Resource Restricted Computability Theoretic Learning: Illustrative Topics and Problems

John Case*

Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716-2586 U.S.A.
case@cis.udel.edu

Abstract. Computability theoretic learning theory (machine inductive inference) typically involves learning programs for languages or functions from a stream of complete data about them and, importantly, allows mind changes as to conjectured programs. This theory takes into account algorithmicity but typically does *not* take into account *feasibility* of computational resources. This paper provides some example results and problems for three ways this theory can be constrained by computational feasibility. Considered are: the learner has memory limitations, the learned programs are desired to be optimal, and there are feasibility constraints on obtaining each output program *and* on the number of mind changes.

1 Introduction and Motivation

Let $\mathbb{N}$ = the set of non-negative integers. Computability theoretic (a.k.a recursion theoretic) learning [28,36] typically involves a scenario as depicted in (1) just below.

$$\text{Data } d_0, d_1, d_2, \ldots \xrightarrow{\text{In}} M \xrightarrow{\text{Out}} \text{ Programs } p_0, p_1, p_2, \ldots \ . \tag{1}$$

In (1), $d_0, d_1, d_2, \ldots$ can be, for example, the elements of a (formal) language $L \subseteq \mathbb{N}$ or the successive values of a function $f : \mathbb{N} \to \mathbb{N}$; M is a machine; and, for its *successful* learning, later p_i's $\approx$ compute the L or f. We will consider different criteria of successful learning of L or f by M. **Ex**-style criteria require that all but finitely many of the p_i's are the same *and* do a good job of computing the L or f. **Bc**-style criteria are more relaxed and powerful [4,12,15] and do not require almost all p_i's be the same.

In the present paper we *survey* some *illustrative*, top down, computational *resource restrictions* on essentially the paradigm of (1). In some sections we work instead with simple variants of (1).

In Section 2 below, $d_0, d_1, d_2, \ldots$ are the elements of a language and the p_i's are usually type-0 grammars [26] (equivalently, r.e. or c.e. indices [39]) for languages.

* Work supported in part by NSF Grant Number CCR-0208616 at UD.

S.B. Cooper, B. Löwe, and A. Sorbi (Eds.): CiE 2007, LNCS 4497, pp. 115–124, 2007.

In Section 2 we consider restrictions on the ability of machines M to remember prior data and conjectures. The motivation, in addition to space-complexity theoretic, is from cognitive science. We will provide some open problems too. Much of the material of Section 2 is from [8,13].

In Section 3 below, the d_i's are successive values of functions f in some complexity theoretically interesting subrecursive classes and the p_i's can be programs in some associated subrecursive programming systems [40]. A not-necessarily-realized *desire* is that successful later programs p_i are not too far from optimally efficient.[1] We provide some examples (from [10]) showing how computational complexity of those later p_i's, which *are* successful at computing the f's, is affected by the size of the subrecursive class to be learned.

The paradigm of (1) is sometimes called *learning in the limit*, and the M in (1) can be thought of as computing an appropriately typed, *limiting* functional. One speaks of M's transitions from outputting p_i to outputting p_{i+1} as *mind changes*.[2] Restrictions on the number of such mind changes have been extensively studied in the literature, beginning with [5,15]. [21] first considered counting down mind changes from notations for possibly transfinite constructive ordinals and proved results to the effect that counting down from bigger constructive ordinals gave more learning power. See also [1,29].[3] In (1), the *time* to calculate each p_i may be infeasible, and the total time to reach the *successful* p_i's may not be algorithmic [17]. In Section 4 below, we present some previously unpublished *very preliminary* work on top down, *feasible* variants of (1), and we indicate problems we hope will be worked out in the future [14]. The rough idea, explained in more detail in Section 4 below, is that: 1. one restricts the M of (1) to iterating a *type-2 feasible functional* in the sense of [27,30,35], *and* 2. one counts down, with another type-2 *feasible* functional, the allowed mind changes from *feasible* notations for constructive ordinals.[4]

