

Not-So-Nearly-Minimal-Size Program Inference ^{*} (Preliminary Report)

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

Sanjay Jain
Institute of Systems Science
National University of Singapore
Singapore 0511
Republic of Singapore

Abstract

Freivalds defined an acceptable programming system independent criterion for learning programs for functions in which the final programs were required to be both correct and “nearly” minimal size, i.e, within a computable function of being purely minimal size. Kinber showed that this parsimony requirement on final programs severely limits learning power. Nonetheless, in, for example, scientific inference, parsimony is considered highly desirable. A *lim-computable function* is (by definition) one computable by a procedure allowed to change its mind finitely many times about its output. Investigated is the possibility of assuaging somewhat the limitation on learning power resulting from requiring parsimonious final programs by use of criteria which require the final, correct programs to be “not-so-nearly” minimal size, e.g., to be within a lim-computable function of actual minimal size. It is interestingly shown that some parsimony in the final program is thereby retained, yet learning power strictly increases. Also considered are lim-computable functions as above but for which notations for constructive ordinals are used to bound the number of mind changes allowed regarding the output. This is a variant of an idea introduced by Freivalds and Smith. For this ordinal complexity bounded version of lim-computability, the power of the resultant learning criteria form strict infinite hierarchies intermediate between the computable and the lim-computable cases. Many open questions are also presented.

1 Introduction

Freivalds [Fre75] defined an acceptable programming system [Rog58, Rog67, MY78] independent criterion for learning programs for functions in which the final programs were required to be both correct and “nearly” minimal size, i.e, within a computable function of being purely minimal size. Kinber [Kin74] announced that this parsimony requirement on final programs severely limited learning power. Refinements for which final programs are allowed to be anomalous [BB75, CS83] appear in [Che81, Che82]. The language learning case is considered in [CJS89]. More stringent parsimony requirements on final programs have been studied too [Fre75, Kin74, FK77, Kin77b, Kin83] [Fre90, JS91], for example, in which the final programs

^{*}Email addresses of the authors are: case@cis.udel.edu, suraj@cis.udel.edu, and sanjay@iss.nus.sg, respectively.

are required to be strictly minimal size. These parsimony restrictions even further limit learning power in ways which are interestingly dependent on the underlying acceptable programming system.

In, for example, scientific inference, parsimony is considered highly desirable; however, the above mentioned results indicate that even weak parsimony restrictions limit learning or inferring power. To begin explaining the results of the present paper: a *lim-computable function* is (by definition) one computable by a procedure allowed to change its mind finitely many times about its output. Investigated in this paper is the possibility of assuaging somewhat the limitation on learning power resulting from requiring parsimonious final programs by use of criteria which require the final, correct programs to be “not-so-nearly” minimal size, i.e., to be within a lim-computable function of actual minimal size. It is interestingly shown (Theorem 4 below) that some parsimony in the final program is thereby retained, yet learning power strictly increases.

Proposition 2, implies that, adding another limit to the parsimony bounding functions results in no parsimony being preserved (and no loss of learning power).

In Section 6 we consider some refinements of our learning criteria. Especially interesting is a refinement in Section 6.2 in which, for the lim-computable functions, notations for constructive ordinals [Rog67] are used to bound the number of mind changes allowed regarding the output. This is a variant of an idea introduced by Freivalds and Smith [FS91]. For this ordinal complexity bounded version of lim-computability, the power of the resultant learning criteria form strict infinite hierarchies intermediate between the computable and the lim-computable cases. Interesting open questions are also presented regarding just how fine are the learning criteria hierarchies generated by these ordinal complexity bounds.

2 Notation

N denotes the set of natural numbers, $\{0, 1, 2, 3, \dots\}$. $i, j, k, m, n, p, q, s, t, w, x, y, z$ (with or without subscripts, superscripts, ...) range over N . ‘*’ denotes a non-member of N such that $(\forall n \in N)[n < * < \infty]$ (‘*’ represents ‘unbounded but finite’). a, b, c, d , similarly, range over $N \cup \{*\}$. $x \dot{-} y$ denotes $\max(\{0, x - y\})$.

\emptyset denotes the empty set. $\in, \notin, \subseteq, \subset$ respectively denote ‘is a member of’, ‘is not a member of’, ‘is a subset of’ and ‘is a proper subset of’. \uparrow denotes ‘is undefined’. \downarrow denotes ‘is defined’.

For S , a subset of N , $\text{card}(S)$ denotes the cardinality of S . So then, ‘ $\text{card}(S) \leq *$ ’ means that $\text{card}(S)$ is finite. $\max(S)$ and $\min(S)$ denote, respectively, the maximum and minimum of the set S , where $\max(\emptyset) = 0$ and $\min(\emptyset) = \infty$. f, g, h with or without decorations range over *total* functions with arguments and values from N . For $a \in (N \cup \{*\})$, if η_1 and η_2 are partial functions, then $\eta_1 =^a \eta_2$ means that $\text{card}(\{x \mid \eta_1(x) \neq \eta_2(x)\}) \leq a$. $\text{domain}(\eta)$ and $\text{range}(\eta)$ respectively denote the domain and range of the partial function η . The set of all total computable functions of one variable is denoted by \mathcal{R} . \mathcal{C} and \mathcal{S} , with or without decorations, ranges over subsets of \mathcal{R} .

φ denotes a fixed *acceptable* programming system for the partial computable functions: $N \rightarrow N$ [Rog58, Rog67, MY78, Ric80, Ric81, Roy87, Mar89]. φ_p denotes the partial computable function computed by program p in the φ -system; W_p denotes the domain of φ_p . Φ denotes an arbitrary fixed Blum complexity measure for the φ -system [Blu67]. For a computable function f , $\text{MinProg}(f) \stackrel{\text{def}}{=} \min(\{p \mid \varphi_p = f\})$.

The quantifier ‘ \forall^∞ ’ means ‘for all but finitely many’; ‘ \exists^∞ ’ means ‘there exists infinitely many’ and ‘ $\exists!$ ’ means ‘there exists a unique’. σ ranges over finite initial segments of total functions. Any unexplained notation is from [Rog67].

3 Explanatory Function Identification

A *learning machine* [Gol67, BB75, CS83] is a computable mapping from the set of all finite initial segments of total functions: $N \rightarrow N$ into $N \cup \{?\}$. Natural number outputs are interpreted as programs in the φ -system. Initially, a learning machine is allowed to output ?'s to indicate that it has not decided on its first program output yet, but once it outputs some program, it is not allowed to output ?'s again. $f[n]$ denotes the finite initial segment $((0, f(0)), (1, f(1)), \dots, (n-1, f(n-1)))$. We say that $\mathbf{M}(f)$ *converges to* p (written $\mathbf{M}(f)\downarrow = p$) iff $(\exists p)(\forall^\infty n)[\mathbf{M}(f[n]) = p]$; $\mathbf{M}(f)$ is undefined if no such p exists.

Definition 1 ([Gol67, BB75, CS83]) Suppose $a, b \in N \cup \{*\}$.

- (a) $\mathbf{M} \mathbf{Ex}_b^a$ -*identifies* f (written: $f \in \mathbf{Ex}_b^a(\mathbf{M})$) $\stackrel{\text{def}}{\iff} [(\exists i \mid \varphi_i =^a f) (\forall^\infty n)[\mathbf{M}(f[n]) = i] \wedge \text{card}(\{n \mid ? \neq \mathbf{M}(f[n]) \neq \mathbf{M}(f[n+1])\}) \leq b]$.
- (b) $\mathbf{Ex}_b^a = \{\mathcal{S} \mid (\exists \mathbf{M})[\mathcal{S} \subseteq \mathbf{Ex}_b^a(\mathbf{M})]\}$.

\mathbf{Ex}_*^0 is written sometimes as \mathbf{Ex} and \mathbf{Ex}_*^a is sometimes written as \mathbf{Ex}^a . Theorem 1 below gives some of the results about \mathbf{Ex}_b^a criteria.

Theorem 1 ([CS83]) For all m, n ,

- (a) $\mathbf{Ex}_0^{n+1} - \mathbf{Ex}^n \neq \emptyset$;
- (b) $\mathbf{Ex}_{m+1}^0 - \mathbf{Ex}_m^* \neq \emptyset$;
- (c) $\mathbf{Ex}_0^* - \bigcup_{n \in N} \mathbf{Ex}^n \neq \emptyset$;
- (d) $\mathbf{Ex}_*^0 - \bigcup_{m \in N} \mathbf{Ex}_m^* \neq \emptyset$.

Blum and Blum [BB75] first showed that $\mathbf{Ex} \subset \mathbf{Ex}^*$.

4 Nearly-Minimal Identification

Freivalds considered the learning of minimal-size programs and showed that such learning is dependent on the acceptable programming system from which programs for functions are learned. He, however, considered a variant of such learning, where the conditions of parsimony on the size of the final programs are relaxed. The $a = 0, b = *$ case of the definition immediately below is essentially the way he relaxed the constraints on parsimony. The criteria of this definition are acceptable programming system independent.

Definition 2 ([Fre75, Che82])

- (a) A learning machine \mathbf{M} \mathbf{Mex}_b^a -*identifies* \mathcal{S} (written: $\mathcal{S} \subseteq \mathbf{Mex}_b^a(\mathbf{M})$) $\stackrel{\text{def}}{\iff}$ there is a computable function g such that, for all $f \in \mathcal{S}$, $\mathbf{M} \mathbf{Ex}_b^a$ -*identifies* f and $\mathbf{M}(f) \leq g(\text{MinProg}(f))$.
- (b) $\mathbf{Mex}_b^a = \{\mathcal{S} \mid (\exists \mathbf{M})[\mathcal{S} \subseteq \mathbf{Mex}_b^a(\mathbf{M})]\}$.

