 and Theorem 9 we get the next result which says that there are collections of functions that can be severely parsimoniously identified with up to  $i + 1$  errors in the final program, but that cannot be identified by allowing only up to  $i$  errors in the final program.

**Theorem 24**  $(\forall i \in N)[\mathbf{SpEx}_0^{i+1} - \mathbf{Ex}^i \neq \emptyset]$ .

This result yields the following anomaly hierarchy that underscores the fact that allowing extra errors in the final program makes larger collections of functions identifiable by severely parsimonious machines.

**Corollary 25**  $\mathbf{SpEx} \subset \mathbf{SpEx}^1 \subset \mathbf{SpEx}^2 \subset \dots \subset \mathbf{SpEx}^*$ .

### 3.3 Mind Changes and Severe Parsimony

We now consider bounding the number of mind changes in severely parsimonious identification. We are able to establish the following result which says that there are collections of functions for which 0-error programs can be identified by severely parsimonious learners by allowing up to  $i + 1$  mind changes, but for which even a finite variant program cannot be  $\mathbf{Ex}$ -identified if up to only  $i$  mind changes are allowed.

**Theorem 26**  $(\forall i \in N)[\mathbf{SpEx}_{i+1} - \mathbf{Ex}_i^* \neq \emptyset]$ .

**PROOF.** We assume that  $\mathbf{M}_0, \mathbf{M}_1, \mathbf{M}_2, \dots$  is an r.e. sequence of learning machines given in Lemma 10. We will construct an acceptable programming system  $\psi$  and a class  $\mathcal{C}$  such that  $\mathcal{C} \in \mathbf{SpEx}_{i+1}^0 - \mathbf{Ex}_i^*$ .

The construction of  $\psi$  will be facilitated by a recursive function  $h$ . We will set  $h(0) = 0$ . For  $j \in N$ ,  $h(j)$  will denote the beginning of the  $j$ th group of programs. Program between  $h(j)$  and  $h(j+1)$  will be further divided into  $(j+2)$  groups of  $(2i+3)$  programs. Programs between  $h(j)$  and  $h(j+1)$  will be used for diagonalization against machine  $\mathbf{M}_j$ . We now proceed formally.

Let  $h(0) = 0$ , and  $h(j+1) = (j+2) \cdot (2i+3) + h(j) + 1$ .

Let  $\psi_{h(j)} = \varphi_j$ . This makes  $\psi$  an acceptable programming system.

Below, we define  $\psi_k$ , for  $k$  such that  $h(j) < k < h(j+1)$ ; these programs will be used for diagonalization against  $\mathbf{M}_j$ .

Let  $g$  be a function defined as follows:

For  $j \in N$ , for  $k \leq j+1$ , for  $s \leq 2i+2$ , let  $g(j, k, s) = h(j) + 1 + (2i+3) \cdot k + s$ . Intuitively, for a given  $j$ ,  $g$  is used to divide the programs between  $h(j)$  and  $h(j+1)$  into  $j+2$  groups of size  $2i+3$  each — the  $k$ -th group consisting of programs,  $g(j, k, 0), g(j, k, 1), \dots, g(j, k, 2i+2)$ . These programs will satisfy the following properties:

- (A)  $(\forall j)(\forall k \leq j+1)(\forall s \leq 2i+2)[\psi_{g(j,k,s)}(0) = j \wedge \psi_{g(j,k,s)}(1) = k]$ .
- (B) For all  $j \in N$ , for all  $k \leq j+1$  and for all  $s \leq 2i+2$ , if  $\psi_{g(j,k,s)} \in \mathcal{R}$ , then  $\psi_{g(j,k,s)}$  satisfies the following properties:

(B.1)  $(\forall x)[\psi_{g(j,k,s)} = s]$ .

(B.2)  $(\forall x, x' \mid 2 \leq x \leq x')[\psi_{g(j,k,s)}(x) \leq \psi_{g(j,k,s)}(x')]$ .

Thus, for  $x \geq 2$ ,  $\psi_{g(j,k,s)}$  is a nondecreasing function bounded by  $s$ .

(B.3)  $\text{card}(\{\psi_{g(j,k,s)}(x) \mid x \geq 2\}) \leq i + 2$ .

Note: If  $\psi_{g(j,k,s)} \notin \mathcal{R}$ , then it would be the case that  $(\forall x \geq 2)[\psi_{g(j,k,s)}(x) \uparrow]$ .

(C) For all  $j \in N$ , for all  $k \leq j + 1$ , there exists an  $s \leq 2i + 2$ , such that  $\psi_{g(j,k,s)} \in \mathcal{R}$  and  $\psi_{g(j,k,s)} \notin \mathbf{Ex}_i^*(\mathbf{M}_j)$ .

We will describe  $\psi_{g(j,k,s)}$  satisfying the above properties later. However, we first show how we can construct  $\mathcal{C} \in (\mathbf{SpEx}_{i+1}^0 - \mathbf{Ex}_i^*)$  using the above properties.

For each  $j$ , let  $k_j \leq j + 1$  be such that  $(\forall l \leq j)[\varphi_l(1) \neq k_j]$ . Note that, by the pigeonhole principle, there exists such a  $k_j$ . This property is needed to ensure that for all  $f \in \mathcal{C}$  (defined below),  $h(f(0)) < \text{MinProg}_\psi(f) < h(f(0) + 1)$ .

Let  $\mathcal{C} = \{\psi_{g(j,k_j,s)} \mid \psi_{g(j,k_j,s)} \in \mathcal{R} \wedge j \in N \wedge s \leq 2i + 2\}$ .

It follows immediately by Property (C) above that,  $\mathcal{C} \notin \mathbf{Ex}_i^*$ . Also using the fact that for all  $f \in \mathcal{C}$ ,  $h(f(0)) < \text{MinProg}_\psi(f) < h(f(0) + 1)$  and Property (B) above, it is easy to show that  $\mathcal{C} \in \mathbf{SpEx}_{i+1}^0$ . To see this note that a machine which on input  $f[n]$ , for  $n > 2$ , outputs  $g(f(0), f(1), f(n-1))$  witnesses that  $\mathcal{C} \in \mathbf{SpEx}_{i+1}^0$ .