2 Memory-Limited and U-Shaped Language Learning

Informally, *U-shaped learning* is as follows. For B a task to learn a desired *behavior*, U-shaped learning occurs when, while learning B, a learner *learns* B, then the learner *un*learns B, and, finally, the the learner *re*learns B. U-shaped learning has been empirically observed by cognitive psychologists in various areas of child development. Examples include understanding of various (Piaget-like)

[1] This will be for cases, unlike in Blum Speed-Up Theorems [36], where there *are* optimally efficient programs.

[2] Learning in the limit is essential, for example, for the iterated forward difference method for fitting polynomials to data [25], where the number of mind changes required depends on the degree of the polynomial generating the data.

[3] Outside computability theoretic learning, [2] characterizes *explicitly* Ershov Hierarchy levels [20,19] by constructive ordinal notation *count down*.

[4] For example, algorithmic counting down mind-changes from any notation w for the smallest infinite ordinal ω is equivalent to declaring if and when a first mind change is made *and then* declaring the *finite* number of *further* mind changes allowed.

conservation principles [42], e.g., the interaction between object tracking and object permanence, and verb regularization [34,37,42,43]. Here is an example of the latter. In English, a child first uses *spoke*, the correct past tense form of the verb *speak*. Then the child *in*correctly uses speaked. Lastly, the child returns to using *spoke*.

One main question of the present section: is U-shaped learning *necessary* for full learning power? I.e., are there classes of tasks learnable *only* by *returning* to *abandoned, correct behavior*?

For example, [3,7] answered formalized versions of the previous question for computability theoretic learning with*out* memory limitations. The answer depends interestingly on the criteria of successful learning: roughly, for criteria of power strictly between **Ex** and **Bc**-styles [9] (and also for **Bc**-style), U-shaped learning *is* necessary for full learning power. Humans have memory limitations, both for previously seen data and, to some extent, for previously made conjectures. In the present section we discuss the necessity of U-shaped learning of grammars for whole formal languages *and* in models with such memory restrictions. *First*, though, we discuss in detail the cases with*out* memory limitations.

A sequence T of natural numbers and #'s is a *text* for a language $L \subseteq \mathbb{N} \Leftrightarrow_{\mathrm{def}}$ $L = \{T(i) \mid T(i) \neq \#\}$. The # represents a pause. The only text for the empty language is an infinite sequence of #'s. The present section employs a variant of (1) *where* d_i is $T(i)$ with T a text for a language, and the p_i's are either r.e. indices *or they are* ?*'s*. ?'s signal that M has no program to conjecture.

In the rest of the present section we restrict our attention to **Ex**-style criteria of success.

Formally: a learner M **TxtEx**-*learns* a *class* of languages $\mathcal{L} \Leftrightarrow_{\mathrm{def}}$, for all $L \in \mathcal{L}$, *on* all texts T for L, M eventually stabilizes to outputting *a single* program successfully generating L.

For a language class learning criterion, **C**-learning, such as **TxtEx**-learning just defined and variants defined below, we write **C** to stand for the collection of all languages classes $\mathcal{L}$ such that some machine M **C**-learns each $L \in \mathcal{L}$.

A learner M is *Non-U-Shaped* (abbreviated: **NU**) on a class $\mathcal{L}$ that M **TxtEx**-learns $\Leftrightarrow_{\mathrm{def}}$, on any text for any language L in $\mathcal{L}$, M never outputs a sequence $\ldots, p, \ldots, q, \ldots, r, \ldots$, where p, r accept/generate L, but q doesn't.

Next we discuss three types of *memory limited* language learning models from the prior literature [11,22,33,44].