In the definition above, the g represents a fudge factor by which the parsimony constraint is loosened. The final programs are, in a sense, *nearly-minimal-size*.

Theorem 2 below gives some of the results about the \mathbf{Mex}_b^a criteria.

Theorem 2 ([Che82]) For all m, n, a, b ,

- (a) $\mathbf{Mex}_b^a \subseteq \mathbf{Ex}_b^a$;
- (b) $\mathbf{Ex} - \mathbf{Mex}^n \neq \emptyset$;
- (c) $\mathbf{Mex}_0^{n+1} - \mathbf{Ex}_m^n \neq \emptyset$;
- (d) $\mathbf{Mex}_{m+1}^0 - \mathbf{Ex}_m^* \neq \emptyset$;
- (e) $\mathbf{Mex}^* = \mathbf{Ex}^*$;
- (f) $\mathbf{Ex}_m^n \subset \mathbf{Mex}_*^n$.

Corollary 1 For all m, n, a, b ,

- (a) $\mathbf{Mex}^n \subset \mathbf{Ex}^n$;
- (b) $\mathbf{Mex}_m^a \subseteq \mathbf{Mex}_n^b \Leftrightarrow (a \leq b) \wedge (m \leq n)$.

Kinber [Fre75, Kin77a] announced that $\mathbf{Mex} \subset \mathbf{Ex}$. Jain has recently shown that $\mathbf{Mex}_0^{n+1} \not\subseteq \mathbf{Ex}^n$.

5 Not-So-Nearly-Minimal-Size Program Inference

Nearly minimal size program inference, as defined by \mathbf{Mex} , requires that the final program size be within a computable fudge factor of the actual minimum program. In the present paper, we wish to successively relax the computable fudge factor constraint and investigate whether the learning power is enhanced. The first means we choose to relax the constraint imposed by \mathbf{Mex} , is essentially to allow *lim_d-computable* fudge factors. *Lim_d-computability* is defined below (Definition 4).

Definition 3

$$\lim_{t \rightarrow \infty} h(x, t) \stackrel{\text{def}}{=} \begin{cases} y & \text{if } (\forall^\infty t)[h(x, t) = y]; \\ \uparrow & \text{otherwise.} \end{cases}$$

We write $h(x, \infty)$ for $\lim_{t \rightarrow \infty} h(x, t)$.

Definition 4 $g : N \rightarrow N$ is *lim_d-computable* $\stackrel{\text{def}}{\Leftrightarrow} (\exists \text{ computable } h : (N \times N) \rightarrow N)(\forall x)[g(x) = h(x, \infty)] \wedge (\forall x)[\text{card}(\{t \mid h(x, t) \neq h(x, t+1)\}) \leq d]$.

Intuitively, in Definition 4, $h(x, t)$ is the output at discrete time t of a mind changing algorithm for g (acting on input x). $(\forall x)[g(x) = h(x, \infty)]$, for h computable, means, then, that, for all x , for all but finitely many times t , the output of the mind changing algorithm on input x is $g(x)$. d is just a bound on how many times the mind changing algorithm for g is allowed to change its mind.

We write lim-computable for $\text{lim}_*\text{-computable}$. It is easy to show that there is a lim-computable function g such that $(\forall \text{ computable } f)(\forall^\infty x)[g(x) > f(x)]$. Hence, the lim-computable functions go way beyond the computable ones; in fact, they have been known since Post [Sha71] to characterize the functions computable with an oracle for the halting problem.

It turns out that, for $d \notin \{0, *\}$, the class of $\text{lim}_d\text{-computable}$ functions *fail* to have some useful closure properties one easily (and correctly) takes for granted in the $d \in \{0, *\}$ cases: it is easy to show that, for $d \notin \{0, *\}$, there is a $\text{lim}_1\text{-computable}$ function g so that for *no* $\text{lim}_d\text{-computable}$, *monotone non-decreasing* function g' do we have $g' \geq g$. Of course, intuitively, the fudge factors that make the most sense to use are *monotone non-decreasing* and many proofs ostensibly require them too. Hence, in our definition just below, we employ the device of considering *only* *monotone non-decreasing* fudge factors. This trick enables one, for example, to show the criteria so introduced are acceptable programming system independent.

Definition 5

- (a) A learning machine \mathbf{M} $\text{Lim}_d\text{Mex}_b^a$ -*identifies* \mathcal{S} (written: $\mathcal{S} \subseteq \text{Lim}_d\text{Mex}_b^a(\mathbf{M}) \stackrel{\text{def}}{\Leftrightarrow}$ there is a monotone non-decreasing $\text{lim}_d\text{-computable}$ function g such that, for all $f \in \mathcal{S}$, $\mathbf{M} \text{Ex}_b^a$ -*identifies* f and $\mathbf{M}(f) \leq g(\text{MinProg}(f))$).
- (b) $\text{Lim}_d\text{Mex}_b^a = \{\mathcal{S} \mid (\exists \mathbf{M})[\mathcal{S} \subseteq \text{Lim}_d\text{Mex}_b^a(\mathbf{M})]\}$.

Hence, our definition requires that the machines converge to a program that is *not-so-nearly-minimal-size*. We mostly write LimMex_b^a instead of $\text{Lim}_*\text{Mex}_b^a$. The following proposition is obvious.

Proposition 1 For all m, a, b ,

- (a) $\text{Mex}_b^a \subseteq \text{LimMex}_b^a \subseteq \text{Ex}_b^a$;
- (b) $\text{Lim}_m\text{Mex}_b^a \subseteq \text{Lim}_{m+1}\text{Mex}_b^a$;
- (c) $\text{Lim}_0\text{Mex}_b^a = \text{Mex}_b^a$.

The following theorem is essentially proved in Chen [Che82] using the class of functions of finite support (i.e., functions that have value 0 on all but finitely many inputs).

Theorem 3 ([Che82]) For all n , $\text{Ex} - \text{LimMex}^n \neq \emptyset$.

Theorem 4 below shows that relaxing the parsimony constraint on final programs from being within a *computable* fudge factor of minimal size to being within a *lim-computable* fudge factor of minimal size *does* result in an increase in learning power. However, by Theorem 3 above, the requirement that final programs be within a lim-computable fudge factor of minimal size nonetheless retains *some* parsimony in the final programs.

Theorem 4 For all n , $\text{LimMex} - \text{Mex}^n \neq \emptyset$.

Theorem 4 turns out to be a consequence of a result we prove later (Theorem 14 in Section 6), and, hence, we do not prove Theorem 4 here. Theorem 4 originally encouraged us to explore whether there was a fine hierarchy between Mex and LimMex based on $\text{lim}_d\text{-computable}$ fudge factors. We consider this next.

Lemma 1 For each $n > 0$, every monotone non-decreasing $\text{lim}_n\text{-computable}$ function is dominated by a monotone non-decreasing *computable* function.

PROOF.

We do the $n = 1$ case only. The other cases are, then, a straightforward lift.

Suppose g is a monotone non-decreasing lim_1 -computable function as witnessed by computable h .

Case 1: $(\forall^\infty x)(\forall t)[h(x, t) = h(x, 0)]$.

Clearly in this case, there exists a g' , computable and monotone non-decreasing, such that $g' \geq g$.

Case 2: $(\exists^\infty x)(\exists t)[h(x, t) \neq h(x, 0)]$.

In this case, we define g' as follows. $g'(0) = g(0)$. For each $x > 0$, $g'(x)$ is defined as follows. Search for a $y \geq x$ and a $t > 0$ such that $h(y, 0) \neq h(y, t)$ and $h(y, t) \geq g'(x - 1)$; set $g'(x) = h(y, t)$, for the y and t so found. Clearly, g' is computable, monotone non-decreasing, and dominates g everywhere. \blacksquare

Clearly by Lemma 1, we have the following

Theorem 5 For each $n \in \mathbb{N}$, $\mathbf{Lim}_n \mathbf{Mex}_b^a = \mathbf{Mex}_b^a$.

We had originally hoped the immediately previous theorem was not true, that there was a fine hierarchy between \mathbf{Mex} and \mathbf{LimMex} based on lim_d -computable fudge factors. In the next section we successfully explore some different sources of restricted parsimony fine structure.

6 Further Generalizations

In Section 6.1 we briefly explore the effect of allowing even looser fudge factors and indicate many presently open questions.

We saw in Theorem 5 above that *natural number* bounds on convergence of limiting procedures for computing fudge factors do *not* provide a hierarchy of criteria between \mathbf{Mex} and \mathbf{LimMex} . In Section 6.2 we consider *constructive ordinal* [Rog67] bounds on such limiting procedures instead. This approach was nicely inspired by [FS91]. We present in Section 6.2 some interesting results providing a fine structure between \mathbf{Mex} and \mathbf{LimMex} . We also indicate many questions open as of the writing of this preliminary report.

6.1 Looser Fudge Factors

One more way that we can examine how programs inferred can be allowed to be not-so-nearly-minimal-size is by allowing even looser fudge factors. To this end, consider the following definitions.

Definition 6 For $h : N^{n+1} \rightarrow N, x \in N, i < n, h(x, t_1, t_2, \dots, t_i, \infty, \dots, \infty) = \lim_{t_{i+1} \rightarrow \infty} h(x, t_1, t_2, \dots, t_i, t_{i+1}, \infty, \dots, \infty)$.