Fix  $j \in N$  and  $k \leq j + 1$ . We now define  $\psi_{g(j,k,s)}$ , for  $s \leq 2i + 2$ .

**begin** {Definition of  $\psi_{g(j,k,s)}$ , for  $s \leq 2i + 2$ }

1. For  $s \leq 2i + 2$ , let  $\psi_{g(j,k,s)}(0) = j$ .

For  $s \leq 2i + 2$ , let  $\psi_{g(j,k,s)}(1) = k$ .

(The above satisfies Property (A).)

2. For  $x \geq 2$ , let  $\psi_{g(j,k,0)}(x) = 0$ .

3. Search for a  $t > 2$ , such that  $\mathbf{M}_j(\psi_{g(j,k,0)}[t]) \neq ?$ .

Let  $p = 0$ .

4. **for**  $s = 0$  to  $i$  **do**

4.1. Define  $\psi_{g(j,k,2s+1)}$  and  $\psi_{g(j,k,2s+2)}$  as follows.

$$\psi_{g(j,k,2s+1)}(x) = \begin{cases} \psi_{g(j,k,p)}(x), & \text{if } x < t; \\ 2s + 1, & \text{otherwise.} \end{cases}$$

$$\psi_{g(j,k,2s+2)}(x) = \begin{cases} \psi_{g(j,k,p)}(x), & \text{if } x < t; \\ 2s + 2, & \text{otherwise.} \end{cases}$$

4.2. Search for  $p' \in \{2s + 1, 2s + 2\}$  and  $t' > t$  such that  $\mathbf{M}_j(\psi_{g(j,k,p)}[t]) \neq \mathbf{M}_j(\psi_{g(j,k,p')}[t'])$ .

4.3. If and when such  $p', t'$  are found, let  $t = t'$  and  $p = p'$ .

**endfor**

**end** {Definition of  $\psi_{g(j,k,s)}$ , for  $s \leq i + 1$ }

We now show that the definition of  $\psi_{g(j,k,s)}$  satisfies clauses (B) and (C) above. Properties (B.1) and (B.2) are immediate from the construction Step 4.1. Property (B.3) holds since, for each  $s \leq i$ , at most one of  $2s + 1, 2s + 2$ , is placed in the range of any  $\psi_{g(j,k,\cdot)}$  (see Step 4.1).

We now show Property (C). Consider the following cases.

*Case 1:* Search at Step 3 does not succeed.

In this case, clearly  $\varphi_{g(j,k,0)} \notin \mathbf{Ex}_i^*(\mathbf{M}_j)$ .

*Case 2:* All the iterations of the for loop at Step 4 terminate.

In this case, for  $p'$  as found in Step 4.2 of the last iteration of the for loop,  $\mathbf{M}_j$  makes at least  $i+1$  mind changes on  $\psi_{g(j,k,p')}$  (since each iteration of the for loop forces at least one mind change by  $\mathbf{M}_j$ ).

*Case 3:* In the above construction, the for loop iteration with  $s = s'$ , is entered but not finished.

In this case  $\mathbf{M}_j(\psi_{g(j,k,2s'+1)}) = \mathbf{M}_j(\psi_{g(j,k,2s'+2)})$  and both  $\psi_{g(j,k,2s'+1)}$  and  $\psi_{g(j,k,2s'+2)}$  are total. Thus, at least one of  $\psi_{g(j,k,2s'+1)}$  and  $\psi_{g(j,k,2s'+2)}$  does not belong to  $\mathbf{Ex}_i^*(\mathbf{M}_j)$ .

From the above cases it follows that the construction satisfies Property (C).  $\blacksquare$

The above theorem yields the following hierarchy with the moral that allowing extra mind changes before the onset of convergence makes larger collections of functions identifiable with respect to severely parsimonious identification.

**Corollary 27** *Let  $a \in N \cup \{*\}$ .  $\mathbf{SpEx}_0^a \subset \mathbf{SpEx}_1^a \subset \dots \subset \mathbf{SpEx}_*^a$*

As a corollary to Theorems 24 and 26, we have

**Corollary 28** *Let  $a, b, c, d \in N \cup \{*\}$ . Suppose  $I \in \{\mathbf{Ex}, \mathbf{SpEx}, \mathbf{MEx}\}$ . Then,  $\mathbf{SpEx}_b^a \subseteq \mathbf{I}_d^c \iff (a \leq c) \wedge (b \leq d)$ .*

## 4 Non-Revolutionary Identification

As noted in the introductory section, the notion of severe parsimony has some drawbacks as a model for induction without paradigm shift. In particular, a severely parsimonious learner can issue conjectures that have wide variance in size because the only restriction on the size of the conjectures is that they be within a computable factor of the minimal size program for the function being learned. Now for a function with large minimal size program (together with a high growth computable “fudge” factor), the requirement of severe parsimony leaves the scope for conjectures to vary greatly in size. What seems to be needed is that the smallest size conjecture and the largest size conjecture on a function do not vary too much. This need motivates the notion of non-revolutionary identification.

**Definition 29** Let  $\mathbf{M}$  and  $f \in \mathcal{R}$  be given. Then  $\text{ProgSet}(\mathbf{M}, f) = \{\mathbf{M}(f[n]) \mid n \in N \wedge \mathbf{M}(f[n]) \neq ?\}$ .

**Definition 30** Let  $a, b \in N \cup \{*\}$ .

- (a)  $\mathbf{M}$  is *non-revolutionary* just in case there exists a recursive  $h$  such that, for each  $f \in \mathcal{R}$ ,  $\max(\text{ProgSet}(\mathbf{M}, f)) \leq h(\min(\text{ProgSet}(\mathbf{M}, f)))$ .
- (b)  $\mathbf{NrEx}_b^a = \{\mathcal{S} \mid (\exists \mathbf{M})[\mathbf{M} \text{ is non-revolutionary and } \mathcal{S} \subseteq \mathbf{Ex}_b^a(\mathbf{M})]\}$ .

