Machine Learning of Higher Order Programs*

Ganesh Baliga† John Case ‡ Sanjay Jain§ Mandayam Suraj¶

March 11, 2007

Abstract

A *generator program* for a computable function (by definition) generates an infinite sequence of programs all but finitely many of which compute that function. Machine learning of generator programs for computable functions is studied. To partially motivate these studies, it is shown that, in some cases, interesting global properties for computable functions can be proved from suitable generator programs which can *not* be proved from *any* ordinary programs for them. The power (for variants of various learning criteria from the literature) of learning generator programs is compared with the power of learning ordinary programs. The learning power in these cases is also compared to that of learning *limiting programs*, i.e., programs allowed finitely many mind changes about their correct outputs.

1 Preliminaries

1.1 Notation

Any unexplained recursion theoretic notation is from [Rog67]. N denotes the set of natural numbers, $\{0, 1, 2, 3, \ldots\}$. Unless otherwise specified, $b, e, i, j, k, l, m, n, p, r, s, t, w, x, y, z$, with or without decorations[1],

*Preliminary version of this paper appeared in the *Proceedings of the Second Symposium on Logical Foundations of Computer Science*, Tver, Russia, July, 1992.

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

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

§Institute of Systems Science, National University of Singapore, Singapore 0511. Email: sanjay@iss.nus.sg.

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

[1]Decorations are subscripts, superscripts and the like.

range over N. $*$ denotes a non-member of N and is assumed to satisfy $(\forall n)[n < * < \infty]$. a with or without decorations, ranges over $N \cup \{*\}$. $\emptyset$ denotes the empty set. $\subseteq$ denotes subset. $\subset$ denotes proper subset. For S, a subset of N, $\mathrm{card}(S)$ denotes the cardinality of S. $\uparrow$ denotes undefined. $\max(\cdot), \min(\cdot)$ denote the maximum and minimum of a set, respectively, where $\max(\emptyset) = 0$ and $\min(\emptyset) =\uparrow$.

η and θ range over *partial* functions with arguments and values from N. $\eta(x)\downarrow$ denotes that $\eta(x)$ is defined; $\eta(x)\uparrow$ denotes that $\eta(x)$ is undefined.

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

We say that η is *monotone* $\overset{\mathrm{def}}{\Leftrightarrow}$ $(\forall x, y \mid x < y)[\eta(x)\downarrow < \eta(y)\downarrow]$. Thus η is monotone iff η is a *strictly* increasing total function.

$\langle i, j\rangle$ stands for an arbitrary, computable, one-to-one encoding of all pairs of natural numbers onto N [Rog67]. Similarly we can define $\langle \cdot, \ldots, \cdot \rangle$ for encoding multiple natural numbers onto N.

φ denotes a fixed *acceptable* programming system for the partial computable functions: $N \to N$ [Rog58, Rog67, MY78]. φ_i denotes the partial computable function computed by program i in the φ-system. Φ denotes an arbitrary Blum complexity measure [Blu67, HU79] for the φ-system. K denotes $\{p \mid \varphi_p(p)\downarrow\}$. K is a standard r.e. set with $\overline{K}$, the complement of K, being constructively non-r.e. [Rog67].

The set of all total computable functions of one variable is denoted by $\mathcal{R}$. $\mathcal{C}$, with or without decorations, ranges over subsets of $\mathcal{R}$. For computable f, $\mathrm{MinProg}(f)$ denotes $\min(\{i \mid \varphi_i = f\})$.

We sometimes consider partial computable functions with multiple arguments in the φ system. In such cases we implicitly assume that a $\langle \cdot, \ldots, \cdot \rangle$ is used to code the arguments, so, for example, $\varphi_i(x, y)$ stands for $\varphi_i(\langle x, y\rangle)$.

The quantifier '$\overset{\infty}{\forall}$' essentially from [Blu67], means 'for all but finitely many'. '$\exists!$' means 'there exists an unique'.

1.2 Fundamental Function Inference Paradigms

A *Learning Machine* (**LM**) [Gol67] is an algorithmic device which takes as its input a set of data given one element at a time, and which from time to time, as it is receiving its input, outputs programs. **LM**s have been used in the study of machine learning or inductive inference of programs for computable functions as well as algorithmic learning of grammars for languages [BB75, CS83, Che81, Ful85, Gol67, OSW86, Wie78, AS83, KW80, Cas86].[2]

M, with or without decorations, ranges over the class of **LM**s. For the learning of a computable function f by an **LM**, **M**, the graph of f is fed to **M** in any order. Without loss of generality [BB75, CS83], we will assume that **M** is fed the graph of f in the sequence $(0, f(0)), (1, f(1)), (2, f(2)), \ldots$. For all computable functions f, $f[n]$ denotes the finite initial segment $((0, f(0)), (1, f(1)), \ldots, (n-1, f(n-1)))$. Let INIT $= \{f[n] \mid f \in \mathcal{R} \wedge n \in N\}$. σ, with or without decorations, ranges over INIT. $\mathbf{M}(\sigma)$ is the last output of **M** by the time it receives all of σ. For the learning criteria discussed in this paper, we can and will assume, without loss of generality, that $\mathbf{M}(\sigma)$ is always defined. We say that $\mathbf{M}(f)$ *converges to* i (written: $\mathbf{M}(f)\downarrow = i$) iff $(\overset{\infty}{\forall} n)[\mathbf{M}(f[n]) = i]$; $\mathbf{M}(f)$ is undefined if no such i exists.

Recall, that according to our convention $a \in N \cup \{*\}$.

Definition 1 [Gol67, BB75, CS83]

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

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

Case and Smith [CS83] introduced another infinite hierarchy of learning criteria which we describe below. "**Bc**" stands for *behaviorally correct.* Barzdin [Bar74] essentially introduced the notion $\mathbf{Bc}^0$.

Definition 2 [CS83]

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

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

We usually write **Ex** for $\mathbf{Ex}^0$ and **Bc** for $\mathbf{Bc}^0$. Theorem 3 just below states some of the basic hierarchy results about the $\mathbf{Ex}^a$ and $\mathbf{Bc}^a$ classes.

Theorem 3 *For all n,*

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

[2]We have not yet investigated language learning analogs of our results.

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

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

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

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

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

Parts (a), (b), (d), and (e) are due to Case and Smith [CS83]. John Steel first observed that $\mathbf{Ex}^* \subseteq \mathbf{Bc}$ and the diagonalization in part (c) is due to Harrington and Case [CS83]. Part (f) is due to Harrington [CS83]. Blum and Blum [BB75] first showed that $\mathbf{Ex} \subset \mathbf{Ex}^*$. Barzdin [Bar74] first showed that $\mathbf{Ex} \subset \mathbf{Bc}$.

2 Higher Order Programs

2.1 Definition and Motivation of Higher Order Programs

We consider two kinds of higher order programs: *limiting programs* (from [CJS92]) and *generator programs* (introduced in the present paper).

First we discuss limiting programs.

For each i, consider the following corresponding procedure for "computing" a (partial) function $\varphi_i^\star$.

```
On input $x$
    for $t = 0$ to $\infty$
    Start a new clone of $\varphi$-program $i$ running on input $(x, t)$
    endfor
```

It is to be understood that

(a) each iterate of the **for**-loop finishes since it merely *starts* a process running and

(b) in some iterates of the **for**-loop the process *started* may itself never converge.