Definition 7 $f : N \rightarrow N$ is $\text{lim}_{d_1, d_2, \dots, d_n}^n$ -computable $\stackrel{\text{def}}{\iff} (\exists \text{ computable } h : N^{n+1} \rightarrow N)(\forall x)[f(x) = h(x, \infty, \dots, \infty)]$ and $(\forall i \mid 1 \leq i \leq n)(\forall x, t_1, t_2, \dots, t_i)[\text{card}(\{t \mid h(x, t_1, \dots, t_{i-1}, t, \infty, \dots, \infty) \neq h(x, t_1, \dots, t_{i-1}, t + 1, \infty, \dots, \infty)\}) \leq d_i]$.

We write *limⁿ-computable* for $\lim_{*,\dots,*}^n$ -computable. The above definitions could be also be generalized to finite but unbounded (i.e., $*$) iterations of limits. In the definition below, we use $\lim_{d_1,d_2,\dots,d_n}^n$ -computable functions to measure allowed deviance of programs from being nearly-minimal-size.

Definition 8

- (a) Suppose \mathbf{M} is a learning machine. $\mathbf{M} \mathbf{Lim}_{d_1,\dots,d_n}^n \mathbf{Mex}_b^a$ -identifies \mathcal{S} (written: $\mathcal{S} \subseteq \mathbf{Lim}_{d_1,\dots,d_n}^n \mathbf{Mex}_b^a(\mathbf{M})$) $\stackrel{\text{def}}{\iff}$ there is a \lim_{d_1,\dots,d_n}^n -computable monotone non-decreasing function g such that, for all $f \in \mathcal{S}$, $\mathbf{M} \mathbf{Ex}_b^a$ -identifies f and $\mathbf{M}(f) \leq g(\text{MinProg}(f))$.
- (b) $\mathbf{Lim}_{d_1,\dots,d_n}^n \mathbf{Mex}_b^a = \{\mathcal{S} \mid (\exists \mathbf{M})[\mathcal{S} \subseteq \mathbf{Lim}_{d_1,\dots,d_n}^n \mathbf{Mex}_b^a(\mathbf{M})]\}$.

We mostly write $\mathbf{Lim}^n \mathbf{Mex}_b^a$ instead of $\mathbf{Lim}_{*,\dots,*}^n \mathbf{Mex}_b^a$. The following proposition implies that, for fudge factors computed by two levels of unrestricted iterated limits, there is essentially no longer any parsimony retained in the resultant final programs.

Proposition 2 $\mathbf{Lim}^2 \mathbf{Mex}^a = \mathbf{Ex}^a$.

PROOF. (\subseteq) Trivial.

(\supseteq) Suppose $\mathcal{S} \subseteq \mathbf{Ex}^a(\mathbf{M})$. Then, $\mathcal{S} \in \mathbf{Lim}^2 \mathbf{Mex}^a$ as witnessed by \mathbf{M} and \lim^2 -computable monotone non-decreasing g , such that, for all x , $g(x) \stackrel{\text{def}}{=} h(x, \infty, \infty)$, where h is defined below. We first define h' as below. It is to be understood, that for values of h' , ?'s are changed to 0's.

$$h'(i, t_1, t_2) = \begin{cases} \mathbf{M}(\varphi_i[t_1]) & \text{if } (\forall x < t_1)[\Phi_i(x) \leq t_2] \wedge \\ & (\forall y \mid t_1 \leq y \leq t_2 \wedge (\forall x < y)[\Phi_i(x) \leq t_2]) \\ & [\mathbf{M}(\varphi_i[t_1]) = \mathbf{M}(\varphi_i[y])]; \\ 0 & \text{otherwise.} \end{cases}$$

We then define h as

$$h(i, t_1, t_2) = \max(\{h'(x, t_1, t_2) \mid x \leq i\}).$$

Clearly, h is monotone non-decreasing and can be seen to dominate h' . ■

Corollary 2 $\mathbf{Mex}^n \subset \mathbf{Lim} \mathbf{Mex}^n \subset \mathbf{Lim}^2 \mathbf{Mex}^n = \mathbf{Lim}^3 \mathbf{Mex}^n = \dots = \mathbf{Ex}^n$.

There are many mostly uninvestigated questions still open. What happens with the iterated limits when we consider criteria that don't necessarily require convergence to a single final program, such as \mathbf{Bc}^a [Bar74, CS83]? Which $\mathbf{Lim}_d \mathbf{Mex}_b^a$ criteria have "limiting-standardizability" style characterizations similar to those first obtained for \mathbf{Mex} in [Fre75]. Generally, except for the cases noted above and their trivial consequences, how do the learning classes $\mathbf{Lim}_{d_1,\dots,d_n}^n \mathbf{Mex}_b^a$ compare to one another?

6.2 Constructive Ordinal Bounds on Limits

We proceed very informally. Some familiarity with a treatment of constructive ordinals such as the ones in [Rog67, Sac90] may be useful to readers of this section. Readers may also find [FS91] useful in this regard.

Intuitively ordinals [Sie65] are representations of well-orderings. 0 represents the empty ordering, 1 represents the ordering of 0 by itself, 2 the ordering $0 < 1$, 3 the ordering $0 < 1 < 2, \dots$. The ordinal ω represents the standard ordering of all of N . $\omega + 1$ represents

the ordering of N consisting of the positive integers in standard order *followed by* 0. $\omega + \omega$ represents the ordering of N consisting of the even numbers in standard order followed by the odd numbers in standard order. The *constructive ordinals* are just those that have a program (called a *notation*) in some system which specifies how to build them (lay them out end to end so to speak). We will informally employ, as our system of notation, the variant of Kleene's system \mathbf{O} presented in [Rog67]. In this system, 2^0 is (by definition) the notation for 0. *Successor* ordinals are those with an immediate predecessor; for example, $1, 2, 3, \omega + 1, \dots$ are successor ordinals with respective immediate predecessors $0, 1, 2, \omega, \dots$. If u is a notation for the immediate predecessor of a successor ordinal, then a notation for that successor ordinal is (by definition) 2^u . All other ordinals are *limit* ordinals; for example, $\omega, \omega + \omega, \dots$ are limit ordinals. Kleene [Kle38, Rog67, Sac90] defined a natural partial ordering of notations, $<_o$, so that two notations so ordered represent respective ordinals with the second larger than the first. We omit details. Suppose $\varphi_p(0), \varphi_p(1), \varphi_p(2), \dots$ are each notations in $<_o$ order. Suppose that the corresponding ordinals are longer and longer initial segments of some limit ordinal which is their sup. For example, some such p generates the respective notations for $0, 1, 2, \dots$ in $<_o$ order, and ω is the sup of this sequence. In general, then, p essentially describes how to build the limit ordinal which is the sup of the ordinals with notations $\varphi_p(0), \varphi_p(1), \varphi_p(2), \dots$. A notation for this limit ordinal is (by definition) $3 \cdot 5^p$. Clearly such limit ordinals have infinitely many such notations, different ones for different generating p 's. Nothing else is a notation. As in the literature on constructive ordinals, we use ' $x \leq_o y$ ' for ' $x <_o y \vee x = y$ ', ' $x \geq_o y$ ' to mean ' $y \leq_o x$ ' and ' $x >_o y$ ' to mean ' $y <_o x$ '. We also recall the function $|\cdot|_o : \mathbf{O} \rightarrow$ the set of ordinals, defined as follows [Kle55, Rog67, Sac90]

$$\begin{aligned} |1|_o &= 0 \\ |2^u|_o &= |u|_o + 1 \\ |3 \cdot 5^p|_o &= \lim_{n \rightarrow \infty} |\varphi_p(n)|_o \end{aligned}$$

The following properties of $<_o$ will be useful to recall [Kle55, Rog67, Sac90].

Fact 1 For all $x, y \in N$,

- (a) $x <_o y \Rightarrow (x \in \mathbf{O} \wedge y \in \mathbf{O})$.
- (b) $x \in \mathbf{O} \Rightarrow 1 \leq_o x$.
- (c) $x <_o y \Rightarrow y \neq 1$.
- (d) $x <_o 2^y \Rightarrow x \leq_o y$.
- (e) $x <_o 3 \cdot 5^p \Rightarrow (\exists n)[x <_o \varphi_p(n)]$.
- (f) $(x \leq_o z \wedge y \leq_o z) \Rightarrow (x <_o y \vee x = y \vee x >_o y)$.

The following fact on notations [Rog67, Sac90] is also important to us.

Fact 2 There exist computable functions h_1 and h_2 such that, for all $v \in \mathbf{O}$,

- (a) $W_{h_1(v)} = \{u \mid u <_o v\} \in \mathbf{O}$;
- (b) $W_{h_2(v)} = \{\langle u_1, u_2 \rangle \mid u_1 <_o u_2 <_o v\}$ is a well-ordering isomorphic to $|v|_o$.

Everyone knows how to use natural numbers as counters. [FS91] introduced the use of constructive ordinals as more general counters. In this subsection we use constructive ordinals to count the allowed mind changes of limiting procedures.

Convention 1 If $n \in N$, then \mathbf{n} is the unique notation of n in the \mathbf{O} notation system.

So, for example, $\mathbf{0} = 1$ and $|\mathbf{0}|_o = 0$; $\mathbf{2} = 2^{2^1} = 4$ and $|\mathbf{2}|_o = 2$.