In the above definition we assume that  $h(\infty) = \infty$ . This is needed for the case when  $\text{ProgSet}(\mathbf{M}, f)$  is empty (thus, when  $\text{ProgSet}(\mathbf{M}, f)$  is empty, the non-revolutionary constraint is satisfied). We consider the comparison between  $\mathbf{Ex}_b^a$ ,  $\mathbf{MEx}_b^a$ ,  $\mathbf{SpEx}_b^a$  and  $\mathbf{NrEx}_b^a$  for various values of  $a$  and  $b$ . To begin with, the following proposition immediately follows from the definition.

**Proposition 31**  $\mathbf{Ex}_0^a \subseteq \mathbf{NrEx}_0^a$ .

We first consider the case when it is possible for severely parsimonious identification to diagonalize against  $\mathbf{NrEx}$ . The next theorem shows that there are collections of functions for which a severely parsimonious machine can identify an error free program if it is allowed up to 1 mind change, but for which even a finite variant program cannot be identified by any non-revolutionary machine even if an unbounded number of mind changes are allowed.

**Theorem 32**  $\mathbf{SpEx}_1^0 - \mathbf{NrEx}_*^* \neq \emptyset$ .

PROOF. We will construct an acceptable programming system  $\psi$  and a class  $\mathcal{C}_\psi$  such that  $\mathcal{C}_\psi \in (\mathbf{SpEx}_1^0 - \mathbf{NrEx}_*^*)$ . Since the classes  $\mathbf{SpEx}_1^0$  and  $\mathbf{NrEx}_*^*$  are acceptable programming system independent, we have the theorem.

Let **ZERO** denote the everywhere 0 function.

Let  $\mathcal{C}_\psi = \{\mathbf{ZERO}\} \cup \{f \mid \min(\{x \mid f(x) \neq 0\}) = \text{MinProg}_\psi(f)\}$ .

It is easy to see that, for all  $\psi$ ,  $\mathcal{C}_\psi \in \mathbf{SpEx}_1^0$ .

We now construct a  $\psi$  such that  $\mathcal{C}_\psi \notin \mathbf{NrEx}_*^*$ .

Let  $g$  be defined as follows.

$$g(0) = 1$$

$$g(j+1) = 2 \cdot g(j) + 3$$

Note that the  $\text{card}(\{p \mid g(j) < p < g(j+1)\}) = g(j) + 2$ .

Let  $\psi_{g(j)} = \varphi_j$ . This makes  $\psi$  an acceptable programming system.

Let  $\psi_0 = \mathbf{ZERO}$ , the everywhere 0 function.

Let  $f_p$  denote the function,

$$f_p(x) = \begin{cases} 0, & \text{if } x < p; \\ p, & \text{otherwise.} \end{cases}$$

Note that for  $p \neq p'$ ,  $f_p \neq f_{p'}$ .

For  $p$  such that  $p > 0$ , and  $p$  is not in the range of  $g$ , let  $\psi_p = f_p$ .

Now, for each  $j$ , at least one of  $f_p$ ,  $g(j) < p < g(j+1)$ , belongs to  $\mathcal{C}_\psi$  (since there are  $g(j) + 2$  such  $f_p$ 's and thus at least one of them is not computed by any  $\psi$  program  $\leq g(j)$ ). It follows that  $\mathcal{C}_\psi$  contains infinitely many  $f_p$ 's.

We now show that  $\mathcal{C}_\psi \notin \mathbf{NrEx}_*^*$ . Suppose by way of contradiction that  $\mathbf{M}$  witnesses  $\mathbf{NrEx}_*^*$  identification of  $\mathcal{C}_\psi$ . Then since  $\mathbf{M}$  witnesses  $\mathbf{NrEx}_*^*$  identification of  $\{\mathbf{ZERO}\}$ , there exists a  $z$  such that  $\mathbf{M}(\mathbf{ZERO}[z]) \neq ?$ . Let  $h$  be an increasing function such that, for all  $f \in \mathcal{R}$ ,  $\max(\text{ProgSet}(\mathbf{M}, f)) \leq h(\min(\text{ProgSet}(\mathbf{M}, f)))$ . It follows that for all  $p > z$  and  $m > z$ ,  $\mathbf{M}(f_p[m]) \leq h(\mathbf{M}(\mathbf{ZERO}[z]))$ . But there are only finitely many programs  $\leq h(\mathbf{M}(\mathbf{ZERO}[z]))$ , and infinitely many  $f_p$ , such that  $p \geq n$  and  $f_p \in \mathcal{C}_\psi$ . Thus,  $\mathbf{M}$  cannot  $\mathbf{Ex}^*$ -identify all  $f_p \in \mathcal{C}_\psi$ . ■

Note that the above proof essentially shows that any class containing an accumulation point is not in  $\mathbf{NrEx}_*^0$ .

As a corollary to Proposition 31 and Theorems 24 and 32 we have the following.

**Corollary 33** *Suppose  $\mathbf{I} \in \{\mathbf{Ex}, \mathbf{MEx}, \mathbf{SpEx}\}$ . Then,*  
 $\mathbf{I}_b^a \subseteq \mathbf{NrEx}_d^c \iff (b = 0) \wedge (a \leq c)$ .

We now consider cases in which there are collections of functions that can be identified by non-revolutionary machines but cannot be identified by other strategies introduced in this paper.

As a corollary to Proposition 31 and results from the previous sections, we have the following.



**Corollary 34** Suppose  $n \in N$ .

- (a)  $\mathbf{NrEx}_0^{n+1} - \mathbf{Ex}_*^n \neq \emptyset$ .
- (b)  $\mathbf{NrEx}_0^0 - \mathbf{SpEx}_*^* \neq \emptyset$ .
- (c)  $\mathbf{NrEx}_0^0 - \mathbf{MEx}_n^* \neq \emptyset$ .

PROOF. Part (a) follows from the fact that  $\mathbf{Ex}_0^{n+1} \subseteq \mathbf{NrEx}_0^{n+1}$  (Proposition 31) and the fact that  $\mathbf{Ex}_0^{n+1} - \mathbf{Ex}_*^n \neq \emptyset$  (implied by Theorem 7(c).)