$\varphi_i^\star(x) \stackrel{\text{def}}{=}$ the unique y (if any) eventually output by all but finitely many of the clones of φ-program i in the **for**-loop above. Equivalently, $\varphi_i^\star(x) \stackrel{\text{def}}{=} \lim_{t \to \infty} \varphi_i(x, t)$.

We shall refer to i as **Lim**-program i (in the $\varphi^\star$-system) when we are thinking of i as encoding the **for**-loop above rather than as encoding φ-program i.

Intuitively, **Lim**-program i (in the $\varphi^\star$-system) is a procedure, which on an input for which it has an output, is allowed to change its mind finitely many times about that output (or even about whether to output at all). N.B. there may be no algorithm for signaling when a **Lim**-program has stopped changing its mind about its output.

The partial functions which are the limit of some *total* computable function are well known to be characterized as exactly the partial functions computable relative to an oracle for the halting problem [Sho59, Put65, Gol65, Sho71, Soa87].[3] This result and its relativizations were first noticed and used by Post [Sha71] and have been employed (sometimes with rediscovery) many times. [LMF76] studied acceptable programming systems for partial functions computable relative to oracles. Many of the results of this paper about **Lim**-programs would hold also for programs in acceptable oracular programming systems with oracle for the halting problem attached, but we will present our **Lim**-program results directly about systems such as $\varphi^\star$.

In the present paper, as in [CJS92], we shall be especially interested in **Lim**-programs (from the $\varphi_i^\star$ system) which happen to compute *partial computable* functions. The learning of **Lim**-programs for *computable* functions is compared to the learning of ordinary φ-programs in [CJS92].

The reader might think that **Lim**-programs for computable functions f are not particularly useful, but with such programs one can discover values for f *eventually.* However, one may not know *when* one has found those values, and it is easy to argue that "eventually" is too long to wait.

Actually, **Lim**-programs can be quite useful.

In physics it is sometimes easier to infer a *global* property of a phenomenon than it is to make more detailed predictions about observations; for example, Kepler's Law that the planets orbit in ellipses is easier to derive than equations of motion of planets. In Section 2.2 we state a result, Theorem 5, extending a result from [CJS92], that it is, in some cases, possible to prove *global* properties of a computable function from a suitable **Lim**-program for it when it is *not* possible to prove these properties from *any* of the ordinary (φ) programs for it.

Next we discuss generator programs.

[3]The class of partial functions which are the limit of some *partial* computable function, i.e., $\{\varphi_i^\star \mid i \in N\}$, is a larger class than the class of partial functions computable in the halting problem.

Informally, a φ-program p is a 0-*generator program* for f just in case φ_p is total and all but finitely many of the programs $\varphi_p(0), \varphi_p(1), \varphi_p(2), \ldots$ compute f.[4]

Definition 4 We say that φ-program p is an a-*generator* for f iff

(a) $\varphi_p \in \mathcal{R}$ and

(b) $(\overset{\infty}{\forall} x)[\varphi_{\varphi_p(x)} =^a f]$.

Our remarks above about the possible usefulness of **Lim**-programs can be applied *mutatis mutandis* to a-generators. An a-*program* for f is a program p such that $\varphi_p =^a f$. In the next section (Section 2.2) we present results to the effect that, for $a = 0, 1$, respectively, it is, in some cases, possible to prove global properties of a computable function from a suitable a-generator program for it when it is *not* possible to prove these properties from *any* of the ordinary b-programs for it, for $b = 0, *$, respectively. We also present such a result comparing **Lim**-programs with 0-generator programs.

2.2 Further Motivation

We next provide the preliminaries for obtaining the provability results as advertised above in the previous section (Section 2.1).

We present our results for extensions of first order arithmetic. Regarding expressing propositions in first order arithmetic, we shall proceed informally. If E is an expression such as '$\Phi_i \leq t$', or 'φ_i is monotone', we shall write $\ll \mathsf{E} \gg$ to denote a naturally corresponding, fixed standard wff of first order arithmetic [Men86] which (semantically) expresses E. We need and assume that

> if E' is obtained from E by changing some numerical values, then $\ll \mathsf{E}' \gg$ can be *algorithmically* obtained from those changed numerical values and $\ll \mathsf{E} \gg$.

It is understood that, if E contains references to partial functions, such as φ and Φ, then in $\ll \mathsf{E} \gg$ these are, in effect, named by standard programs for them. It is well known that wffs extensionally equivalent (with respect to standard models) may not be intensionally or provably equivalent [Fef60]. In what follows, when we use the $\ll \mathsf{E} \gg$ notation, it will always be for propositions that are easily seen to be (semantically) expressible in first order arithmetic.[5] '$\vdash$' denotes the provability relation.

[4]It is without loss of generality for our learning criteria introduced in Section 3 below that we require φ_p be total.

[5]This informal discussion of provability and expressibility is based on Section 4.3 of [RC92].

The following theorem extends slightly a theorem from [CJS92] and motivates the usefulness of **Lim**-programs over ordinary φ-programs. The furthermore clause is new. Grigori Schwarz suggested to us the problem of whether it could be added.

Theorem 5 *Suppose* **T** *is an axiomatizable (i.e., r.e.* [Cra53]*) first order theory which extends Peano Arithmetic* [Men86] *and in which one can* not *prove anything false about monotonicity of (partial) computable functions computed by programs in φ. Then there exist $f \in \mathcal{R}$ and e such that $\varphi^\star_e = f$ and f is monotone, yet*

(a) $(\forall i \mid \varphi_i = f)[\mathbf{T} \nvdash \ll \varphi_i$ *is monotone* $\gg]$ and

(b) $[\mathbf{T} \vdash \ll \varphi^\star_e$ *is monotone* $\gg]$.

Furthermore, $[\mathbf{T} \vdash \ll \varphi^\star_e$ *is computable* $\gg]$.

The proof of Theorem 5 can be obtained by a simple modification of its predecessor in [CJS92]. The proof of Theorem 11 below can be similarly obtained. We omit the details for both. An anonymous referee nicely pointed out that, if we replace the global property of 'monotone' in Theorems 5 and 11 by 'total' and note that (i) there is an r.e. set of **Lim**-programs for $\mathcal{R}$ and (ii) there is neither an r.e. set of (ordinary) programs nor an r.e. set of 0-generators for $\mathcal{R}$, then these modified theorems involve the well studied provably recursive functions [Kre51, Kre58, Fis65, Rog57, Ros84] and quite easily follow. We originally chose to work with monotonicity, rather than totality, since it is about the global shape of a curve and, hence, a better analog of the elliptical shape of orbits.

Next we present the advertised two theorems (Theorem 6 and 10) motivating the usefulness of generator programs over ordinary programs. The range containment property featured in Theorem 6 is a somewhat technical global property for computable functions. Theorem 10 deals with a variant of the monotonicity property of functions, and this property is clearly an interesting global property.

Theorem 6 *Suppose* **T** *is an axiomatizable (i.e., r.e.) first order theory which extends Peano Arithmetic and in which one can* not *prove anything false about the containment of ranges of (partial) computable functions (computed by programs in φ) in $\overline{K}$. Then there exist $f \in \mathcal{R}$ and p_0 such that φ_{p_0} is total, p_0 is a* 0*-generator for f, and* $\mathrm{range}(f) \subseteq \overline{K}$*, yet*

(a) $(\forall i \mid \varphi_i = f)[\mathbf{T} \nvdash \ll \mathrm{range}(\varphi_i) \subseteq \overline{K} \gg]$ and

(b) $\mathbf{T} \vdash \ll (\overset{\infty}{\forall}\, t)[\mathrm{range}(\varphi_{\varphi_{p_0}(t)}) \subseteq \overline{K}] \gg$.