In the definition of *lim_d-computable* (Definition 7) $d \in N$ played the role of a counter for allowed mind-changes in the limiting process. For each notation u in \mathbf{O} , we will define *lim_u-computable* (Definition 9), where, intuitively, u serves as a *transfinite* counter of allowed mind changes in the limiting procedure. This definition will conflict slightly with Definition 7, and, hence, after Definition 9, we no longer use Definition 7. As we will see, though, for $d \in N$, *lim_d-computable* from Definition 9 corresponds to *lim_d-computable* from Definition 7.

Intuitively, h in Definition 9 just below plays a similar role to h in Definition 7, and the function *tfcounter* in Definition 9 serves as a transfinite counter. As before, t can be thought of a discrete time parameter. Further explanation is given just after Definition 9.

Definition 9 Suppose $u \in \mathbf{O}$. $g : N \rightarrow N$ is *lim_u-computable* $\stackrel{\text{def}}{\Leftrightarrow}$ there exist computable functions $h : (N \times N) \rightarrow N$ and *tfcounter* : $(N \times N) \rightarrow N$ such that

- (a) $(\forall y)[g(y) = h(y, \infty)]$,
- (b) $(\forall y, t)[\text{tfcounter}(y, t) \in \mathbf{O}]$,
- (c) $(\forall y)[\text{tfcounter}(y, 0) = u]$,
- (d) $(\forall y, t)[\text{tfcounter}(y, t + 1) \leq_o \text{tfcounter}(y, t)]$, and
- (e) $(\forall y, t)[h(y, t + 1) \neq h(y, t) \Rightarrow \text{tfcounter}(y, t + 1) <_o \text{tfcounter}(y, t)]$.

Part (b) of Definition 9 restricts the transfinite counter values to be notations $\in \mathbf{O}$. Part (c) of Definition 9 initializes the transfinite counter at u . By Fact 2, the notations $<_o u$ are well-ordered; hence, part (d) of Definition 9 implies the counter cannot not descend infinitely. Part (e) of Definition 9 guarantees that, when h has a mind change, then the transfinite counter *must* decrement. This part does *not* restrict how much it decrements. It also allows the transfinite counter to decrement without an accompanying mind change in h . These latter two properties are combinatorially convenient. Note that parts (b) through (e) imply that $h(y, \infty)$ is defined and part (a) defines $g(y)$ to be the value $h(y, \infty)$.

For $u \in \mathbf{O}$, we now (partly re-)define the learning criterion $\mathbf{Lim}_u \mathbf{Mex}_b^a$ to be just like $\mathbf{Lim} \mathbf{Mex}_b^a$ except that the fudge factor g is *lim_u-computable* as defined in Definition 9. $\mathbf{Lim}_* \mathbf{Mex}$ retains its original meaning. Clearly, for $d \in N$, $\mathbf{Lim}_d \mathbf{Mex}_b^a$ from this definition corresponds to $\mathbf{Lim}_d \mathbf{Mex}_b^a$ from Definition 5. It is easy to show $\mathbf{Lim}_u \mathbf{Mex}_b^a$ is acceptable programming system independent.

We are interested in comparing, for various $u \in \mathbf{O}$, the classes $\mathbf{Lim}_u \mathbf{Mex}_b^a$. We should note that, unfortunately it is open and mostly uninvestigated¹ whether, for all $u, u' \in \mathbf{O}$ such that $|u|_o = |u'|_o$, $\mathbf{Lim}_u \mathbf{Mex}_b^a = \mathbf{Lim}_{u'} \mathbf{Mex}_b^a$. Moreover, we do not know whether, for all $u, u' \in \mathbf{O}$ such that $|u|_o = |u'|_o$, the class of *lim_u-computable* functions = the class of *lim_{u'}-computable* functions. We have also not investigated the possible connections between the *lim_u-computable* characteristic functions and the hierarchies of [Ers68]. We have further not considered the

¹We know a few special cases.

possible dependencies of the lim_u -computable functions or of the classes $\mathbf{Lim}_u\mathbf{Mex}_b^a$ on the choice of notation *system* [Kle38, Rog67].

Before we proceed to compare, for various $u \in \mathbf{O}$, the classes $\mathbf{Lim}_u\mathbf{Mex}_b^a$, we state the following six theorems.

Theorem 6 ([CK37, Kle55, Rog67, Sac90]) There exists a computable function $+_o$ such that, for all $x, y \in N$,

$$x +_o y = \begin{cases} x & \text{if } y = 1; \\ 2^{(x+_om)} & \text{if } y = 2^m, m > 1; \\ 3 \cdot 5^q & \text{if } y = 3 \cdot 5^p, (\text{where } (\forall n)[\varphi_q(n) = x +_o \varphi_p(n)]); \\ 7 & \text{otherwise .} \end{cases}$$

Furthermore, q is a computable, 1–1 function of x and p .

The 1–1-ness of $+_o$ just mentioned is crucial for proving parts of Theorem 7, which in turn are necessary for proving many result that follow.

$+_o$ has the following useful properties.

Theorem 7 ([CK37, Kle55, Sac90]) For all x, y and $z \in N$,

- (a) $x, y \in \mathbf{O} \Leftrightarrow x +_o y \in \mathbf{O}$.
- (b) $x, y \in \mathbf{O} \Rightarrow |x +_o y|_o = |x|_o + |y|_o$.
- (c) $(x, y \in \mathbf{O} \wedge y \neq 1) \Rightarrow x <_o x +_o y$.
- (d) $(x \in \mathbf{O} \wedge z <_o y) \Leftrightarrow (x +_o z) <_o (x +_o y)$.
- (e) $(x \in \mathbf{O} \wedge y = z \in \mathbf{O}) \Leftrightarrow (x +_o y) = (x +_o z)$.
- (f) $x \leq_o z <_o (x +_o y) \Rightarrow (\exists!z)[y' <_o y \wedge (x +_o y') = z]$.

Note that $+_o$ (on \mathbf{O}) is non-commutative like $+$ for ordinals; $+_o$ is, however, also non-associative (on \mathbf{O}) unlike $+$ for ordinals. We adopt the convention that $x +_o y +_o z$ means $(x +_o y) +_o z$. The non-associativity of $+_o$ leads to some subtleties that are otherwise absent when dealing with ordinals, as opposed to notations.

The next theorem is a slight modification of a theorem in [CK37]. The $y = 2$ case is treated differently from therein and ensures that all the parts of Theorem 9 hold.

Theorem 8 ([CK37]) There is a computable function \times_o such that, for all $x, y \in N$,

$$x \times_o y = \begin{cases} 1 & \text{if } y = 1; \\ x & \text{if } y = 2; \\ (x \times_o m) +_o x & \text{if } y = 2^m, m > 1; \\ 3 \cdot 5^q & \text{if } y = 3 \cdot 5^p, (\text{where } (\forall n)[\varphi_q(n) = x \times_o \varphi_p(n)]); \\ 7 & \text{otherwise.} \end{cases}$$

Furthermore, q is a computable, 1–1 function of x and p .

Like $+_o$, \times_o is neither commutative nor associative (on \mathbf{O}), and, as for $+_o$, in unparenthesized expressions involving \times_o , we associate to the left.

Analogous to Theorem 7, we have the following theorem for \times_o .

Theorem 9 For all x, y and $z \in N$.

- (a) $(y \neq 1) \Rightarrow [x, y \in \mathbf{O} \Leftrightarrow x \times_o y \in \mathbf{O}]$.
- (b) $x, y \in \mathbf{O} \Rightarrow |x \times_o y|_o = |x|_o \times |y|_o$.
- (c) $(x \neq 1 \wedge y \neq 1 \wedge y \neq 2) \Rightarrow [x, y \in \mathbf{O} \Leftrightarrow x <_o x \times_o y]$.
- (d) $(x \neq 1) \Rightarrow [(x \in \mathbf{O} \wedge z <_o y) \Leftrightarrow x \times_o z <_o x \times_o y]$.
- (e) $(x \neq 1) \Rightarrow [(x \in \mathbf{O} \wedge y = z \in \mathbf{O}) \Leftrightarrow x \times_o y = x \times_o z]$.
- (f) $x' <_o x \Leftrightarrow (\forall y)[x \times_o y +_o x' <_o x \times_o (y +_o \mathbf{1})]$.
- (g) $x \leq_o z <_o (x \times_o y) \Rightarrow (\exists! y' <_o y, x' <_o x)[z = (x \times_o y') +_o x']$.

Another theorem from [CK37] is the following, except for a slight modification for the $y = 2$ case as in Theorem 9.

Theorem 10 ([CK37]) There is a computable function \exp_o such that, for all $x, y \in N$,

$$x \exp_o y = \begin{cases} 2 & \text{if } y = 1; \\ x & \text{if } y = 2; \\ (x \exp_o m) \times_o x & \text{if } y = 2^m, m > 1; \\ 3 \cdot 5^q & \text{if } y = 3 \cdot 5^p, (\text{where } (\forall n)[\varphi_q(n) = x \exp_o \varphi_p(n)]); \\ 7 & \text{otherwise.} \end{cases}$$

Furthermore, q is a computable, 1–1 function of x and p .

Again, similar comments about commutativity and associativity hold for \exp_o as for \times_o and $+_o$. The following are some of the properties of \exp_o defined immediately above.

Theorem 11 For all x, y and $z \in N$.