Iterative Learning: M **It**-*learns* a *class* of languages $\mathcal{L} \Leftrightarrow_{\mathrm{def}}$ M **TxtEx**-learns $\mathcal{L}$ *but* M has access only to its own just previous conjecture (if any) and to its current text datum.

m-Feedback Learning is like **It**-learning, *but*, while the learner has access to its just previous conjecture and to the current text datum, it can also make m simultaneous *recall* queries, i.e., queries as to whether up to m items of its choice have already been seen in input data thus far.

n-Bounded Example Memory Learning is also like **It**-learning, *but*, while the learner has access to its just previous conjecture and to the current text datum, it remembers up to n previously seen data items that it chooses to remember.

For $m, n > 0$, m-Feedback and n-Bounded Example Learning are incomparable, but separately make strict learning hierarchies increasing in m, n [11].

N.B. For the cases of $m, n > 0$, it is completely open as to whether U-shapes are necessary for full power of m-Feedback and n-Bounded Example Learning!

Results about the necessity of U-shapes for **It**-learning and for *more severely restricted* variants of the other models just above *have* been obtained and some are discussed below. For **It**-learning, U-shapes are *not* needed:

Theorem 1 ([13]). NUIt = It.[5]

Memoryless Feedback Learners are restricted versions of Feedback Learners above: for $m > 0$, an $\mathbf{MLF}_m$*-learner* has *no* memory of previous conjectures but has access to its current text datum and can make $m > 0$ simultaneous recall queries — queries as to whether up to m items of its choice have already been been seen in input data. The $\mathbf{MLF}_m$ learning criteria form a hierarchy increasing in m: $\mathbf{MLF}_m \subset \mathbf{MLF}_{m+1}$ [8].

Theorem 2 ([8]). U-shaped learning is necessary for each level of this hierarchy: for $m > 0$, $\mathbf{NUMLF}_m \subset \mathbf{MLF}_m$.

Bounded Memory States Learners [32] are restricted variants of Bounded Example Learners above: for $c > 1$, a $\mathbf{BMS}_c$-Learner does *not* remember any previous conjectures, has access to current text datum, *and* can store *any one of* c different values it chooses in its memory. This latter corresponds exactly to remembering $\log_2(c)$ bits. The $\mathbf{BMS}_c$ learning criteria form a hierarchy increasing in $c > 1$: $\mathbf{BMS}_c \subset \mathbf{BMS}_{c+1}$ [32].

Theorem 3 ([8]). U-shaped learning is *not needed* for 2-Bounded Memory States Learners: $\mathbf{NUBMS}_2 = \mathbf{BMS}_2$.

Open Questions: is U-shaped learning necessary for $\mathbf{BMS}_c$*-learning with $c > 2$?* Humans remember some bits, remember some prior data, can recall whether they've seen some data, and, likely, store their just prior conjecture. *Is U-shaped learning necessary for such combinations?*

3 Complexity of Learned Programs

Results in the present section are selected from many in [10], and we employ (1) in the case that d_i is $f(i)$, *where* $f \in \mathcal{F} \subseteq \mathcal{R}_{0,1}$, the class of all (total) computable functions : $\mathbb{N} \rightarrow \{0, 1\}$.

Suppose $a \in \mathbb{N} \cup \{*\}$. a is for anomaly count. When $a = *$, a stands for finitely many.

$\mathcal{F} \in \mathbf{Ex}^a \Leftrightarrow_{\mathrm{def}} (\exists M)(\forall f \in \mathcal{F})$
$[M$ fed $f(0), f(1), \ldots,$ outputs $p_0, p_1, \ldots \wedge (\exists t)[p_t = p_{t+1} = \cdots \wedge p_t$ computes f — except at up to a inputs]].

[5] As per our convention above, **It**, respectively **NUIt**, stands for the collection of all languages classes $\mathcal{L}$ such that some machine M **It**-learns, respectively **NUIt**-learns, each $L \in \mathcal{L}$.