Part (b) follows from the fact that  $\mathbf{Ex}_0^0 \subseteq \mathbf{NrEx}_0^0$  (Proposition 31) and the fact that  $\mathbf{Ex}_0^0 - \mathbf{SpEx}_*^* \neq \emptyset$  (this is Corollary 20(b).)

Part (c) follows from the fact that  $\mathbf{Ex}_0^0 \subseteq \mathbf{NrEx}_0^0$  (Proposition 31) and the fact that  $\mathbf{Ex}_0^0 - \mathbf{MEx}_n^* \neq \emptyset$  (this is Theorem 9(c).) ■

Additionally, our proof of Theorem 26 also shows that the following result holds.

**Theorem 35**  $\mathbf{NrEx}_{i+1}^0 - \mathbf{Ex}_i^* \neq \emptyset$ .

PROOF. Proof of Theorem 26 actually also shows this theorem, since the machine given in the proof of Theorem 26 to  $\mathbf{SpEx}_{i+1}^0$ -identify  $\mathcal{C}$  also  $\mathbf{NrEx}_{i+1}^0$ -identifies  $\mathcal{C}$ . ■

As a corollary to the above theorem and Part (a) of Corollary 34, we have the following relationship between  $\mathbf{NrEx}_b^a$  and  $\mathbf{Ex}_d^c$  for various values of  $a, b, c, d$ .

**Corollary 36**  $\mathbf{NrEx}_b^a \subseteq \mathbf{Ex}_d^c$  iff  $a \leq c$  and  $b \leq d$ .

Since for  $n \in N$  and  $a \in N \cup \{*\}$ ,  $\mathbf{NrEx}_n^a \subseteq \mathbf{Ex}_n^a \subset \mathbf{MEx}^a$  (the latter inclusion implied by Theorem 9(b)), the only case left to consider is  $\mathbf{NrEx}_*^a$  versus  $\mathbf{MEx}_*^b$ . The next theorem helps settle this question; it shows that there are collections of functions that can be identified by non-revolutionary machines but cannot be identified in the nearly minimal sense even if the number of errors allowed in the final program is large but bounded. The proof is surprisingly a bit complex technically.

**Theorem 37** For  $n \in N$ ,  $\mathbf{NrEx}_*^0 - \mathbf{MEx}_*^n \neq \emptyset$ .

PROOF. For the ease of writing the proof we show only  $\mathbf{NrEx}^0 - \mathbf{MEx}^0 \neq \emptyset$ . The proof can be easily generalized to prove the theorem for  $n > 0$ , by using a cylindrification of the class  $\mathcal{C}$  defined below.

We construct an acceptable programming system  $\psi$  and a class  $\mathcal{C} \not\subseteq \mathbf{MEx}^0$ , such that  $\mathcal{C} \in \mathbf{NrEx}^0$  (with respect to programming system  $\psi$ ). Since  $\mathbf{NrEx}^0$  is acceptable programming system independent we have our theorem.

Let  $g$  be defined as follows.

$$\begin{aligned} g(0) &= 0 \\ g(i+1) &= g(i) + i + 2 \end{aligned}$$

Let  $\psi_{g(i)} = \varphi_i$ . This makes  $\psi$  an acceptable programming system.

We will be taking  $\mathcal{C} = \{\psi_j \mid \psi_j \in \mathcal{R} \wedge j \notin \text{range}(g)\}$ .

For  $j$  not in the range of  $g$ , the following properties will be satisfied by  $\psi_j$ :

- (A)  $\psi_j(0) = i$ , where  $g(i) < j < g(i+1)$ .
- (B) Either  $\psi_j \in \mathcal{R}$  or  $(\forall x > 0)[\psi_j(x) \uparrow]$ .
- (C) For all  $k, l$ , such that  $\varphi_l$  is total, there exist  $r \in N$  and  $s \leq \langle k, l, r \rangle$  such that  $\psi_{g(\langle k, l, r \rangle) + s + 1}$  is total but  $\psi_{g(\langle k, l, r \rangle) + s + 1} \notin \mathbf{MEx}(\mathbf{M}_k, \varphi_l)$ .

It is easy to see that  $\mathcal{C} \in \mathbf{NrEx}^0$ . This is so since one can easily construct a machine which, using Property (A), (B) above,  $\mathbf{Ex}$ -identifies  $f \in \mathcal{C}$  by using only the  $\psi$ -programs  $g(f(0)) + 1, g(f(0)) + 2, \dots, g(f(0) + 1) - 1$ . Also by Property (C) above, we will have that  $\mathcal{C} \notin \mathbf{MEx}^0$ . Thus, construction of  $\psi_j$  satisfying the above properties would complete the proof.

For each  $k, l$ , by the implicit use of the parametric recursion theorem [37], we now define,

- $\varphi_{e(k,l)}$  (and thus  $\psi_{g(e(k,l))}$ ).
- functions  $\psi_j$ , where for some  $r \in N$  and  $s \leq \langle k, l, r \rangle$ ,  $j = g(\langle k, l, r \rangle) + s + 1$ .

We will do so in such a way that Property (C) above is satisfied. This essentially uses the idea used by Chen [16] to show that functions of finite support cannot be  $\mathbf{MEx}$ -identified.

Intuitively,  $e(k, l)$ , plays the role of  $e$  in Chen's construction. The functions of finite support "actually needed" for diagonalization are computed by programs  $j$  of the form  $g(\langle k, l, r \rangle) + s + 1$ .

**begin** {Definition of  $\varphi_{e(k,l)}$  and, for all  $r \in N$  and  $s \leq \langle k, l, r \rangle$ ,  $\psi_{g(\langle k, l, r \rangle) + s + 1}$ }

1. For all  $r, s \leq \langle k, l, r \rangle$ , let  $\psi_{g(\langle k, l, r \rangle) + s + 1}(0) = \langle k, l, r \rangle$ .

This ensures that Property (A) is satisfied.