- (a) $(y \neq 1) \Rightarrow [x, y \in \mathbf{O} \Leftrightarrow x \exp_o y \in \mathbf{O}]$.
- (b) $x, y \in \mathbf{O} \Rightarrow |x \exp_o y|_o = |x|_o \exp_o |y|_o$.
- (c) $(x \neq 1 \wedge x \neq 2 \wedge y \neq 1 \wedge y \neq 2) \Rightarrow [x, y \in \mathbf{O} \Leftrightarrow x <_o x \exp_o y]$.
- (d) $(x \neq 1 \wedge x \neq 2) \Rightarrow [(x \in \mathbf{O} \wedge z <_o y) \Leftrightarrow x \exp_o z <_o x \exp_o y]$.
- (e) $(x \neq 1 \wedge x \neq 2) \Rightarrow [(x \in \mathbf{O} \wedge y = z \in \mathbf{O}) \Leftrightarrow x \exp_o y = x \exp_o z]$.
- (f) $(x \neq 1) \Rightarrow [x' <_o x \Leftrightarrow (\forall y)[(x \exp_o y) \times_o x' <_o x \exp_o (y +_o \mathbf{1})]]$.
- (g) $x \leq_o z <_o (x \exp_o y) \Rightarrow (\exists! y' <_o y, x' <_o x, z' <_o (x \exp_o y'))[z = ((x \exp_o y') \times_o x') +_o z']$.

We recall that $\omega = \lim_{n \rightarrow \infty} |\mathbf{n}|_o$.

Convention 2 w , with or without *subscripts*, ranges over notations for ω .

Similar to Proposition 1, we have

Proposition 3 For all $u, v \in \mathbf{O}$ such that $u \leq_o v$ and $a, b \in N \cup \{*\}$,

- (a) $\mathbf{Lim}_u \mathbf{Mex}_b^a \subseteq \mathbf{Lim}_v \mathbf{Mex}_b^a$,

(b) $\mathbf{Lim}_0 \mathbf{Mex}_b^a = \mathbf{Mex}_b^a$.

Again, similar to Lemma 1, we have

Lemma 2 For all $u, v, w \in \mathbf{O}$ such that $v <_o u \times_o w$, every monotone, non-decreasing \lim_v -computable function is dominated by a monotone, non-decreasing \lim_u -computable function.

PROOF. Let $u, y, w \in \mathbf{O}$ be such that $y <_o u \times_o w$. Using Fact 1, there exists k such that $y <_o u \times_o \mathbf{k}$. So it suffices to prove the lemma for $y <_o u \times_o \mathbf{k}$, for each k . The proof proceeds similarly to that of Lemma 1.

We will do only the $k = 2$ case. The other cases can be proved on similar lines.

Suppose g is a monotone, non-decreasing $\lim_{u \times_o 2}$ -computable function as witnessed by h and $tfcounter$.

Case 1: $(\forall^\infty y)[tfcounter(y, \infty) >_o u]$.

Let $C = \{y \mid tfcounter(y, \infty) \leq_o u\}$. Since C is finite, the set C' defined as $C' = \{\langle y, n \rangle \mid y \in C \wedge h(y, \infty) = n\}$ is also finite and hence recursive.

For all $y \notin C$, $u +_o u \geq_o tfcounter(y, \infty) >_o u$. So, by Theorem 9, part (g), for all $v \notin C$, t , there exists a unique u_t such that $\mathbf{0} <_o u_t \leq_o u$ and $u +_o u_t = tfcounter(v, t)$. We note that these u_t 's can be found effectively. Define computable functions h' and $tfcounter'$ thus.

$$h'(v, t) = \begin{cases} n & \text{if } \langle v, n \rangle \in C'; \\ h(y, t) & \text{if } y \notin C. \end{cases}$$

$$tfcounter'(y, t) = \begin{cases} u & \text{if } y \in C; \\ u_t & \text{if } y \notin C \wedge tfcounter(y, t) = u +_o u_t. \end{cases}$$

Define $g'(y) = h'(y, \infty)$. Then, clearly, g' is a monotone non-decreasing \lim_u -computable function as witnessed by h' and $tfcounter'$ and g' dominates g .

Case 2: Not *Case 1*.

So, $(\exists^\infty y)[tfcounter(y, \infty) \leq_o u]$. Let $C = \{y \mid tfcounter(y, \infty) \leq_o u\}$. Note that C is an infinite, recursively enumerable set. Hence, there exists $C' \subseteq C$, such that C' is recursive. Let $C' = \{y_0 < y_1 < \dots\}$ be such an infinite recursively enumerable subset of C . Define computable functions h' and $tfcounter'$ thus.

$$h'(z, t) = \begin{cases} 0 & \text{if } tfcounter(y_z, t) >_o u; \\ h(y_z, t) & \text{otherwise .} \end{cases}$$

$$tfcounter'(z, t) = \begin{cases} u & \text{if } tfcounter(y_z, t) >_o u; \\ tfcounter(y_z, t) & \text{otherwise .} \end{cases}$$

Define $g'(y) = h'(y, \infty)$. Then, clearly, g' is a monotone non-decreasing \lim_u -computable function as witnessed by h' and $tfcounter'$ and g' dominates g . ■

The above lemma immediately gives us

Theorem 12 For all a, b , for all $u, v, w \in \mathbf{O}$ such that $v <_o u \times_o w$, $\mathbf{Lim}_v \mathbf{Mex}_b^a = \mathbf{Lim}_u \mathbf{Mex}_b^a$.

The immediately above theorem shows that no gain in learning power results by using \lim_v -computable fudge factors instead of \lim_u -computable fudge factors, if u and v are related as above (i.e., there exists $w \in \mathbf{O}$ such that $v <_o u \times_o w$). Our next theorem, Theorem 13, shows that, for suitable $u \in \mathbf{O}$, when $|u|_o < \omega^\omega$, learning power strictly increases if we use $\lim_{u \times_o w}$ -computable fudge factors instead of \lim_u -computable fudge factors. It is useful first to have the immediately following proposition, our inspiration for which came from Cantor's Normal Form Theorem for ordinals [Sie65, Page 323].

We recall, by Convention 2, that w_1, w_2, \dots (as well as w) are all notations for ω .

Proposition 4 For all m , for all w_1, w_2, \dots, w_{m+1} , for all x , $x <_o (w_1 \times_o w_2 \times_o \dots \times_o w_{m+1}) \Leftrightarrow (\exists! n_1, n_2, \dots, n_{m+1}) [x = (w_1 \times_o w_2 \times_o \dots \times_o w_m \times_o \mathbf{n}_1) +_o (w_1 \times_o w_2 \times_o \dots \times_o w_{m-1} \times_o \mathbf{n}_2) +_o \dots +_o (w_1 \times_o \mathbf{n}_m) +_o \mathbf{n}_{m+1}]$.

Furthermore, for the left to right direction, the values for n_1, n_2, \dots, n_{m+1} can be algorithmically found.

The above proposition can be proved by induction on m .

Convention 3 For every $u \in O, n \in N$, $u^n = u \exp_o \mathbf{n}$.

Theorem 13 For all $w_1, w_2, \dots, w_{m+1}, m, n$,

$\mathbf{Lim}_{w_1 \times_o w_2 \times_o \dots \times_o w_{m+1}} \mathbf{Mex} - \mathbf{Lim}_{w_1 \times_o w_2 \times_o \dots \times_o w_m} \mathbf{Mex}^n \neq \emptyset$.

PROOF.

For typographical convenience, we will prove the theorem for the case when $w_1 = w_2 = \dots = w_{m+1} = w$. The other cases easily follow. Also, we do only the $n = 0$ case here. The $n \neq 0$ cases, then, can be proved easily by modifying steps 2.3 and 4 in the second half of this proof. We first introduce some definitions that will help us in turn to define classes to prove this theorem.

For all f, p, n , let

$$S(f, p, n) = \{f(\langle p, x \rangle) \mid f(\langle p, x \rangle) \neq 0 \wedge \text{card}(\{y \leq x \mid f(\langle p, y \rangle) \neq 0\}) \leq n\}.$$

Intuitively, $S(f, p, n)$ is the set of all non-zero values in the p^{th} cylinder of the computable function f , provided there are less than n such values; otherwise, it is the first n non-zero values in the p^{th} cylinder. Suppose without loss of generality that $\langle 0, 0 \rangle = 0$.

For all f and $k > 0$, let

$$\begin{aligned} L_1^f &= \{f(\langle 0, 0 \rangle)\} \\ L_{k+1}^f &= \bigcup_{\langle p, n \rangle \in L_k^f} S(f, p, n) \end{aligned}$$

Intuitively, if $L_1^f = \{\langle p, n \rangle\}$, then L_2^f is just $S(f, p, n)$. For $k > 0$, L_{k+1}^f is the union of sets of numbers, each of whose cardinality and content are determined by one or more elements of L_k^f . In the proof immediately below, we define and use a computable function f for which, for $k > 0$, L_{k+1}^f is a *disjoint* union of sets of numbers, each of whose cardinality and content are determined by a *distinct* element of L_k^f . Such an f helps in clarity of presentation, though it is not necessary for proving this theorem.

For all m , let $\mathcal{S}_{\omega^m} = \{f \mid \lim_{t \rightarrow \infty} f(\langle 1, t \rangle) \downarrow = p \wedge \varphi_p = f \wedge p \leq \max(L_m^f)\}$.