Furthermore, $\mathbf{T} \vdash \ll [(\overset{\infty}{\forall}\, t)[\varphi_{\varphi_{p_0}(t)} \in \mathcal{R}] \wedge (\overset{\infty}{\forall}\, t)[\varphi_{\varphi_{p_0}(t)} = \varphi_{\varphi_{p_0}(t+1)}]] \gg$.

Proof. Suppose the hypotheses. Fix an automatic theorem prover for $\mathbf{T}$. In what follows, any reference to proving something in $\mathbf{T}$ within so many steps refers to steps in the execution of this automatic theorem prover.

Note that, since $\overline{K}$ is not r.e., $\{x \mid x \in \overline{K} \wedge (\forall i \mid x \in \mathrm{range}(\varphi_i))[\mathbf{T} \nvdash \ll \mathrm{range}(\varphi_i) \subseteq \overline{K} \gg]\}$ is not empty; hence, $z_0 \overset{\mathrm{def}}{=} \min(\{x \mid x \in \overline{K} \wedge (\forall i \mid x \in \mathrm{range}(\varphi_i))[\mathbf{T} \nvdash \ll \mathrm{range}(\varphi_i) \subseteq \overline{K} \gg]\})$ is in $\overline{K}$.

Let z_1 be such that $\mathbf{T} \vdash \ll z_1 \in \overline{K} \gg$.

For each t, let $S_t = \{\varphi_j(y) \mid j \leq t \wedge y \leq t \wedge \Phi_j(y) \leq t$ and $\mathbf{T} \vdash \ll \mathrm{range}(\varphi_j) \subseteq \overline{K} \gg$ in $\leq t$ steps $\}$ and let $T_t = \{x \leq t \mid \Phi_x(x) \leq t\}$. Note that, for all t, $S_t \subseteq \overline{K}$ and $T_t \subseteq K$. Moreover, canonical indices [Rog67] for the finite sets S_t and T_t can be found algorithmically from t.

Let h, computable, be such that for all x, y, $\varphi_{h(x)}(y) = x$ (by Kleene's s-m-n theorem [Rog67] such an h exists).

Let p_0 be such that,

$$\varphi_{p_0}(t) = \begin{cases} h(\min(\overline{S_t \cup T_t})), & \text{if } S_t \cap T_t = \emptyset\ ; \\ h(z_1), & \text{otherwise.} \end{cases}$$

A simple analysis shows that $z_0 = \lim_{t\to\infty}[\min(\overline{S_t \cup T_t})]$. Let $f = \lambda x.z_0$. Note that $(\overset{\infty}{\forall}\, t)[\varphi_{\varphi_{p_0}(t)} = f]$. Clearly, $(\forall i \mid \varphi_i = f)[\mathbf{T} \nvdash \ll \mathrm{range}(\varphi_i) \subseteq \overline{K} \gg]$ (by the definition of z_0). Since, in Peano Arithmetic, it can be proved that $\ll [(\forall t)[S_t \cap T_t = \emptyset] \Rightarrow [\lim_{t\to\infty}(\overline{S_t \cup T_t}) \neq \emptyset]] \gg$ and also that $\ll [(\exists t)[S_t \cap T_t \neq \emptyset] \Rightarrow (\overset{\infty}{\forall}\, t)[S_t \cap T_t \neq \emptyset]] \gg$, it is easy to show in Peano Arithmetic that $\ll [(\overset{\infty}{\forall}\, t)[\mathrm{range}(\varphi_{\varphi_{p_0}(t)}) \subseteq \overline{K}]] \gg$. Furthermore, it can be shown in Peano Arithmetic that $[(\forall t)[\varphi_{\varphi_{p_0}(t)} \in \mathcal{R}] \wedge (\overset{\infty}{\forall}\, t)[\varphi_{\varphi_{p_0}(t)} = \varphi_{\varphi_{p_0}(t+1)}]]$. □

We will now consider the monotonicity result for the 1-generator programs. First we present a few definitions.

Definition 7 *Let $a \in N \cup \{*\}$. We say that p is an a-*nice_generator *for $f \in \mathcal{R}$, iff $\varphi_p \in \mathcal{R}$, and, either*

(a) $(\overset{\infty}{\forall}\, n)[\varphi_{\varphi_p(n)} = f]$ *or*

(b) $(\exists! g \in \mathcal{R})[(\overset{\infty}{\forall}\, n)[\varphi_{\varphi_p(n)} =^a g]] \wedge (\overset{\infty}{\forall}\, n)[\varphi_{\varphi_p(n)} =^a f]$.

Definition 8 *Let $a \in N \cup \{*\}$. We say that p is a-*nice_gen_monotone *iff $(\exists f \in \mathcal{R} \mid f$ is monotone$)[p$ is a-nice_generator for $f]$.*

Definition 9 *Let $a \in N \cup \{*\}$. We say that i is a*-monotone *iff $(\exists f \in \mathcal{R} \mid f$ is monotone$)[\varphi_i =^a f]$.*

Theorem 10 *Suppose* **T** *is an axiomatizable (i.e., r.e.) first order theory which extends Peano Arithmetic, and in which one can* not *prove anything false of the form 'i is $*$-monotone'. Then there exist $f \in \mathcal{R}$ and p_0 such that φ_{p_0} is total, p_0 is 1-nice_generator for f, and f is monotone, yet*

(a) $(\forall i \mid \varphi_i =^* f)[\mathbf{T} \nvdash \ll i$ *is $*$-monotone* $\gg]$ and

(b) $[\mathbf{T} \vdash \ll p_0$ *is* 1-*nice_gen_monotone* $\gg]$.

Proof. Suppose the hypotheses. Fix an automatic theorem prover for **T**. As in the proof of Theorem 6 above, in what follows any reference to proving something in **T** within so many steps refers to steps in the execution of this automatic theorem prover.

By the operator recursion theorem [Cas74] there exists a p_0, such that the (partial) functions $\varphi_{\varphi_{p_0}(\cdot)}$ may be described as follows. The (partial) functions $\varphi_{\varphi_{p_0}(\cdot)}$ are described in stages.