2. Wait until  $\varphi_l(e(k, l)) \downarrow$ .

Let  $\text{bnd} = \varphi_l(e(k, l))$

3. Pick  $r$  to be so large that  $\langle k, l, r \rangle > \text{bnd} + 2$ .

We will use the programs  $g(\langle k, l, r \rangle) + s + 1$ , where  $s \leq \langle k, l, r \rangle$ , for diagonalization against

$\mathbf{MEx}$ -identification by  $\mathbf{M}_k$ , using  $\varphi_l$  as the recursive bound.

4. For  $r' \neq r$ , and  $s' \leq \langle k, l, r' \rangle$ ,  $\psi_{g(\langle k, l, r' \rangle) + s' + 1}(x)$  is undefined for  $x > 0$ .
5. Let  $\psi_{g(\langle k, l, r \rangle) + 1}$  be defined as follows:

$$\psi_{g(\langle k, l, r \rangle) + 1}(x) = \begin{cases} \langle k, l, r \rangle, & \text{if } x = 0; \\ 0, & \text{otherwise.} \end{cases}$$

6. Let  $\text{Cancel} = \emptyset$ ,  $t = 0$ .

Let  $\varphi_{e(k,l)}(0) = \langle k, l, r \rangle$ .

**for**  $w = 1$  **to**  $\langle k, l, r \rangle$  **do**

**loop**

Dovetail Steps 6.1 and 6.2 until one of them succeeds. If Step 6.1 succeeds before Step 6.2 does, if ever, then go to Step 6.3. If Step 6.2 succeeds before Step 6.1 does, if ever, then go to Step 6.4.

- 6.1. Search for  $p \leq \text{bnd}$  such that  $p \notin \text{Cancel}$ , and  $y > t$ ,  $\varphi_p(y) \downarrow$ .

- 6.2. Search for  $t' > t$ , such that  $\mathbf{M}_k(\varphi_{g(\langle k, l, r \rangle) + w}[t']) > \text{bnd}$ .

- 6.3. If and when such a  $p, y$  in Step 6.1. is found define  $\psi_{g(\langle k, l, r \rangle) + w + 1}$  as follows:

$$\psi_{g(\langle k, l, r \rangle) + w + 1}(x) = \begin{cases} \psi_{g(\langle k, l, r \rangle) + w}(x), & \text{if } x < y; \\ \varphi_p(x) + 1, & \text{if } x = y; \\ 0, & \text{otherwise.} \end{cases}$$

Let  $\text{Cancel} = \text{Cancel} \cup \{p\}$ .

For  $x \leq y$ , let  $\varphi_{e(k,l)}(x) = \psi_{g(\langle k, l, r \rangle) + w + 1}(x)$ .

Let  $t = y$ , as found in Step 6.1.

Go to Step 6.5.

- 6.4. For  $x < t'$ , let  $\varphi_{e(k,l)}(x) = \psi_{g(\langle k, l, r \rangle) + w}(x)$ .

Let  $t = t'$ .

**forever**

6.5. Continue with the next iteration of for loop.

**endfor**

**end** {Definition of  $\varphi_{e(k,l)}$  and, for all  $r \in N$  and  $s \leq \langle k, l, r \rangle$ ,  $\varphi_{g(\langle k, l, r \rangle) + s + 1}$  }

The above construction clearly, satisfies properties (A) and (B) above. We now show that it satisfies Property (C). So suppose  $k$  and  $l$  are given such that  $\varphi_l$  is total. Thus, Step 2 in the construction succeeds in seeing that  $\varphi_l(e(k, l)) \downarrow$ . Let  $\text{bnd}$ ,  $r$  be as defined in Steps 2 and 3 of the construction. Clearly, the for loop can be executed at most  $\text{bnd} + 1$  times, since the search in Step 6.1 can succeed only  $\text{bnd} + 1$  times (after which Cancel contains all the elements  $\leq \text{bnd}$ ). Thus, there is a last iteration of the for loop. So suppose the last iteration of the for loop is with index value  $w$ . Now clearly,  $\psi_{g(\langle k, l, r \rangle) + w}$  is a total function and thus a member of  $\mathcal{C}$ . We will show that  $\mathbf{M}_k$  does not **MEx**-identify  $\psi_{g(\langle k, l, r \rangle) + w}$ , with the recursive fudge factor  $\varphi_l$ . We consider two cases:

*Case 1:* Step 6.2 succeeds finitely often.

In this case either  $\mathbf{M}_k$  does not converge on  $\psi_{g(\langle k, l, r \rangle) + w}$ , or it converges to a program  $\leq \text{bnd}$ . However, by Step 6.1 and 6.3 all programs  $\leq \text{bnd}$  are convergently different from  $\psi_{g(\langle k, l, r \rangle) + w}$  or compute non-total functions.

*Case 2:* Step 6.2 succeeds infinitely often.

In this case clearly,  $\varphi_{e(i,j)} = \psi_{g(\langle k, l, r \rangle) + w}$ . However,  $\mathbf{M}_k$  on  $\psi_{g(\langle k, l, r \rangle) + w}$  infinitely often outputs a program  $> \text{bnd} = \varphi_l(e(k, l))$  (since Step 6.2 succeeds infinitely often). Thus,  $\mathbf{M}_k$  does not **MEx**-identify  $\psi_{g(\langle k, l, r \rangle) + w}$  with the recursive fudge factor  $\varphi_l$ .

From the above cases we have that  $\mathbf{M}_k$  does not witness **MEx** identification of  $\psi_{g(\langle k, l, r \rangle) + w}$ , and thus Property (C) above is satisfied.  $\blacksquare$

As a corollary to Theorem 37, Theorem 9 and Corollary 34 we have,

**Corollary 38**  $\text{NrEx}_b^a \subseteq \text{MEx}_d^c \iff (d = *) \wedge (c \geq a) \wedge [(b \neq *) \vee (c = *)]$ .

## 5 Severe Parsimony with Bc and Fex

We now extend the notion of severe parsimony to behaviorally correct identification and vacillatory identification. We first introduce these two criteria.