We will now construct an inductive inference machine \mathbf{M} and $\lim_{w^{m+1}}$ computable g such that $\mathcal{S}_{\omega^{m+1}} \in \mathbf{Lim}_{w^{m+1}} \mathbf{Mex}$ as witnessed by \mathbf{M} and g . (We note that the same class can be used for different w 's that are notations for ω .) For all n , define $\mathbf{M}(f[n])$ as follows. Let $i_n = \max(\{j \mid \langle 1, j \rangle < n\})$. Let $\mathbf{M}(f[n]) = f(\langle 1, i_n \rangle)$.

Firstly, we define total, computable functions φ_j^t for each j and t as follows.

$$\varphi_j^t(x) = \begin{cases} \varphi_j(x) & \text{if } (\forall y \leq x)[\Phi_j(y) \leq t]; \\ 0 & \text{otherwise.} \end{cases}$$

Next, for all j, t, p, n , let

$$T(\langle j, t \rangle, p, n) = S(\varphi_j^t, p, n).$$

For all $k > 0$, and i, t , let

$$\begin{aligned} P_1^{\langle \leq i, t \rangle} &= \{\varphi_j(\langle 0, 0 \rangle) \mid j \leq i \wedge \Phi_j(\langle 0, 0 \rangle) \leq t\} \\ P_{k+1}^{\langle \leq i, t \rangle} &= \bigcup_{j \leq i} \bigcup_{\langle p, n \rangle \in P_k^{\langle \leq i, t \rangle}} T(\langle j, t \rangle, p, n) \end{aligned}$$

So, for all $k > 0$, $\lim_{t \rightarrow \infty} P_k^{\langle \leq i, t \rangle} \supseteq \bigcup_{j \leq i \wedge \varphi_j \in \mathcal{R}} L_k^{\varphi_j}$.

Also, let

$$\begin{aligned} \text{card}_k^{\langle \leq i, t \rangle} &= \text{card}(P_k^{\langle \leq i, t \rangle}). \\ \text{sum}_k^{\langle \leq i, t \rangle} &= \sum_{\langle p, n \rangle \in P_k^{\langle \leq i, t \rangle}} n. \end{aligned}$$

We now define $h, \text{tfcounter}$ as follows. For all i , let $h(i, 0) = 0$; $\text{tfcounter}(i, 0) = w^{m+1}$;

For $t > 0$, we define $h(i, t)$ and $\text{tfcounter}(i, t)$ as follows.

$$h(i, t) = \max(P_{m+1}^{\langle \leq i, t \rangle});$$

Let $n_1 = i + 1 - \text{card}_1^{\langle \leq i, t \rangle}$. For $1 \leq k \leq m$ let $n_{k+1} = \text{sum}_k^{\langle \leq i, t \rangle} - \text{card}_{k+1}^{\langle \leq i, t \rangle}$.
 $\text{tfcounter}(i, t) = w^m \times_o \mathbf{n}_1 +_o w^{m-1} \times_o \mathbf{n}_2 +_o \dots +_o w \times_o \mathbf{n}_m +_o \mathbf{n}_{m+1}$.

Finally, we define g as follows. For all y , $g(y) = h(y, \infty)$.

It can be verified that g is $\lim_{w^{m+1}}$ computable, as witnessed by h and tfcounter , and that for every $\varphi_i \in \mathcal{S}_{w^{m+1}}$, $\mathbf{M}(\varphi_i) \leq g(\text{MinProg}(\varphi_i))$.

We next show that $\mathcal{S}_{w^{m+1}} \notin \mathbf{Lim}_{w^m} \mathbf{Mex}$. We will show this only for $m > 0$ case. $m = 0$ case can be proved in a similar but much simpler manner.

Let \wp_1, \wp_2, \dots be the increasing sequence of prime numbers.

For all notations v of the form $w^{m-1} \times_o \mathbf{n}_1 +_o w^{m-2} \times_o \mathbf{n}_2 +_o \dots +_o w \times_o \mathbf{n}_{m-1} +_o \mathbf{n}_m$, for all k , such that $1 \leq k \leq m$, let

$$\text{num}_k^v = n_k.$$

For $k \leq m$, let

$$\text{prod}_k^v = 1 + \prod_{i=1}^k \wp_i^{1+n_i}.$$

We note that, for all v , $\text{prod}_0^v = 2$. Suppose $v = w^{m-1} \times_o \mathbf{n}_1 +_o w^{m-2} \times_o \mathbf{n}_2 +_o \dots +_o w \times_o \mathbf{n}_{m-1} +_o \mathbf{n}_m$ and $v' = w^{m-1} \times_o \mathbf{n}'_1 +_o w^{m-2} \times_o \mathbf{n}'_2 +_o \dots +_o w \times_o \mathbf{n}'_{m-1} +_o \mathbf{n}'_m$. Then, $\text{prod}_k^v = \text{prod}_k^{v'}$ if and only if, for $1 \leq i \leq k$, $n_i = n'_i$.

Suppose by way of contradiction that $\mathcal{S}_{w^{m+1}} \in \mathbf{Lim}_{w^m} \mathbf{Mex}$ as witnessed by \mathbf{M} and \lim_{w^m} -computable g . Let $h, \text{tfcounter}$ witness that g is \lim_{w^m} -computable. For u such that $|u|_o \neq 0$, any \lim_u -computable function g is dominated by some \lim_u -computable function g' such that g' makes at least one mind change on every input; this can be proved on the lines of Lemma 1.

By the Operator Recursion Theorem, there exists a recursive 1-1, increasing e such that, for all x , the functions $\varphi_{e(x)}$ may be defined as follows.

Let t be the least number such that $h(e(0), t) \neq h(e(0), 0)$.

Let $\text{curbnd} = h(e(0), t)$; $\text{curbot} = \text{current} = 1$; $\text{curtop} = \text{curbot} + \text{curbnd} + 1$.

Let $v = \text{tfcounter}(e(0), t)$;

Let $\varphi_{e(0)}(\langle 0, 0 \rangle) = \langle \text{prod}_0^v, 1 + \text{num}_1^v \rangle$.

for $k = 0$ **to** $m - 2$ **do**

$\varphi_{e(0)}(\langle \text{prod}_k^v, 0 \rangle) = \langle \text{prod}_{k+1}^v, 1 + \text{num}_{k+2}^v \rangle$;

endfor

Let $\varphi_{e(0)}(\langle \text{prod}_{m-1}^v, 0 \rangle) = e(\text{curtop})$;

For $x \leq \max(\{ \langle 0, 0 \rangle, \langle \text{prod}_0^v, 0 \rangle, \dots, \langle \text{prod}_{m-1}^v, 0 \rangle \})$, such that $\varphi_{e(0)}(x)$, has not been defined till now let $\varphi_{e(0)}(x) = 0$. Let x_s denote the least x such that $\varphi_{e(0)}(x)$ has not been defined before stage s .

Let $\text{Cancel} = \emptyset$.

Let $\text{lastmindch} = t$.

Go to stage 0.

Begin Stage s

1. For $x < x_s$, let $\varphi_{e(\text{current})}(x) = \varphi_{e(0)}(x)$.

2. Let $y = x_s$

repeat

2.1. If y is of the form $\langle 1, z \rangle$, then let

$\varphi_{e(\text{current})}(y) = e(\text{current})$.

Otherwise let $\varphi_{e(\text{current})}(y) = 0$.

2.2. If $h(e(0), \text{lastmindch}) \neq h(e(0), y + \text{lastmindch})$, then go to step 3.

2.3. If there exists a $i \leq \text{curbnd}$, $i \notin \text{Cancel}$, and $x_s < \langle 0, x \rangle < y$, such that $\varphi_i(\langle 0, x \rangle) \downarrow$ in $\leq y$ steps, then go to step 4.

2.4. If $\mathbf{M}(\varphi_{e(\text{current})}[y + 1]) > \text{curbnd}$, then go to step 5.

2.5. Let $y = y + 1$.

forever

3. Let $v = \text{tfcounter}(e(0), \text{lastmindch})$;

Let $\bar{v} = \text{tfcounter}(e(0), \text{lastmindch} + y)$;

Let $\text{curbnd} = h(e(0), \text{lastmindch} + y)$.

Let $\text{curbot} = \text{curtop} + 1$.

Let $\text{curtop} = \text{curtop} + 1 + \text{curbnd}$.

Let $\text{lastmindch} = \text{lastmindch} + y$

Let i be the least value such that $\text{num}_i^{\bar{v}} \neq \text{num}_i^v$ (So, $\text{prod}_{i-1}^v = \text{prod}_{i-1}^{\bar{v}}$. Also, for $i \leq k \leq m - 1$, $\varphi_{e(0)}(\langle \text{prod}_k^{\bar{v}}, \cdot \rangle)$ has no non-zero values wherever it is defined till now.)

Let x be the least value such that $\varphi_{e(0)}(\langle \text{prod}_{i-1}^{\bar{v}}, x \rangle)$ has not been defined till now. Let

$\varphi_{e(0)}(\langle \text{prod}_{i-1}^v, x \rangle) = \langle \text{prod}_i^{\bar{v}}, 1 + \text{num}_{i+1}^{\bar{v}} \rangle$.

for $k = i$ **to** $m - 2$ **do**

Let x be the least value such that $\varphi_{e(0)}(\langle \text{prod}_k^{\bar{v}}, x \rangle)$ has not been defined till now.

$\varphi_{e(0)}(\langle \text{prod}_k^{\bar{v}}, x \rangle) = \langle \text{prod}_{k+1}^{\bar{v}}, 1 + \text{num}_{k+2}^{\bar{v}} \rangle$;

endfor

Let x be the least value such that $\varphi_{e(0)}(\langle \text{prod}_{m-1}^{\bar{v}}, x \rangle)$ has not been defined till now.