Let $\varphi_{\varphi_{p_0}(0)}(0) = 0$. x_s denotes the least input on which $\varphi_{\varphi_{p_0}(0)}$ has not been defined before stage s. l_s denotes the least number such that, $\varphi_{\varphi_{p_0}(l_s)}$ has not been defined on any input before the start of stage s. Thus $x_0 = 1$ and $l_0 = 1$. Go to stage 0.

```
Begin stage $s$

1. For all $x < x_s$, let $\varphi_{\varphi_{p_0}(l_s)}(x) = \varphi_{\varphi_{p_0}(0)}(x)$.
2. Let $r_s = l_s$ and $y = x_s - 1$.
3. Let $P = \{j \leq s \mid \mathbf{T} \vdash \ll j$ is $*$-monotone $\gg$ in $\leq s$ steps$\}$.
4. repeat
   4.1 $r_s = r_s + 1; y = y + 1$.
   4.2 For $x < y$, let $\varphi_{\varphi_{p_0}(r_s)}(x) = \varphi_{\varphi_{p_0}(l_s)}(x)$.
       For $x \geq y$, whenever $\varphi_{\varphi_{p_0}(l_s)}(x)$ gets defined, let $\varphi_{\varphi_{p_0}(r_s)}(x) = \varphi_{\varphi_{p_0}(l_s)}(x)$ (i.e. $\varphi_{\varphi_{p_0}(r_s)}$ "follows"
         $\varphi_{\varphi_{p_0}(l_s)}$ from now on. Note that because of this step $\varphi_{\varphi_{p_0}(r_s)} = \varphi_{\varphi_{p_0}(l_s)}$).
   4.3 Let $\varphi_{\varphi_{p_0}(l_s)}(y) = \varphi_{\varphi_{p_0}(l_s)}(y-1) + \text{card}(P) + 2$.
   4.4 If there exists an $x$, $x_s \leq x < y$, such that $(\forall j \in P)[\Phi_j(x) \leq y]$, then let $z$ be the least such $x$, and
         go to step 5.
   forever
```

5. For $x \in \{x \neq z \mid x_s \leq x \leq y\}$, let $\varphi_{\varphi_{p_0}(0)}(x) = \varphi_{\varphi_{p_0}(l_s)}(x)$.

   Let $\varphi_{\varphi_{p_0}(0)}(z) = w$, where $w = \min(\{x \mid \varphi_{\varphi_{p_0}(l_s)}(z) < x < \varphi_{\varphi_{p_0}(l_s)}(z+1)\} - \{\varphi_j(z) \mid j \in P\})$.

   (Note that w is not undefined since, $\varphi_{\varphi_{p_0}(l_s)}(z+1) - \varphi_{\varphi_{p_0}(l_s)}(z) = \text{card}(P) + 2$.)

   For $x > y$, whenever $\varphi_{\varphi_{p_0}(0)}(x)$ gets defined, let $\varphi_{\varphi_{p_0}(l_s)}(x) = \varphi_{\varphi_{p_0}(0)}(x)$ (i.e. $\varphi_{\varphi_{p_0}(l_s)}$ "follows" $\varphi_{\varphi_{p_0}(0)}$ from now on).

   (Note that due to step 5, $\varphi_{\varphi_{p_0}(l_s)} =^1 \varphi_{\varphi_{p_0}(0)}$.)

6. Go to stage $s+1$.

   (Note that $x_{s+1} = y+1$ and $l_{s+1} = r_s + 1$).

End stage s.

Now define f as follows.

$$f = \begin{cases} \varphi_{\varphi_{p_0}(0)}, & \text{if infinitely many stages are executed;} \\ \varphi_{\varphi_{p_0}(l_s)}, & \text{if stage } s \text{ starts but never finishes.} \end{cases}$$

A simple case analysis shows that,

f is monotone, and

p_0 is a 1-nice_generator for f, and thus p_0 is 1-nice_gen_monotone.

Furthermore, this proof of p_0 being 1-nice_gen_monotone can be formalized in Peano Arithmetic.

Now, since $\mathbf{T}$ does not prove anything false about $*$-monotonicity of programs, all stages halt. This implies that $f = \varphi_{\varphi_{p_0}(0)}$. Thus for all j, such that $\mathbf{T} \vdash \ll j$ is $*$-monotone $\gg$, there exist infinitely many x such that, $\varphi_j(x) \neq f(x)$ (by the diagonalization at step 5 on input z, for each stage s). The theorem follows. $\square$

Next is the promised theorem (Theorem 11) comparing **Lim**-programs and 0-generator programs.

Theorem 11 *Suppose* $\mathbf{T}$ *is an axiomatizable (i.e., r.e.) first order theory which extends Peano Arithmetic such that, for each p, one can* not *prove anything false of the form '*$(\overset{\infty}{\forall} t)[\varphi_{\varphi_p(t)}$ *is monotone* $]$*'. Then there exist* $f \in \mathcal{R}$ *and* e *such that* $\varphi_e^\star = f$ *and* f *is monotone, yet*

(a) $(\forall p \mid p$ *is a 0-generator for* $f)[\mathbf{T} \nvdash \ll (\overset{\infty}{\forall} t)[\varphi_{\varphi_p(t)}$ *is monotone* $] \gg]$ and

(b) $[\mathbf{T} \vdash \ll \varphi_e^\star$ *is monotone* $\gg]$.

Furthermore, $[\mathbf{T} \vdash \ll \varphi_e^\star$ *is computable* $\gg]$.

3 Learning Higher Order Programs

Definition 12 [CJS92]

(a) A machine $\mathbf{M}$, $\mathbf{LimEx}^a$*-identifies* f (written: $f \in \mathbf{LimEx}^a(\mathbf{M})$) iff $\mathbf{M}(f)\downarrow = i$ such that $[f =^a \varphi_i^\star]$.

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

(c) $\mathbf{M}$ $\mathbf{LimBc}^a$*-identifies* $f \in \mathcal{R}$ iff $(\overset{\infty}{\forall} n)[\varphi^\star_{\mathbf{M}(f[n])} =^a f]$.

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

We write $\mathbf{LimEx}$ for $\mathbf{LimEx}^0$ and $\mathbf{LimBc}$ for $\mathbf{LimBc}^0$. We do not consider $\mathbf{LimBc}^a$ further since from [CJS92] $\mathcal{R} \in \mathbf{LimBc}$.

It is shown in [CJS92], for example, that

Theorem 13 *For all a, i,*

(a) $\mathbf{Ex}^a \subset \mathbf{LimEx}^a$*,*

(b) $\mathbf{LimEx} - \mathbf{Bc}^i \neq \emptyset$*,*

(c) $\mathbf{Ex}^{i+1} - \mathbf{LimEx}^i \neq \emptyset$*,*

(d) $\mathbf{Ex}^* - \bigcup_{i \in N} \mathbf{LimEx}^i \neq \emptyset$*, and*

(e) $\mathbf{Bc} - \mathbf{LimEx}^* \neq \emptyset$*.*

Definition 14

(a) A machine $\mathbf{M}$, $\mathbf{GenEx}^a$*-identifies* f (written: $f \in \mathbf{GenEx}^a(\mathbf{M})$) iff $\mathbf{M}(f)\downarrow$ to some a-generator for f.

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

(c) A machine $\mathbf{M}$, $\mathbf{GenBc}^a$*-identifies* f (written: $f \in \mathbf{GenBc}^a(\mathbf{M})$) iff $(\overset{\infty}{\forall} n)[\mathbf{M}(f[n])$ is an a-generator for $f]$.

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

We write $\mathbf{GenEx}$ for $\mathbf{GenEx}^0$ and $\mathbf{GenBc}$ for $\mathbf{GenBc}^0$. The terminology in Definition 14 just above should not be confused with that in [CL82].

3.1 Results

The next theorem (Theorem 15) leaves open the questions of whether some containment relations are proper. These questions are settled by the end of the paper.

Theorem 15 *For all a,*

(a) $\mathbf{Ex}^a \subseteq \mathbf{GenEx}^a$,

(b) $\mathbf{GenEx}^a \subseteq \mathbf{Bc}^a$,

(c) $\mathbf{GenEx} = \mathbf{Ex}$, *and*

(d) $\mathcal{R} \in \mathbf{GenBc}$.

Proof.

(a), (b) Easy to prove.

(c) Given $\mathbf{M}$, we will construct $\mathbf{M}'$ such that $\mathbf{GenEx}(\mathbf{M}) \subseteq \mathbf{Ex}(\mathbf{M}')$.