**Definition 39** [15, 5] Let  $a \in N \cup \{*\}$ .

- (a)  $\mathbf{M}$  **Bc**<sup>a</sup>-identifies  $f$  (written:  $f \in \mathbf{Bc}^a(\mathbf{M})$ ) just in case,  $(\forall^\infty n)[\varphi_{\mathbf{M}(f[n])} =^a f]$ . We define the class  $\mathbf{Bc}^a = \{\mathcal{S} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{S} \subseteq \mathbf{Bc}^a(\mathbf{M})]\}$ .
- (b)  $\mathbf{M}$  **Fex**<sup>a</sup>-identifies  $f$  (written:  $f \in \mathbf{Fex}^a(\mathbf{M})$ ) just in case there exists a nonempty finite set  $D$  of  $a$ -error programs for  $f$  such that for all but finitely many  $n$ ,  $\mathbf{M}(f[n]) \in D$ . We define the class  $\mathbf{Fex}^a = \{\mathcal{S} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{S} \subseteq \mathbf{Fex}^a(\mathbf{M})]\}$ .

The next definition adapts the notion of severe parsimony to **Bc**<sup>a</sup>-identification and **Fex**<sup>a</sup>-identification.

**Definition 40** Let  $a \in N \cup \{*\}$ .

- (a)  $\mathbf{SpBc}^a = \{\mathcal{S} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathbf{M} \text{ is severely parsimonious on } \mathcal{S} \text{ and } \mathbf{M} \mathbf{Bc}^a\text{-identifies } \mathcal{S}]\}$ .

(b)  $\mathbf{SpFex}^a = \{\mathcal{S} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathbf{M} \text{ is severely parsimonious on } \mathcal{S} \text{ and } \mathbf{M} \text{ Fex}^a\text{-identifies } \mathcal{S}]\}$ .

The following proposition is immediate.

**Proposition 41**  $(\forall a \in N \cup \{*\})[\mathbf{SpBc}^a = \mathbf{SpFex}^a]$ .

The  $a = 0$  case of the following theorem can be easily handled with a trick of Barzdin and Podeniaks [5] and the  $a = *$  case with a trick of Case and Smith [15]; however, the proof of the other cases is non-trivial.

**Theorem 42**  $(\forall a \in N \cup \{*\})[\mathbf{SpFex}^a = \mathbf{SpEx}^a]$ .

PROOF. For  $a = *$ , the theorem follows using the proof of Case and Smith [15] for  $\mathbf{Fex}^* = \mathbf{Ex}^*$  (the simulation done in [15] maintains the size of the programs output within a recursive factor of the maximum program output).

Now suppose  $a \in N$ . Suppose  $\mathbf{M}$  is given. We will construct a machine  $\mathbf{M}'$  which  $\mathbf{Ex}^a$ -identifies the class of functions which are  $\mathbf{Fex}^a$ -identified by  $\mathbf{M}$ . Moreover  $\mathbf{M}'$  will satisfy the following property:

There exists a recursive function  $h$  such that, for any  $f \in \mathcal{R}$ , the following two conditions are satisfied:

- $\max(\text{ProgSet}(\mathbf{M}', f)) \leq h(\max(\text{ProgSet}(\mathbf{M}, f)))$  and
- $\min(\text{ProgSet}(\mathbf{M}', f)) \geq \min(\text{ProgSet}(\mathbf{M}, f))$ .

This would imply the theorem.

Intuitively, we would like to do the Case and Smith [15] simulation of  $\mathbf{Fex}$ -identification, except that we do patches in a slightly different way (see details below). This would limit the variation in program sizes.

We first define some functions useful in the definition of  $\mathbf{M}'$ .

Let  $\text{unionpatch}$  be a recursive function (defined implicitly using the s-m-n theorem [37]) such that  $\varphi_{\text{unionpatch}(S,E,k)}$ , where  $k \in N$  and  $S, E$  are finite subsets of  $N$ , can be defined as follows.

$\varphi_{\text{unionpatch}(S,E,k)}(x)$

**if**  $x \in E$ , **then**

Output  $\varphi_k(x)$ .

**else**

Search for a  $j \in S$  such that  $\varphi_j(x) \downarrow$ .

Output  $\varphi_j(x)$  for first such  $j$  found.

**endif**

End  $\varphi_{\text{unionpatch}(S,E,j)}(x)$ .

Intuitively,  $\varphi_{\text{unionpatch}(S,E,j)}$  computes the union of programs in  $S$ , except at inputs from the set  $E$ , where it computes the output based on a specific program  $k$ .

Let  $\text{convrr}(S) = \{x \mid (\exists j, j' \in S)[\varphi_j(x) \downarrow \neq \varphi_{j'}(x) \downarrow]\}$ .

Note that if, for some  $f \in \mathcal{R}$ , for all  $j \in S$ ,  $\varphi_j =^a f$ , then  $\text{card}(\text{convrr}(S)) \leq a \cdot \text{card}(S)$ .

Let  $\text{upatchtwo}$  be a recursive function (implicitly defined using s-m-n theorem [37]) such that  $\varphi_{\text{upatchtwo}(S,i,k)}$  maybe defined as follows. (We assume without loss of generality that  $\text{upatchtwo}(S, i, k) \geq \max(S)$ ).

$\varphi_{\text{upatchtwo}(S,i,k)}$

1. Search for a set  $E$  of cardinality  $i$  such that  $E \subseteq \{x \mid (\exists j, j' \in S)[\varphi_j(x) \downarrow \neq \varphi_{j'}(x) \downarrow]\}$ .  
Note that such a set  $E$  may not exist if  $\text{card}(\{x \mid (\exists j, j' \in S)[\varphi_j(x) \downarrow \neq \varphi_{j'}(x) \downarrow]\}) < i$ . In that case  $\varphi_{\text{upatchtwo}(S,i,k)}(x) \uparrow$ , for all  $x$ .
2. If and when such an  $E$  is found, let  $\varphi_{\text{upatchtwo}(S,i,k)} = \varphi_{\text{unionpatch}(S,E,k)}$ .

End  $\varphi_{\text{upatchtwo}}$

Intuitively,  $\varphi_{\text{upatchtwo}(S,i,k)}$  assumes that  $\text{card}(\text{convrr}(S)) = i$ , and then just simulates  $\varphi_{\text{unionpatch}(S,E,k)}$ , for  $E$ , a subset of  $\text{convrr}(S)$  of size  $i$ .