$\mathcal{F} \in \mathbf{Bc}^a \Leftrightarrow_{\mathrm{def}} (\exists M)(\forall f \in \mathcal{F})$
$[M$ fed $f(0), f(1), \ldots,$ outputs $p_0, p_1, \ldots \wedge (\exists t)[p_t, p_{t+1}, \ldots$ each computes f — except at up to a inputs]].

Turing machines herein are multi-tape.

For $k \geq 1$, $\mathcal{P}^k =_{\mathrm{def}}$ the class of all $\{0,1\}$-valued functions computable by Turing Machines in $O(n^k)$ time, where n is the length of the input expressed in *dyadic* notation.[6] $\mathcal{P} =_{\mathrm{def}} \bigcup \mathcal{P}^k$.

Let slow be any fixed slow growing unbounded function $\in \mathcal{P}^1$, e.g., $\leq$ an inverse of Ackermann's function as from [16, Section 21.4]. $\mathcal{Q}^k =_{\mathrm{def}}$ the class of all $\{0,1\}$-valued functions computable in $O(n^k \cdot \log(n) \cdot \mathrm{slow}(n))$ time. $\mathcal{P}^k \subset \mathcal{Q}^k \subset \mathcal{P}^{k+1}$. The first proper inclusion is essentially from [24,26] and appears to be best known.

$\mathcal{P} \in \mathbf{Ex}^0$. $\mathcal{P}^k \in \mathbf{Ex}^0$ too (where each output conjecture runs in k-degree polytime).

$\mathcal{CF}$, the class of all characteristic functions of the context free languages [26], $\in \mathbf{Ex}^0$ [23].

From [15] (with various credits to Bārzdiņš, the Blums, Steel, and Harrington): $\mathbf{Ex}^0 \subset \mathbf{Ex}^1 \subset \mathbf{Ex}^2 \subset \cdots \subset \mathbf{Ex}^* \subset \mathbf{Bc}^0 \subset \mathbf{Bc}^1 \subset \cdots \subset \mathbf{Bc}^*$, and $\mathcal{R}_{0,1} \in \mathbf{Bc}^*$.

We introduce some basic, useful notation.

$(\forall^\infty x)$ means for all but finitely many $x \in \mathbb{N}$.

$\mathcal{U} =_{\mathrm{def}} \{f \in \mathcal{R}_{0,1} \mid (\forall^\infty x)[f(x) = 1]\}$ $(\subset \mathcal{P}^1)$. $\mathcal{U}$ is an example of a class of particularly *easy* functions.

$\varphi_p^{\mathrm{TM}} =_{\mathrm{def}}$ the partial computable function : $\mathbb{N} \to \mathbb{N}$ computed by Turing machine program (number) p.

$\Phi_p^{\mathrm{TM}}(x) =_{\mathrm{def}}$ the runtime of Turing machine program (number) p on input x, if p halts on x, and undefined, otherwise.

$\Phi_p^{\mathrm{WS}}(x) =_{\mathrm{def}}$ the work space used by Turing machine program (number) p on input x, if p halts on x, and undefined, otherwise.

Clearly, $\mathcal{U} \subset \mathcal{REG}$, the class all characteristic functions of regular languages [26].

$f[n] =_{\mathrm{def}}$ the sequence $f(0), \ldots, f(n-1)$.

$M(f[n]) =_{\mathrm{def}}$ M's output based only on $f[n]$.

Of course, since finite automata do not employ a work tape, $\exists M$ witnessing $\mathcal{REG} \in \mathbf{Ex}^0$ such that $(\forall n, x)[\Phi_{M(f[n])}^{\mathrm{TM}}(x) = |x| + 1 \ \wedge \ \Phi_{M(f[n])}^{\mathrm{WS}}(x) = 0]$.