Let $P(p,x,j,t) = [\Phi_p(j) \leq t \wedge \Phi_{\varphi_p(j)}(x) \leq t]$. Note that, for all p,x,j, $(\exists t)[P(p,x,j,t)] \Leftrightarrow (\overset{\infty}{\forall} t)[P(p,x,j,t)] \Leftrightarrow \varphi_{\varphi_p(j)}(x)\downarrow$.

By the Kleene's s-m-n theorem there exists a computable g such that

$$\varphi_{g(p,j)}(x) = \begin{cases} \varphi_{\varphi_p(k')}(x), & \text{if } (\exists k \geq j)(\exists t)[P(p,x,k,t)] \wedge \\ & (\exists t')[\langle k', t'\rangle = \min(\{\langle k,t\rangle \mid k \geq j \wedge P(p,x,k,t)\})]; \\ \uparrow, & \text{otherwise.} \end{cases}$$

Note that if p, j and f are such that

$[p$ is a 0-generator for $f]$ and $[(\forall k \geq j)[\mathrm{card}(\{x \mid \varphi_{\varphi_p(k)}(x)\downarrow \neq f(x)\}) = 0]]$,

then $\varphi_{g(p,j)} = f$.

Let $j_n^f = 1 + \max(\{j \leq n \mid (\exists y < n)[$ For $p = \mathbf{M}(f[n]), P(p,y,j,n) \wedge \varphi_{\varphi_p(j)}(y) \neq f(y)]\})$. Note that if $f \in \mathbf{GenEx}(\mathbf{M})$, and $p = \mathbf{M}(f)$, then for all but finitely many n, $j_n^f = 1 + \max(\{j \mid (\exists y)[\varphi_{\varphi_p(j)}(y)\downarrow \neq f(y)]\})$.

Let $\mathbf{M}'(f[n]) = g(\mathbf{M}(f[n]), j_n^f)$.

It is easy to see, using the property of g discussed above, that $\mathbf{GenEx}(\mathbf{M}) \subseteq \mathbf{Ex}(\mathbf{M}')$.

(d) Let $\mathrm{prog}(f[n],t) = \min(\{n\} \cup \{i \leq n \mid (\forall x < n)[\Phi_i(x) \leq t \wedge \varphi_i(x) = f(x)]\})$.

It is easy to see that, for all $f \in \mathcal{R}$,

$$(\forall n > \text{MinProg}(f))(\overset{\infty}{\forall}\, t)[\text{prog}(f[n], t) = \text{MinProg}(f)] \tag{1}$$

Let $\mathbf{M}$ be such that, for all f, n, $\mathbf{M}(f[n])$ is a program for $\lambda t.[\text{prog}(f[n], t)]$.

Using (1) it immediately follows that $\mathcal{R} \in \mathbf{GenBc}(\mathbf{M})$. $\Box$

Theorem 16 below shows that it may not always be possible to tradeoff anomalies by allowing learning machines to output higher order programs in the limit.

Theorem 16 $(\forall i)[\mathbf{Ex}^{i+1} - \mathbf{GenEx}^i \neq \emptyset]$.

PROOF. Let $\mathcal{C} = \{f \mid \varphi_{f(0)} =^{i+1} f\}$. Clearly, $\mathcal{C}$ is in $\mathbf{Ex}^{i+1}$. $\mathcal{C} \notin \mathbf{GenEx}^i$ can be shown by a simple modification of the proof that $\mathcal{C} \notin \mathbf{Ex}^i$ in [CS83]. We omit the details. $\Box$

Similarly it can be shown that,

Theorem 17 $\mathbf{Ex}^* - \bigcup_{i \in N} \mathbf{GenEx}^i \neq \emptyset$.

As a corollary to Theorems 16 and 17 we have

Corollary 18 $(\forall i)[\mathbf{GenEx}^{i+1} - \mathbf{GenEx}^i \neq \emptyset]$.

Corollary 19 $[\mathbf{GenEx}^0 \subset \mathbf{GenEx}^1 \subset \cdots \subset \mathbf{GenEx}^i \subset \mathbf{GenEx}^{i+1} \subset \cdots \subset \mathbf{GenEx}^*]$.

Theorem 20 $\mathbf{Bc} - \mathbf{GenEx}^* \neq \emptyset$.

PROOF. Let $\mathcal{C} = \{f \in \mathcal{R} \mid (\overset{\infty}{\forall}\, x \in N)[\varphi_{f(x)} = f]\}$. It is easy to see that $\mathcal{C} \in \mathbf{Bc}$. $\mathcal{C} \notin \mathbf{GenEx}^*$ can be shown by a simple modification of the proof that $\mathcal{C} \notin \mathbf{Ex}^*$ in [CS83]. We omit the details. $\Box$

Theorem 21 $(\forall i)[\mathbf{GenEx}^{i+1} - \mathbf{Bc}^i \neq \emptyset]$.

PROOF.

Let $\mathcal{C} = \{f \in \mathcal{R} \mid f(0)$ is an $(i+1)$-generator for f $\}$.

It is easy to see that $\mathcal{C} \in \mathbf{GenEx}^{i+1}$. Suppose by way of contradiction that $\mathbf{M}$ $\mathbf{Bc}^i$-identifies $\mathcal{C}$. Then, by the operator recursion theorem [Cas74], there exists an 1–1, computable function q such that the (partial) functions $\varphi_{q(\cdot)}$ may be defined as follows.

Let x_s^0 denote the least x such that $\varphi_{q(0)}(x)$ has not been defined before stage s. Let x_s^1 denote the least x such that $\varphi_{q(1)}(x)$ has not been defined before stage s. Let $\varphi_{q(0)}(0) = q(1)$. Let $\varphi_{q(1)}(0) = q(0)$. Go to stage 2.

```
Begin stage $s$

1.  For $x < x_s^1$, let $\varphi_{q(s)}(x) = \varphi_{q(1)}(x)$.
2.  Let $y_s^0 = x_s^0$, $y_s^1 = x_s^1$.
    repeat
      2.1  Let $\varphi_{q(0)}(y_s^0) = q(s)$.
      2.2  Let $\varphi_{q(s)}(y_s^1) = 0$.
      2.3  Let $y_s^0 = y_s^0 + 1$.
      2.4  Let $y_s^1 = y_s^1 + 1$.
      2.5  If $(\exists m, z \mid x_s^1 \leq m \leq z < y_s^1)$, such that, for each $x \leq i$, $\Phi_{\mathbf{M}(\varphi_{q(s)}[m])}(z+x) \leq y_s^1$ and $\varphi_{\mathbf{M}(\varphi_{q(s)}[m])}(z+x) = 0$, then go to step 3.
    forever
3.  Let $z$ be as found in step 2.5.
      3.1  For each $x \in \{w \mid x_s^1 \leq w < y_s^1 \wedge w \notin \{z + r \mid r \leq i\}\}$, let $\varphi_{q(1)}(x) = 0$.
      3.2  For each $x \in \{z + r \mid r \leq i\}$, let $\varphi_{q(1)}(x) = 1$.
4.  For $x \geq y_s^1$, whenever $\varphi_{q(1)}(x)$ gets defined, let $\varphi_{q(s)}(x) = \varphi_{q(1)}(x)$ (i.e. $\varphi_{q(s)}$ "follows" $\varphi_{q(1)}$ from now on).
    (Note that this step ensures that $\varphi_{q(s)} =^{i+1} \varphi_{q(1)}$.)
5.  Go to stage $s + 1$.

End stage $s$
```

Now consider the following cases:

Case 1: Infinitely many stages are executed.