Now suppose  $\mathbf{M}$  is given. Let  $\mathbf{M}'$  be defined as follows:

$\mathbf{M}'(f[m])$

Let  $P = \{\mathbf{M}(f[m']) \mid m' \leq m\}$ .

Let  $S = \{i \in P \mid \text{card}(\{x < m \mid \Phi_i(x) \leq m \wedge \varphi_i(x) \neq f(x)\}) \leq a\}$ .

Let  $E = \{x < n \mid (\exists j \in S)[\Phi_j(x) \leq m \wedge \varphi_j(x) \neq f(x)]\}$ .

**if**  $S = \emptyset$ ,

**then**

output  $\mathbf{M}'(f[m-1])$  (if  $m = 0$  then output ?).

**else**

output  $\text{upatchtwo}(S, \text{card}(E), p)$ , where  $p$  is the (least) program in  $S$  which minimizes  $\text{card}(\{y \in E \mid \Phi_p(y) > m \vee \varphi_p(y) \neq f(y)\})$ .

**endif**

End  $\mathbf{M}'(f[n])$

We first show that  $\mathbf{M}'$   $\mathbf{Ex}^a$ -identifies every function that is  $\mathbf{Fex}^a$ -identified by  $\mathbf{M}$ . To see this suppose  $f \in \mathbf{Fex}^a(\mathbf{M})$  is given.

Let  $P = \text{ProgSet}(\mathbf{M}, f)$ .

Let  $S = \{j \in P \mid \text{card}(\{x \mid \varphi_j(x) \downarrow \neq f(x)\}) \leq a\}$ .

Let  $E = \{x \mid (\exists j \in S)[\varphi_j(x) \downarrow \neq f(x)]\}$ .

Let  $p$  be the (least) element of  $S$  which minimizes,  $\text{card}(\{y \in E \mid \Phi_p(y) \uparrow \vee \varphi_p(y) \downarrow \neq f(y)\})$ .

It is easy to see that  $\mathbf{M}'(f) \downarrow = \text{upatchtwo}(S, \text{card}(E), p)$ , and

$$\text{card}(\{x \mid \varphi_{\text{upatchtwo}(S, \text{card}(E), p)}(x) \neq f(x)\}) \leq \min(\{\text{card}(\{x \mid \varphi_j(x) \neq f(x)\}) \mid j \in S\}) \leq a$$

(since  $S$  contains an  $a$  error program for  $f$ ). Thus,  $\mathbf{M}'$   $\mathbf{Ex}^a$ -identifies  $f$ .

We now show the bound on the elements of  $\text{ProgSet}(\mathbf{M}', f)$  as claimed, i.e., we show that, there exists a recursive function  $h$  such that, for any  $f \in \mathcal{R}$ ,  $\max(\text{ProgSet}(\mathbf{M}', f)) \leq h(\max(\text{ProgSet}(\mathbf{M}, f)))$  and  $\min(\text{ProgSet}(\mathbf{M}', f)) \geq \min(\text{ProgSet}(\mathbf{M}, f))$ .

Due to the fact that  $\text{upatchtwo}(S, i, j) \geq \max(S)$ , we have, for any  $f \in \mathcal{R}$  and  $m \in N$ ,  $\mathbf{M}'(f[m]) \geq \min(\text{ProgSet}(\mathbf{M}, f))$ . Thus,  $\min(\text{ProgSet}(\mathbf{M}', f)) \geq \min(\text{ProgSet}(\mathbf{M}, f))$ .

Let  $h$  be a function defined as follows:

$$h(j) = \max(\{\text{upatchtwo}(S, i, j') \mid S \subseteq \{x \leq j\} \wedge i \leq a * \text{card}(S) \wedge j' \in S\}).$$

Clearly,  $h$  is a recursive function. Moreover, for all  $f \in \mathcal{R}$  and  $m \in N$ , such that  $\mathbf{M}'(f[m]) \neq ?$ , we have that,  $\mathbf{M}'(f[m]) \leq h(\max(\text{ProgSet}(\mathbf{M}, f)))$ . Thus,

$$\max(\text{ProgSet}(\mathbf{M}', f)) \leq h(\max(\text{ProgSet}(\mathbf{M}, f))).$$

The bound on  $\text{ProgSet}(\mathbf{M}', f)$  is as claimed, and hence the theorem follows. ■

Note that Proposition 41 and the proof of Theorem 42 also work for the non-revolutionary versions of  $\mathbf{Fex}^a$  and  $\mathbf{Bc}^a$ .

## 6 Future Work

Besides the proposed handling of the cases of language learning and of slightly less conservative, non paradigm-shifting criteria in which the fudge factor functions  $h$  are allowed to be limiting-recursive [14], it would be interesting to investigate models of paradigm shifts in relation to massive changes in program control structure [34, 35, 38] and to connect paradigm shifts to the need, in some cases, for training sequences [1].

The present paper considered machine induction without revolutionary paradigm shifts where the learning machine was allowed to conjecture hypotheses from an acceptable programming system. An interesting direction to consider is a similar investigation for identification in non-standard programming systems. This direction is likely to be particularly interesting since several non-standard programming systems have been shown to be particularly suited to identification (for example, see Wiehagen [44, 43]).

## Acknowledgements

We are grateful to Raghu Raghavan who suggested to the second author to look at the phenomenon of paradigm shift in science. The authors' discussion of the difficulties of formalizing paradigm shift eventually led to the present paper.

We are also grateful to the referee for many helpful suggestions, and especially for pointing out the investigation of machine induction without paradigm shifts in the context of non-standard programming systems.

This research was partially supported by a grant from the Australian Research Council.