A result of [41] is strengthened by

Theorem 4 ([10]). Suppose $k \geq 1$ and that M witnesses either $\mathcal{Q}^k \in \mathbf{Ex}^*$ or $\mathcal{Q}^k \in \mathbf{Bc}^0$ (special case: M witnesses $\mathcal{Q}^k \in \mathbf{Ex}^0$).
Then: $(\exists f \in \mathcal{U})(\forall k$-degree polynomials $p)$
$(\forall^\infty n)(\forall^\infty x)[\Phi_{M(f[n])}^{\mathrm{TM}}(x) > p(|x|)]$.

If we increase the generality of a machine M to handle $\mathcal{Q}^k$ instead of merely $\mathcal{P}^k$, this *forces* the run-times of M's successful outputs on some *easy* $f \in \mathcal{U}$ worse

[6] The *dyadic* representation of an input natural number $x =_{\mathrm{def}}$ the x-th finite string over $\{0,1\}$ *in lexicographical order*, where the counting of strings starts with zero [40]. Hence, unlike with binary representation, lead zeros matter.

than any k-degree polynomial bound, i.e., to be *sub*optimal. But, for learning only $\mathcal{P}^k$, this need not happen. Hence, we see, in Theorem 4, ones adding slightly to the generality of a learner M produces final, successful programs with a complexity deficiency. Another complexity deficiency in final programs caused by learning too much is provided by the following

Theorem 5 ([10]). Suppose M $\mathbf{Ex}^*$-learns $\mathcal{CF}$ and $k, n \geq 1$(special case: M witnesses $\mathcal{CF} \in \mathbf{Ex}^0$). Then there is an easy f, an $f \in \mathcal{U}$, such that, if p is M's final program on f, for *some distinct* $x_0, \ldots, x_{n-1}$, program p uses more than k workspace squares on each of inputs $x_0, \ldots, x_{n-1}$.

In [10] there are further such complexity deficiencies in final programs caused by learning too much, again where the deficiencies are on *easy* functions. *Assuming $\mathcal{NP}$ separates* from $\mathcal{P}$ (with $\mathcal{NP}$ also treated as a class of $\{0, 1\}$-valued characteristic functions of sets $\subseteq \mathbb{N}$), then one gets a complexity deficiency in final programs caused by learning $\mathcal{NP}$ instead of $\mathcal{P}$. There is a similar result in [10] for $\mathcal{BQP}$, a quantum version of polynomial-time [6], in place of $\mathcal{NP}$ — *assuming $\mathcal{BQP}$ separates* from $\mathcal{P}$. In these results the complexity deficient learned programs have *un*necessary non-determinism or quantum parallelism.

4 Feasible Iteration of Feasible Learning Functionals

The material of this section not credited to someone else is from [14].

One-shot **Ex**-style procedures output at most a single (hopefully correct) conjectured program [28]. *Feasible* deterministic one-shot function learning can be modeled by the polytime multi-tape Oracle Turing machines (OTMs) as used in [27] (see also [30,35]). We call the corresponding functionals *basic feasible functionals*. These polytime OTMs have a query tape and a reply tape. To query an oracle f, an OTM writes the dyadic representation of an $x \in \mathbb{N}$ on the query tape and enters its query state. The query tape is then erased, and the dyadic representation of $f(x)$ appears on the reply tape. The cost model is the same as for non-oracle Turing machines, *except* for the additional cost of a query to the oracle. This is handled with the length-cost model, where the cost of a query is $max(|f(x)|, 1)$, where $|f(x)|$ is the length of the string on the reply tape.[7] The next three definitions provide the formal details re the polytime constraint on basic feasible functionals.

Definition 1 ([30]). The *length* of $f : \mathbb{N} \to \mathbb{N}$ is the function $|f| : \mathbb{N} \to \mathbb{N}$ such that $|f| = \lambda n.max(\{|f(x)| \mid |x| \leq n\})$.