In this case, let $f = \varphi_{q(1)}$. Clearly, $\varphi_{q(1)} \in \mathcal{C}$ (since $\varphi_{q(0)}$ is an $(i{+}1)$ generator for $\varphi_{q(1)}$). However, because of the success of the condition in step 2.5 in each stage, and the diagonalization in step 3.2, for infinitely many m, $\varphi_{\mathbf{M}(\varphi_{q(1)}[m])} \neq^i \varphi_{q(1)}$. Thus $f \notin \mathbf{Bc}^i(\mathbf{M})$.

Case 2: Stage s starts but never finishes.

In this case, let $f = \varphi_{q(s)}$. Clearly, $f \in \mathcal{C}$. Moreover, for all but finitely many m, for infinitely many x, $\varphi_{\mathbf{M}(f[m])}(x) \neq 0$ (otherwise the test at step 2.5 would succeed). Thus $f \notin \mathbf{Bc}^i(\mathbf{M})$.

From the above cases it follows that $\mathcal{C} \notin \mathbf{Bc}^i$. $\square$

As a corollary to Theorem 21 and Theorem 3(c) we have

Corollary 22 $(\forall i \geq 1)[\mathbf{GenEx}^i - \mathbf{Ex}^* \neq \emptyset]$.

The following theorem follows as a corollary to Theorems 13(b) and 15(b).

Theorem 23 $(\forall i)[\mathbf{LimEx} - \mathbf{GenEx}^i \neq \emptyset]$.

The following theorems show some tradeoff results in learning generators *vis-à-vis* learning **Lim**-programs. Note that there is a gap between the diagonalization in Theorem 24 and simulation in Theorem 25. We leave it as an open question to find an exact tradeoff relationship between the different **GenEx** and **LimEx** learning criteria.

Theorem 24 $(\forall i \geq 1)[\mathbf{GenEx}^i - \mathbf{LimEx}^{\lfloor 3i/2 \rfloor - 1} \neq \emptyset]$.

The proof of this theorem is complicated and uses the operator recursion theorem [Cas74].

Proof.

Let $\mathcal{C} = \{f \in \mathcal{R} \mid f(0) \text{ is an } i\text{-generator for } f\ \}$.

It is easy to see that $\mathcal{C} \in \mathbf{GenEx}^i$. Suppose by way of contradiction that $\mathbf{M}$ $\mathbf{LimEx}^{\lfloor 3i/2 \rfloor - 1}$-identifies $\mathcal{C}$. Then, by the operator recursion theorem [Cas74], there exists an 1–1, computable function q such that the (partial) functions $\varphi_{q(\cdot)}$ may be described as follows.

By the Kleene s-m-n theorem, there exists a computable L such that for all p, x, t,

$$\varphi_{L(p)}(x,t) = \begin{cases} \varphi_p(x,t'), & (\exists t'' \leq t)[\Phi_p(x,t'') \leq t] \wedge \\ & t' = \max(\{t'' \leq t \mid \Phi_p(x,t'') \leq t\}); \\ 0, & \text{otherwise.} \end{cases}$$

Note that, for all p, $\varphi_{L(p)}$ is a total function, and $\varphi_p^\star \subseteq \varphi_{L(p)}^\star$.

Let x_s^0 denote the least x, such that $\varphi_{q(0)}(x)$ has not been defined before stage s. Let x_s^1 denote the least x, such that $\varphi_{q(1)}(x)$ has not been defined before stage s.

Let $\varphi_{q(0)}(0) = q(1)$. Let $\varphi_{q(1)}(0) = q(0)$. Go to stage 1.

```
Begin stage $s$

1.  For $x < x_s^1$, let $\varphi_{q(2s)}(x) = \varphi_{q(2s+1)}(x) = \varphi_{q(1)}(x)$.
    For all $z < i$, let $\varphi_{q(2s)}(x_s^1 + z) = 0$ and $\varphi_{q(2s+1)}(x_s^1 + z) = 1$.

2.  For $b_0, b_1, \ldots, b_{i-1} \in \{0,1\}$ and $y \in N$, define $f_{b_0 b_1 \ldots b_{i-1},y}$ as follows:
```

$$f_{b_0 b_1 \ldots b_{i-1},y}(x) = \begin{cases} \varphi_{q(1)}(x), & \text{if } x < x_s^1; \\ b_{x-x_s^1}, & \text{if } x_s^1 \le x \le x_s^1 + i - 1; \\ y, & \text{if } x_s^1 + i \le x < x_s^1 + \lfloor \frac{3i}{2} \rfloor; \\ 0, & \text{otherwise.} \end{cases}$$

```
3.  Let $y_s^0 = x_s^0$ and $y_s^1 = x_s^1 + \lfloor \frac{3i}{2} \rfloor$.

    repeat

    3.1  Let $\varphi_{q(2s)}(y_s^1) = \varphi_{q(2s+1)}(y_s^1) = 0$.
    3.2  Let $C_1 = \text{card}(\{x \mid x_s^1 \le x < x_s^1 + i \wedge \varphi_{L(\mathbf{M}(\varphi_{q(1)}[x_s^1]))}(x, y_s^1) = 1\})$.
    3.3  if $C_1 \ge \lceil \frac{i}{2} \rceil$ then
                 let $\varphi_{q(0)}(y_s^0) = q(2s)$.
         else
                 let $\varphi_{q(0)}(y_s^0) = q(2s+1)$.
         endif
    3.4  $y_s^0 = y_s^0 + 1$, $y_s^1 = y_s^1 + 1$.
    3.5  If there exist $b_0, b_1, \ldots, b_{i-1} \in \{0,1\}$ and $y \le y_s^1$ such that $\mathbf{M}(\varphi_{q(1)}[x_s^1]) \ne \mathbf{M}(f_{b_0 b_1 \ldots b_{i-1},y}[y_s^1])$,
         then go to step 4.

    forever

4.  Let $b_0, b_1, \ldots, b_{i-1}$, $y$ be as found in step 3.5.

    4.1  For $z < i$, let $\varphi_{q(1)}(x_s^1 + z) = b_z$.
    4.2  For $z < \lfloor \frac{i}{2} \rfloor$, let $\varphi_{q(2s)}(x_s^1 + i + z) = \varphi_{q(2s+1)}(x_s^1 + i + z) = \varphi_{q(1)}(x_s^1 + i + z) = y$.
    4.3  For $z$ such that $x_s^1 + \lfloor \frac{3i}{2} \rfloor \le z < y_s^1$, let $\varphi_{q(1)}(x_s^1 + i + z) = 0$.
    4.4  For $x \ge y_s^1$, whenever $\varphi_{q(1)}(x)$ gets defined, let $\varphi_{q(2s)}(x) = \varphi_{q(2s+1)}(x) = \varphi_{q(1)}(x)$ (i.e. $\varphi_{q(2s)}$ and
         $\varphi_{q(2s+1)}(x)$ "follow" $\varphi_{q(1)}$ from now on; also, because of step 4, $\varphi_{q(2s)} =^i \varphi_{q(1)}$ and
         $\varphi_{q(2s+1)} =^i \varphi_{q(1)}$).
```

5. Go to stage $s+1$.

End stage s

Now consider the following cases.