## References

- [1] D. Angluin, W. Gasarch, and C. Smith. Training sequences. *Theoretical Computer Science*, 66(3):255–272, 1989.
- [2] G. Baliga, J. Case, S. Jain, and M. Suraj. Machine learning of higher order programs. *Journal of Symbolic Logic*, 59(2):486–500, June 1994.
- [3] J. M. Barzdin and R. Freivalds. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.
- [4] J. M. Barzdin and R. Freivalds. Prediction and limiting synthesis of recursively enumerable classes of functions. *Latvijas Valsts Univ. Zinatm. Raksti*, 210:101–111, 1974.
- [5] J. M. Barzdin and K. Podnieks. The theory of inductive inference. In *Mathematical Foundations of Computer Science, High Tatras, Czechoslovakia*, pages 9–15, 1973.
- [6] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.
- [7] M. Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14:322–336, 1967.
- [8] M. Blum. On the size of machines. *Information and Control*, 11:257–265, 1967.

- [9] J. Case. The power of vacillation. In D. Haussler and L. Pitt, editors, *Proceedings of the Workshop on Computational Learning Theory*, pages 133–142. Morgan Kaufmann Publishers, Inc., 1988. Journal version being revised as per referees’ suggestions for possible publication in *SIAM Journal on Computing*.
- [10] J. Case. The power of vacillation in language learning. Technical Report 93-08, University of Delaware, 1992. Expands on [9]; journal version being revised for possible publication in *SIAM Journal on Computing*.
- [11] J. Case, S. Jain, and Suzanne Ngo Manguelle. Refinements of inductive inference by popperian and reliable machines. *Kybernetika*, 30(1):23–52, 1994.
- [12] J. Case, S. Jain, and A. Sharma. On learning limiting programs. *International Journal of Foundations of Computer Science*, 3(1):93–115, March 1992.
- [13] J. Case, S. Jain, and A. Sharma. Vacillatory learning of nearly minimal size grammars. *Journal of Computer and System Sciences*, 49(2):189–207, October 1994.
- [14] J. Case, S. Jain, and M. Suraj. Not-so-nearly-minimal-size program inference. In Klaus P. Jantke and Steffen Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *Lecture Notes in Artificial Intelligence*, pages 77–96. Springer-Verlag, 1995.
- [15] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.
- [16] K. Chen. Tradeoffs in inductive inference of nearly minimal sized programs. *Information and Control*, 52:68–86, 1982.
- [17] R. Freivalds. Minimal Gödel numbers and their identification in the limit. In *Proceedings of the International Conference on Mathematical Foundations of Computer Science, Mariánské Lázně*, pages 219–225. Springer-Verlag, 1975. Lecture Notes in Computer Science 32.
- [18] R. Freivalds. Inductive inference of minimal programs. In M. Fulk and J. Case, editors, *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 3–20. Morgan Kaufmann Publishers, Inc., August 1990.
- [19] R. Freivalds and E. B. Kinber. Limit identification of minimal Gödel numbers. *Theory of Algorithms and Programs 3; Riga 1977*, pages 3–34, 1977.
- [20] M. Fulk. Robust separations in inductive inference. *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri*, pages 405–410, 1990.
- [21] M. A. Fulk and S. Jain. Approximate inference and scientific method. *Information and Computation*, 114(2):179–191, November 1994.
- [22] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- [23] S. Jain. On a question about learning nearly minimal programs. *Information Processing Letters*, 53(1):1–4, January 1995.
- [24] S. Jain and A. Sharma. Program size restrictions in computational learning. *Theoretical Computer Science A*, 127(2):351–386, May 1994.
- [25] S. Jain and A. Sharma. Prudence in vacillatory language identification. *Mathematical Systems Theory*, 28(3):267–279, May-June 1995.
- [26] Thomas Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, 1970.

- [27] S. Lange and P. Watson. Machine discovery in the presence of incomplete or ambiguous data. In K. Jantke and S. Arikawa, editors, *Algorithmic Learning Theory*, volume 872 of *Lecture Notes in Artificial Intelligence*, pages 438–452. Springer-Verlag, Berlin, Reinhardtbrunn Castle, Germany, October 1994.
- [28] M. Machtey and P. Young. *An Introduction to the General Theory of Algorithms*. North Holland, New York, 1978.
- [29] A. Meyer. Program size in restricted programming languages. *Information and Control*, 21:382–394, 1972.
- [30] A. Meyer and D. Ritchie. The complexity of loop programs. In *Proceedings of the 22nd National ACM Conference*, pages 465–469. Thomas Book Co., 1967.
- [31] Ernst E. Moody. *The Logic of William of Ockham*. New York, Sheed and Ward Inc., 1935.
- [32] D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, Cambridge, Mass., 1986.
- [33] H. Putnam. Probability and confirmation. In *Mathematics, Matter, and Method*. Cambridge University Press, 1975.
- [34] G. Riccardi. *The Independence of Control Structures in Abstract Programming Systems*. PhD thesis, SUNY/ Buffalo, 1980.
- [35] G. Riccardi. The independence of control structures in abstract programming systems. *Journal of Computer and System Sciences*, 22:107–143, 1981.
- [36] H. Rogers. Gödel numberings of partial recursive functions. *Journal of Symbolic Logic*, 23:331–341, 1958.
- [37] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1967. Reprinted, MIT Press 1987.
- [38] J. Royer. *A Connotational Theory of Program Structure*. Lecture Notes in Computer Science 273. Springer Verlag, 1987.
- [39] J. Royer and J. Case. *Subrecursive Programming Systems: Complexity and Succinctness*. Progress in Theoretical Computer Science. Birkhäuser Boston, 1994.
- [40] N. Shapiro. 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, 1971.
- [41] W. M. Thorburn. Occam’s Razor. *Mind*, pages 287–288, 1915.
- [42] W. M. Thorburn. The myth of Occam’s Razor. *Mind*, pages 345–353, 1918.
- [43] R. Wiehagen. On the complexity of program synthesis from examples. *Elektronische Informationsverarbeitung und Kybernetik*, 22:305–323, 1986.
- [44] R. Wiehagen. A thesis in inductive inference. In P. Schmitt J. Dix, K. Jantke, editor, *Nonmonotonic and Inductive Logic, 1st International Workshop, Karlsruhe, Germany*, volume 543 of *Lecture Notes in Artificial Intelligence*, pages 184–207. Springer Verlag, 1990.