Let $\varphi_{e(0)}(\langle \text{prod}_{m-1}^{\bar{v}}, x \rangle) = e(\text{curtop})$;

For $x_s \leq z < \max(\{\langle \text{prod}_{i-1}^{\bar{v}}, x \rangle, \langle \text{prod}_i^{\bar{v}}, \dots, \langle \text{prod}_{m-1}^{\bar{v}} \rangle\})$ such that $\varphi_{e(0)}(z)$ has not been defined till now, let $\varphi_{e(0)}(z) = 0$.

Go to stage $s + 1$.

4. Let $\text{current} = \text{current} + 1$. For $i, \langle 0, x \rangle$ as found in step 2.3, let $\varphi_{e(0)}(\langle 0, x \rangle) = \varphi_i(\langle 0, x \rangle) + 1$.

For $x_s \leq y < \langle 0, x \rangle$, let $\varphi_{e(0)}(y) = 0$.

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

Go to stage $s + 1$.

5. For $x_s \leq x \leq y$, let $\varphi_{e(0)}(x) = \varphi_{e(\text{current})}(x)$.

Go to stage $s + 1$.

End stage s

Case 1: Each stage is entered and terminates.

Since $<_o$ is well-founded, step 2.2 can succeed only a finite number of times. The value of curbnd is changed only when step 2.2 succeeds. Since each time step 2.3 occurs, a new $i < \text{curbnd}$ is cancelled, step 2.3 can succeed only a finite number of times in between occurrences of step 2.2. Thus, all but finitely often, step 2.4 succeeds. Also, since the values of current and curtop are changed only when step 2.2 or step 2.3 succeeds, they eventually stabilize. Let current_{fin} and curtop_{fin} be the eventual values of these variables.

Furthermore, $\varphi_{e(0)} = \varphi_{e(\text{current}_{fin})}$, and is total. Let $f = \varphi_{e(0)}$. We now show that $f \in \mathcal{S}_{\omega^{m+1}}$. Firstly, $\lim_{t \rightarrow \infty} f(\langle 1, t \rangle) = e(\text{current}_{fin})$. Next, $f(\langle 0, 0 \rangle) = \langle \text{prod}_0^v, 1 + \text{num}_v^1 \rangle$, where $v = \text{tfcounter}(e(0), t)$, for t as found before stage 0. The only cause for $f(\langle \text{prod}_0^v, x \rangle)$ to take a non-zero value is either before stage 0 or when the step just before the **for** loop in step 3 occurs with $i = 1$. This can happen, in all, only $1 + \text{num}_1^v$ times, which is $\leq \pi_2(f(\langle 0, 0 \rangle))$. Similarly, it can be shown that for each $0 < k \leq m$, for each $\langle p, n \rangle \in L_k^f$, $\text{card}(\{x \mid f(\langle p, x \rangle) \neq 0\}) \leq n$. Also, current is always $\leq \text{curtop}$ since step 4 can occur at most $\text{curbnd} + 1$ times between occurrences of step 3. Finally, $e(\text{curtop}) \in L_{m+1}^f$ at all stages, and since e is 1-1 increasing, $e(\text{current}_{fin}) \leq e(\text{curtop}_{fin})$. Hence, $f \in \mathcal{S}_{\omega^{m+1}}$. However, $\mathbf{M}(f) \uparrow$ or $\mathbf{M}(f) \not\leq \text{curbnd} = \lim_{t \rightarrow \infty} h(p(0), t)$.

So, $f \notin \mathbf{Lim}_{\omega^m} \mathbf{Mex}$ as witnessed by \mathbf{M} and g .

Case 2: Some stage s starts, but does not terminate.

Let current_{fin} , curbnd_{fin} and curtop_{fin} be the final values of current , curbnd and curtop (i.e., those before stage s starts). Let $f = \varphi_p(\text{current}_{fin})$. It can be argued on lines similar to those in *Case 1* that $f \in \mathcal{S}_{\omega^{m+1}}$. \mathbf{M} , on all but finitely many initial segments of f outputs a program $\leq \text{curbnd}_{fin}$ (otherwise step 2.4. would succeed). However, for all $i \leq \text{curbnd}_{fin}$, either $i \in \text{Cancel}$, and thus $\varphi_i \neq f$, or φ_i diverges on infinitely many inputs (otherwise step 2.3. would succeed). It follows that \mathbf{M} does not **Ex-identify** f .

From the above cases it follows that $\mathcal{S}_{\omega^{m+1}} \notin \mathbf{Lim}_{\omega^m} \mathbf{Mex}$ as witnessed by \mathbf{M} and g . \blacksquare

The immediately previous theorem leaves unanswered the comparisons of $\mathbf{Lim}_u \mathbf{Mex}$ with $\mathbf{Lim}_v \mathbf{Mex}$, when $|u|_o \geq \omega^\omega$ and $u \times_o w \leq_o v$. The next theorem partly resolves some of these comparisons.

Theorem 14 For all notations u , there exists a notation $v >_o u$ such that, for all n , $\mathbf{Lim}_v \mathbf{Mex} - \mathbf{Lim}_u \mathbf{Mex}^n \neq \emptyset$.

PROOF. We assume without loss of generality that $|u|_o \geq \omega$. Consider the following class of functions.

$$\begin{aligned} \mathcal{C} = \{f \mid & \\ & f(\langle 3, \infty \rangle) \downarrow = p \wedge \varphi_p = f \wedge \\ & f(\langle 2, \infty \rangle) \downarrow \geq p \wedge \\ & (\forall z)[f(\langle 2, z \rangle) \neq f(\langle 2, z+1 \rangle) \Rightarrow f(\langle 1, z \rangle) \neq f(\langle 1, z+1 \rangle)] \wedge \\ & (\forall z)[f(\langle 1, z \rangle) \in \mathbf{O}] \wedge \\ & (\forall z)[f(\langle 1, z \rangle) \geq_o f(\langle 1, z+1 \rangle)] \wedge \\ & f(\langle 1, 0 \rangle) = u \\ & \} \end{aligned}$$

It can be shown, using a variant of the proof of the negative part of Theorem 13, that $\mathcal{C} \notin \mathbf{Lim}_u \mathbf{Mex}^n$. We will show that there exists a $v >_o u$ such that $\mathcal{C} \in \mathbf{Lim}_v \mathbf{Mex}$.

Let \mathbf{M} be such that, for every n , $\mathbf{M}(f[n])$ is defined as follows. Let $i_n = \max(\{j \mid \langle 3, j \rangle < n\})$. Let $\mathbf{M}(f[n]) = f(\langle 3, i_n \rangle)$. This machine clearly **Ex**-identifies any $f \in \mathcal{C}$. We will construct a \mathbf{Lim}_v function, g , such that for every $f \in \mathcal{C}$, $g(\text{MinProg}(f)) \geq \mathbf{M}(f)$.

To this end let $\eta_1(j, t) = \varphi_j(\langle 2, t \rangle)$, and $\eta_2(j, t) = \varphi_j(\langle 1, t \rangle)$. Let h' and $tfcounter'$ be such that the following 6 conditions are satisfied. (Note that such a h' , $tfcounter'$ can be easily constructed; we omit the details).

1. h' and $tfcounter'$ are total, computable functions.
2. $(\forall j, z)[h'(j, z) \neq h'(j, z+1) \Rightarrow tfcounter'(j, z) \neq tfcounter'(j, z+1)]$.
3. $(\forall j, z)[tfcounter'(j, z) \in \mathbf{O}]$.
4. $(\forall j, z)[tfcounter'(j, z) \geq_o tfcounter'(j, z+1)]$.
5. $(\forall j)[tfcounter'(j, 0) = u +_o \mathbf{1}]$.
6. **if** $\eta_1(j, \cdot)$ and $\eta_2(j, \cdot)$ are such that

$$\begin{aligned} & [(\forall t)[\eta_1(j, t) \downarrow \wedge \eta_2(j, t) \downarrow] \wedge \\ & (\forall z)[\eta_1(j, z) \neq \eta_1(j, z+1) \Rightarrow \eta_2(j, z) \neq \eta_2(j, z+1)] \wedge \\ & (\forall z)[\eta_2(j, z) \in \mathbf{O}] \wedge \\ & (\forall z)[\eta_2(j, z) \geq_o \eta_2(j, z+1)] \wedge \\ & [\eta_2(j, 0) = u] \end{aligned}$$

then $\lim_{t \rightarrow \infty} [\eta_1(j, t)] = h'(j, \infty)$.

Let $g'(y) = h(y, \infty)$, for every y . Clearly, g' is a lim-computable function. Also, $g'(\text{MinProg}(f)) \geq \mathbf{M}(f)$, for $f \in \mathcal{C}$.

We will next show that there exists a v such that some lim_o -computable function dominates g' .

Let $v = \mathbf{2} \exp_o(u \times_o w)$;

Define h and $tfcounter$ as follows.

For all j, t , $h(j, t) = \max(\{h'(i, t) \mid i \leq j\})$.

For all j , $tfcounter(j, t) = v$, if $t = 0$.

$tfcounter(j, t) = \text{match}(tfcounter'(j, t), \dots, tfcounter'(0, t))$, for $t > 0$, where match is as defined below.

(To avoid unnecessary complexity, we allow match to take a variable number of arguments.)

Let $\mathbf{2}i$ be the notation for $2i$. We now define match .