Case 1: Infinitely many stages are executed.

In this case, let $f = \varphi_{q(1)}$. Clearly, $f \in \mathcal{C}$ (since $\varphi_{q(0)}$ is a i generator for f). However, $\mathbf{M}(f)\uparrow$ (since, the only way infinitely many stages can be executed is the success of the condition at step 3.5 in each stage s). Thus $f \notin \mathbf{LimEx}^{\lfloor 3i/2 \rfloor - 1}$.

Case 2: Stage s starts but never finishes.

For all $b_0, b_1, \ldots, b_{i-1} \in \{0,1\}$ and $y \in N$, let $f_{b_0 b_1 \ldots b_{i-1},y}$ be as defined in step 2 of stage s. Note that in this case, for all $b_0, b_1, \ldots, b_{i-1} \in \{0,1\}$ and $y \in N$, $\mathbf{M}(f_{b_0 b_1 \ldots b_{i-1},y}) = \mathbf{M}(\varphi_{q(1)}[x_s^1])$.

Let $Conv_1 = \{x \mid [x_s^1 \leq x < x_s^1 + i] \bigwedge [\varphi^{\star}_{\mathbf{M}(\varphi_{q(1)}[x_s^1])}(x)\downarrow = 1]\}$.

Let $Convnon_1 = \{x \mid [x_s^1 \leq x < x_s^1 + i] \bigwedge [\varphi^{\star}_{\mathbf{M}(\varphi_{q(1)}[x_s^1])}(x)\downarrow \neq 1]\}$.

Let y be such that, for all $z \in \{z' \mid x_s^1 + i \leq z' < x_s^1 + \lfloor \frac{3i}{2} \rfloor\}$, $\varphi^{\star}_{\mathbf{M}(\varphi_{q(1)}[x_s^1])}(z) \neq y$ (note that there exists such a y).

Case 2.1: $\mathrm{card}(Conv_1) \geq \lceil i/2 \rceil$.

Note that in this case $(\overset{\infty}{\forall} x)[\varphi_{q(0)}(x) = q(2s)]$. For $k < i$, if $x_s^1 + k \in Conv_1$, then let $b_k = 0$; else, let $b_k = 1$. Note that $f_{b_0 b_1 \ldots b_{i-1},y} \neq^{\lfloor 3i/2 \rfloor - 1} \varphi^{\star}_{\mathbf{M}(\varphi_{q(1)}[x_s^1])}$. Since $(i - \mathrm{card}(Conv_1)) + \lfloor \frac{i}{2} \rfloor \leq i$, $f_{b_0 b_1 \ldots b_{i-1},y} =^{i} \varphi_{q(2s)}$. Thus $f_{b_0 b_1 \ldots b_{i-1},y} \in \mathcal{C} - \mathbf{LimEx}^{\lfloor 3i/2 \rfloor - 1}(\mathbf{M})$.

Case 2.2: $\mathrm{card}(Convnon_1) \geq \lceil i/2 \rceil$.

Note that in this case $(\overset{\infty}{\forall} x)[\varphi_{q(0)}(x) = q(2s+1)]$. For $k < i$, if $x_s^1 + k \in Convnon_1$, then let $b_k = 1$; else, let $b_k = 0$. Note that $f_{b_0 b_1 \ldots b_{i-1},y} \neq^{\lfloor 3i/2 \rfloor - 1} \varphi^{\star}_{\mathbf{M}(\varphi_{q(1)}[x_s^1])}$. Since $(i - \mathrm{card}(Convnon_1)) + \lfloor \frac{i}{2} \rfloor \leq i$, $f_{b_0 b_1 \ldots b_{i-1},y} =^{i} \varphi_{q(2s+1)}$. Thus $f_{b_0 b_1 \ldots b_{i-1},y} \in \mathcal{C} - \mathbf{LimEx}^{\lfloor 3i/2 \rfloor - 1}(\mathbf{M})$.

Case 2.3: $\mathrm{card}(Conv_1) < \lceil i/2 \rceil$ and $\mathrm{card}(Convnon_1) < \lceil i/2 \rceil$.

Let $S \subseteq \{x_s^1 + z \mid z < i\}$ be a set of cardinality $\lceil i/2 \rceil$ such that $Convnon_1 \subseteq S$ and $S \cap Conv_1 = \emptyset$ (note that there exists such an S). Note that in this case $(\overset{\infty}{\forall} x)[\varphi_{q(0)}(x) \in \{q(2s), q(2s+1)\}]$. For $k < i$, if $x_s^1 + k \in S$, then let $b_k = 1$; else, let $b_k = 0$. Note that

$f_{b_0 b_1 \ldots b_{i-1}, y} \neq^{\lfloor 3i/2 \rfloor - 1} \varphi^{\star}_{\mathbf{M}(\varphi_{q(1)}[x^1_s])}$. Since $\mathrm{card}(S) + \lfloor \frac{i}{2} \rfloor \leq i$, $f_{b_0 b_1 \ldots b_{i-1}, y} =^i \varphi_{q(2s)}$. Also, since $(i - \mathrm{card}(S)) + \lfloor \frac{i}{2} \rfloor \leq i$, $f_{b_0 b_1 \ldots b_{i-1}, y} =^i \varphi_{q(2s+1)}$. Thus $f_{b_0 b_1 \ldots b_{i-1}, y} \in \mathcal{C} - \mathbf{LimEx}^{\lfloor 3i/2 \rfloor - 1}(\mathbf{M})$.

From the above cases it follows that $\mathcal{C} \notin \mathbf{LimEx}^{\lfloor 3i/2 \rfloor - 1}$. □

Theorem 25 $(\forall i)[\mathbf{GenEx}^i \subset \mathbf{LimEx}^{2i}]$.

Proof. We show that $\mathbf{GenEx}^i \subseteq \mathbf{LimEx}^{2i}$. Proper containment will then follow using Theorem 23.

Let F be the (partial) function from triplets of natural numbers to sequences defined as follows.

$F(p, r, m)$ = the lexicographically least sequence σ of length $m + 1$, if such a sequence exists, which satisfies (2).

$$(\forall j \mid r \leq j < m)[\mathrm{card}(\{x \mid x \leq m \wedge \Phi_p(j) \leq m \wedge \Phi_{\varphi_p(j)}(x) \leq m \wedge \varphi_{\varphi_p(j)}(x) \neq \sigma(x)\}) \leq i]; \qquad (2)$$

$F(p, r, m) =\uparrow$ if no σ of length $m + 1$ satisfies (2).

By the Kleene's s-m-n theorem [Rog67] there exists a computable g such that for all p, x, r, t

$$\varphi_{g(p,r)}(x, t) = (F(p, r, t + x))(x). \qquad (3)$$

Let $\mathrm{Err}(f, n, p) = \max(\{j + 1 \mid \mathrm{card}(\{x < n \mid \Phi_p(j) \leq n \wedge \Phi_{\varphi_p(j)}(x) \leq n \wedge \varphi_{\varphi_p(j)}(x) \neq f(x)\}) > i\})$.

Given $\mathbf{M}$, define $\mathbf{M}'$ as follows. $\mathbf{M}'(f[n]) = g(\mathbf{M}(f[n]), \mathrm{Err}(f, n, \mathbf{M}(f[n])))$.

It is easy to see that $\mathbf{GenEx}^i(\mathbf{M}) \subseteq \mathbf{LimEx}^{2i}(\mathbf{M}')$. □

Theorem 26 $\mathbf{LimEx}^* \subseteq \mathbf{GenEx}^*$.

Proof. We will describe a computable function g such that, for all p, $g(p)$ is such that

$$\varphi_{g(p)} \text{ is total and } (\forall f \in \mathcal{R})[\varphi^{\star}_p =^* f \Rightarrow g(p) \text{ is a } *\text{-generator for } f]. \qquad (4)$$

Given, $\mathbf{M}$ define $\mathbf{M}'(f[n]) = g(\mathbf{M}(f[n]))$. It is easy to see that $\mathbf{LimEx}^*(\mathbf{M}) \subseteq \mathbf{GenEx}^*(\mathbf{M}')$.

Now we show how to obtain the g as claimed above.

By Kleene's s-m-n theorem there exist computable h, prog, g such that the following three conditions hold.

(a) For all p, x, t,

$$\varphi_{h(p)}(x,t) = \begin{cases} \varphi_p(x,t'), & (\exists t'' \leq t)[\Phi_p(x,t'') \leq t] \wedge \\ & t' = \max(\{t'' \leq t \mid \Phi_p(x,t'') \leq t\}); \\ 0, & \text{otherwise.} \end{cases}$$

Note that for all p, $\varphi_{h(p)}$ is a total function. Moreover, $\varphi_p^\star \subseteq \varphi_{h(p)}^\star$.

(b) For all p, j, y,

$$\text{prog}(p,j,y) = \min(\{j\} \cup \{i < j \mid \text{card}(\{x \leq j \mid \Phi_i(x) > y \vee \varphi_i(x) \neq \varphi_{h(p)}(x,y)\}) \leq i\}). \quad (5)$$

(c) For all p, j, y, $[\varphi_{g(p)}$ is total $\wedge\, \varphi_{\varphi_{g(p)}(j)}(y) = \varphi_{\text{prog}(p,j,y)}(y)]$.

Suppose, p is such that for some $f \in \mathcal{R}$, $\varphi_p^\star =^* f$. Let $\text{finerr}(p) = \min(\{i \mid \text{card}(\{x \mid \varphi_p^\star(x)\uparrow \vee \varphi_i(x) \neq \varphi_p^\star(x)\}) \leq i\})$. It is easy to see that $\varphi_{\text{finerr}(p)} =^* f$. Moreover,

$$(\forall f \in \mathcal{R})(\forall p \mid \varphi_p^\star =^* f)(\overset{\infty}{\forall}\, j)(\overset{\infty}{\forall}\, y)[\text{prog}(p,j,y) \leq \text{finerr}(p) \wedge \varphi_{\text{prog}(p,j,y)} =^* f]. \quad (6)$$

It follows that,

$$(\forall f \in \mathcal{R})(\forall p \mid \varphi_p^\star =^* f)(\overset{\infty}{\forall}\, j)(\overset{\infty}{\forall}\, y)[\varphi_{\text{prog}(p,j,y)}(y) = f(y)]. \quad (7)$$

It follows from (7) and the definition of g, that, g satisfies (4). $\Box$

4 Open Problems

Note that there is a gap between the diagonalization in Theorem 24 and simulation in Theorem 25. We leave it as an open question to find an exact tradeoff relationship between the different **GenEx** and **LimEx** learning criteria. It is also open at present whether $\mathbf{GenEx}^* = \mathbf{LimEx}^*$ (note that by Theorem 26 $\mathbf{LimEx}^* \subseteq \mathbf{GenEx}^*$).

References

[AS83] D. Angluin and C. Smith. A survey of inductive inference: Theory and methods. *Computing Surveys*, 15:237–289, 1983.

[Bar74] J. M. Barzdin. Two theorems on the limiting synthesis of functions. *In Theory of Algorithms and Programs, Latvian State University, Riga*, 210:82–88, 1974. In Russian.

[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.

[Cas74] J. Case. Periodicity in generations of automata. *Mathematical Systems Theory*, 8:15–32, 1974.

[Cas86] J. Case. Learning machines. In W. Demopoulos and A. Marras, editors, *Language Learning and Concept Acquisition*. Ablex Publishing Company, 1986.

[Che81] K. Chen. *Tradeoffs in Machine Inductive Inference*. PhD thesis, SUNY at Buffalo, 1981.

[CJS92] J. Case, S. Jain, and A. Sharma. On learning limiting programs. *International Journal of Foundations of Computer Science*, 3(1):93–115, 1992.

[CL82] J. Case and C. Lynes. Machine inductive inference and language identification. In M. Nielsen and E. M. Schmidt, editors, *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*, volume 140, pages 107–115. Springer-Verlag, Berlin, 1982.

[Cra53] W. Craig. On axiomatizability within a system. *Journal of Symbolic Logic*, 18:30–32, 1953.

[CS83] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.

[Fef60] S. Feferman. Arithmetization of metamathematics in a general setting. *Fundamenta Mathematicae*, 49:35–92, 1960.

[Fis65] P. Fischer. Theory of provable recursive functions. *Transactions of the American Mathematical Society*, 117:494–520, 1965.

[Ful85] M. Fulk. *A Study of Inductive Inference machines*. PhD thesis, SUNY at Buffalo, 1985.

[Gol65] E. M. Gold. Limiting recursion. *Journal of Symbolic Logic*, 30:28–48, 1965.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory Languages and Computation*. Addison-Wesley Publishing Company, 1979.

[Kre51] G. Kreisel. On the interpretation of non-finitist proofs. *Journal of Symbolic Logic*, 17:241–267, 1951.

[Kre58] G. Kreisel. Mathematical significance of consistency proofs. *Journal of Symbolic Logic*, 23:155–182, 1958.

[KW80] R. Klette and R. Wiehagen. Research in the theory of inductive inference by GDR mathematicians – A survey. *Information Sciences*, 22:149–169, 1980.

[LMF76] N. Lynch, A. Meyer, and M. Fischer. Relativization of the theory of computational complexity. *Transactions of the American Mathematical Society*, 220:243–287, 1976.

[Men86] E. Mendelson. *Introduction to Mathematical Logic*. Brooks-Cole, San Francisco, 1986. 3rd Edition.

[MY78] M. Machtey and P. Young. *An Introduction to the General Theory of Algorithms*. North Holland, New York, 1978.

[OSW86] 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.

[Put65] H. Putnam. Trial and error predicates and the solution to a problem of Mostowski. *Journal of Symbolic Logic*, 30:49–57, 1965.

[RC92] J. Royer and J. Case. *Intensional Subrecursion and Complexity Theory*. Research Notes in Theoretical Science. Pitman Press, being revised for expected publication, 1992.

[Rog57] H. Rogers. Provably recursive functions. *Bulletin of the American Mathematical Society*, 63:140, 1957.

[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.

[Ros84] H. Rose. *Subrecursion: Functions and Hierarchies.* Oxford University Press, 1984.

[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.

[Sho59] J. Shoenfield. On degrees of unsolvability. *Annals of Mathematics*, 69:644–653, 1959.

[Sho71] J. Shoenfield. *Degrees of Unsolvability.* North-Holland, 1971.

[Soa87] R. Soare. *Recursively Enumerable Sets and Degrees.* Springer-Verlag, 1987.

[Wie78] R. Wiehagen. *Zur Theorie der Algorithmischen Erkennung.* PhD thesis, Humboldt-Universitat, Berlin, 1978.