Begin match(u_j, u_{j-1}, \dots, u_0)

For $1 \leq i \leq j$, let $u'_i = (u \times_o \mathbf{2i}) +_o u_i$ and $v_i = \mathbf{2} \exp_o u'_i$.

Let $\text{sum}_0 = v_0$; for $i < j$, let $\text{sum}_{i+1} = v_{i+1} +_o \text{sum}_i$.

(Note that sum_j is just the summation of v_i 's in a *right associative* manner.)

Let match(u_j, u_{j-1}, \dots, u_0) = sum_j .

End match

We now define g as follows. For all y , $g(y) = h(y, \infty)$. Clearly g dominates g' . We will next show that g is indeed lim_v -computable as witnessed by h and $tfcounter$.

Note that (in the definition of match), since for all i , $1 \leq i \leq j$, u_i is $\leq_o u +_o \mathbf{1}$, we have that $u'_i <_o u'_{i+1}$, for $0 \leq i < j$. The following claim is helpful.

Claim 1 Suppose $j \in N$, and $u_0, \dots, u_j \in O$, for $i \leq j$, are given. Assume further that $u_i \leq u +_o \mathbf{1}$, for $i \leq j$. For $i \leq j$, let sum_i, u'_i and v_i be as defined in the procedure for match(u_j, \dots, u_0). Then, for $k < j$, $v_{k+1} >_o \text{sum}_k$.

PROOF. By induction on k .

Base Case: $k = 1$.

Since $u'_0 <_o u'_1$, $\text{sum}_0 = \mathbf{2} \exp_o u'_0 <_o \mathbf{2} \exp_o u'_1 = v_1$ (using properties of $+_o$ and \exp_o from Theorems 7 and 11). Hence, $v_1 >_o \text{sum}_0$.

Inductive Case: Suppose the claim is true for $k = r - 1$, where $r < j - 1$. We will show that it holds for $k = r$. $\text{sum}_r = v_r +_o \text{sum}_{r-1} <_o v_r +_o v_r$ (by the inductive hypothesis) = $(\mathbf{2} \exp_o u'_r) +_o (\mathbf{2} \exp_o u'_r) = \mathbf{2} \exp_o (u'_r +_o \mathbf{1}) \leq_o \mathbf{2} \exp_o u'_{r+1} = v_{r+1}$.

This proves the inductive hypothesis. \square

We continue with the proof of Theorem 14.

Let j be an arbitrary, fixed value. We note that $h(j, t) \neq h(j, t + 1)$ implies that for some $i \leq j$, $tfcounter'(i, t) \neq tfcounter'(i, t + 1)$. To show that g is lim_o -computable, it is then sufficient to show that, given, for $i \leq j$, $u_i \leq_o u +_o \mathbf{1}$ and $\bar{u}_k <_o u_k$, for some $k \leq j$, $[\text{match}(u_j, \dots, u_k, \dots, u_0) <_o \text{match}(u_j, \dots, \bar{u}_k, \dots, u_0)]$.

Now, in the definition of match, for $0 \leq i \leq j$, let sum_i, u'_i and v_i be as defined in the procedure for match(u_j, \dots, u_0); also, let $\overline{\text{sum}}_i, \bar{u}'_i$ and \bar{v}_i be the values of sum_i, u'_i, v_i respectively, as defined in the procedure for match($u_j, \dots, u_{k+1}, \bar{u}_k, u_{k-1}, \dots, u_0$). Thus, to show that $\text{sum}_j >_o \overline{\text{sum}}_j$, it suffices to show $v_k >_o \overline{\text{sum}}_k$. But by Claim 1, we have $\bar{v}_k >_o \overline{\text{sum}}_{k-1}$ and thus, $\overline{\text{sum}}_k <_o \bar{v}_k \times_o \mathbf{2} = \mathbf{2} \exp_o (\bar{u}'_k +_o \mathbf{1}) \leq_o \mathbf{2} \exp_o u'_k = v_k$. Hence $\text{sum}_j >_o \overline{\text{sum}}_j$. \blacksquare

As a corollary to Theorem 14, we have

Corollary 3 For all $u \in O$, $\mathbf{LimMex} - \mathbf{Lim}_u \mathbf{Mex}^n \neq \emptyset$.

Theorem 4 of Section 5 is a consequence of the immediately above result.

We conjecture that, for all constructive ordinals α , for suitable $u \in O$ such that $|u|_o = \alpha$, Theorem 13 can be extended to: $\mathbf{Lim}_{u \times_o w} \mathbf{Mex} - \mathbf{Lim}_u \mathbf{Mex}_n \neq \emptyset$. We are a bit more confident of this conjecture for $\alpha < \text{small constructive epsilon numbers}$ [Sie65, CK37].

Just as we iterated limits in Section 6.1, we can do the same with our new definition of $\mathbf{Lim}_u \mathbf{Mex}_b^a$ and study the learning classes $\mathbf{Lim}_{u_1, \dots, u_n}^n \mathbf{Mex}_b^a$. Generally, except for the cases noted above and their trivial consequences, we do not know how the learning classes $\mathbf{Lim}_{u_1, \dots, u_n}^n \mathbf{Mex}_b^a$ compare to one another.

References

- [Bar74] J. M. Barzdin. Two theorems on the limiting synthesis of functions. *Theory of Algorithms and Programs, Latvian State University, Riga*, 88:82–88, 1974.
- [BB75] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.
- [Blu67] M. Blum. A machine independent theory of the complexity of recursive functions. *Journal of the ACM*, 14:322–336, 1967.
- [Che81] K. Chen. *Tradeoffs in Machine Inductive Inference*. PhD thesis, Computer Science Department, SUNY at Buffalo, 1981.
- [Che82] K. Chen. Tradeoffs in inductive inference of nearly minimal sized programs. *Information and Control*, 52:68–86, 1982.
- [CJS89] J. Case, S. Jain, and A. Sharma. Convergence to nearly minimal size grammars by vacillating learning machines. In R. Rivest, D. Haussler, and M.K. Warmuth, editors, *Proceedings of the Second Annual Workshop on Computational Learning Theory, Santa Cruz, California*, pages 189–199. Morgan Kaufmann Publishers, Inc., August 1989. Journal version in press for *Journal of Computer and System Sciences*.
- [CK37] A Church and S. C. Kleene. Formal definitions in the theory of ordinal numbers. *Fundamenta Mathematicae*, 28:11–21, 1937.
- [CS83] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.
- [Ers68] Y.L. Ershov. A hierarchy of sets II. *Algebra and Logic*, 7:212–232, 1968.
- [FK77] R. Freivalds and E. B. Kinber. Limit identification of minimal Gödel numbers. *Theory of Algorithms and Programs 3;Riga 1977*, pages 3–34, 1977.
- [Fre75] R. Freivalds. Minimal Gödel numbers and their identification in the limit. *Lecture Notes in Computer Science*, 32:219–225, 1975.
- [Fre90] 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.
- [FS91] R. Freivalds and C. H. Smith. On the role of procrastination for machine learning. Technical Report TR-91-27, UMIACS, 1991. To appear in *Information and Computation*.
- [Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- [JS91] S. Jain and A. Sharma. Program size restrictions in inductive learning. In *AAAI Symposium Series, Symposium: Machine Learning of Natural Language and Ontology*, March 1991. Journal version accepted by *Theoretical Computer Science*.

- [Kin74] E. B. Kinber. On the synthesis in the limit of almost minimal Gödel numbers. *Theory Of Algorithms and Programs, LSU, Riga*, 1:221–223, 1974.
- [Kin77a] E. B. Kinber. On a theory of inductive inference. *Lecture Notes in Computer Science*, 56:435–440, 1977.
- [Kin77b] E. B. Kinber. On limit identification of minimal Gödel numbers for functions from enumerable classes. *Theory of Algorithms and Programs 3;Riga 1977*, pages 35–56, 1977.
- [Kin83] E. B. Kinber. A note on limit identification of c-minimal indices. *Electronische Informationverarbeitung und Kybernetik*, 19:459–463, 1983.
- [Kle38] S. C. Kleene. Notations for ordinal numbers. *Journal of Symbolic Logic*, 3:150–155, 1938.
- [Kle55] S. C. Kleene. On the forms of predicates in the theory of constructive ordinals, II. *American Journal of Mathematics*, 77:405–428, 1955.
- [Mar89] Y. Marcoux. Composition is almost as good as s-1-1. In *Proceedings, Structure in Complexity Theory—Fourth Annual Conference*. IEEE Computer Society Press, 1989.
- [MY78] M. Machtey and P. Young. *An Introduction to the General Theory of Algorithms*. North Holland, New York, 1978.
- [Ric80] G. Riccardi. *The Independence of Control Structures in Abstract Programming Systems*. PhD thesis, SUNY Buffalo, 1980.
- [Ric81] G. Riccardi. The independence of control structures in abstract programming systems. *Journal of Computer and System Sciences*, 22:107–143, 1981.
- [Rog58] H. Rogers. Gödel numberings of partial recursive functions. *Journal of Symbolic Logic*, 23:331–341, 1958.
- [Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York, 1967. Reprinted, MIT Press, 1987.
- [Roy87] J. Royer. *A Connotational Theory of Program Structure*. Lecture Notes in Computer Science 273. Springer Verlag, 1987.
- [Sac90] G. E. Sacks. *Higher Recursion Theory*. Springer-Verlag, 1990.
- [Sha71] 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.
- [Sie65] W. Sierpinski. *Cardinal and ordinal numbers*. PWN –Polish Scientific Publishers, 1965. Second revised edition.