Definition 2 ([30]). A *second-order polynomial* over type-1 variables $g_0, ..., g_m$ and type-0 variables $y_0, ..., y_n$ is an expression of one of the following five forms:

[7] N.B. For the present section, then, the general paradigm (1) from Section 1 above is modified to allow the machine M to *query* input functions f for their values — instead of M's merely passively receiving the successive values of such f's.

$$a$$
$$y_i$$
$$\mathbf{q}_1 + \mathbf{q}_2$$
$$\mathbf{q}_1 \cdot \mathbf{q}_2$$
$$g_j(\mathbf{q}_1)$$

where $a \in \mathbb{N}$, $i \leq n$, $j \leq m$, and $\mathbf{q}_1$ and $\mathbf{q}_2$ are second-order polynomials over $\overrightarrow{g}$ and $\overrightarrow{y}$.

Definition 3 ([30]). Suppose $k \geq 1$ and $l \geq 0$. Then $F : (\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^l \to \mathbb{N}$ is a *basic feasible functional* if and only if there is an OTM $\mathbf{M}$ and a second-order polynomial $\mathbf{q}$, such that, for each input $(f_1, ..., f_k, x_1, ..., x_l)$,

(1) $\mathbf{M}$ outputs $F(f_1, ..., f_k, x_1, ..., x_l)$, and
(2) $\mathbf{M}$ runs within $\mathbf{q}(|f_1|, ..., |f_k|, |x_1|, ..., |x_l|)$ time steps.

In the context of learning in the limit, we are interested in how to define *feasible* for *limiting*-computable type-2 functionals. This is discussed below, but, first, is presented some background material on notations for constructive ordinals.

Ordinals are representations of well-orderings. The *constructive ordinals* are just those that have a program, called a *notation*, which specifies how to build them (lay them out end to end, so to speak) [39]. Let O be Kleene's system of notations for each constructive ordinal [39], importantly, with the accompanying $<_o$ relation on O. We omit details but refer the reader to the excellent [39].

Everyone knows how to use (notations for) finite ordinals for counting *down*.

As indicated in Section 1 above, we have in mind *iterating* basic feasible learning functionals with *feasible* counting down of iterations from *feasible* notations for constructive ordinals, We want to see worked out the details of this model of feasible for **Ex** learning. We believe we have a correct formalization of the concept of feasible notations and feasible counting down. From space limitations, below we present *very* simple examples only.

Here is a promised very simple example. It is based on a system of notations we call O_ω. O_ω provides notations for all and only the finite ordinals and ω, the first infinite ordinal. This restricted system especially reduces the complexity of computing notations. $O_\omega =_{\text{def}} \mathbb{N}$. For $u \in O_\omega$, if u is even, u is a notation for the (finite) ordinal $u/2$. Otherwise, u is a notation for the (infinite) ordinal ω. For even $x, y \in O_\omega$, $x < y \Rightarrow x <_{O_\omega} y$, and for any even x and odd y, $x <_{O_\omega} y$. No other pairs of numbers satisfy the $<_{O_\omega}$ relation. For $x \in \mathbb{N}$, $\underline{x}$ is defined as the notation for the *finite* ordinal x, and, in this system, $\underline{x} = 2x$.

Next we present one *of many ways* to define feasibly iterated feasible learning. We are interested to investigate more ways and are currently pursuing this. For $t \in \mathbb{N}$, 0^t is (by definition) the string of 0's of length t. It is common in complexity theory to call 0^t a *tally*. We write ε for 0^t when $t = 0$.

Below φ is acceptable [39] *and* has a linear time implementation of S-m-n [40].

Definition 4. Suppose $u \in O_\omega$. A set of functions $\mathcal{S}$ is $\mathbf{Itr}_u$-*feasibly learnable* if and only if there exist basic feasible functionals $H : (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ and $\mathbf{F} : (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ such that for all $f \in \mathcal{S}$, there exists $k \in \mathbb{N}$ such that,

(1) $\mathbf{F}(f, \varepsilon) \leq_{O_\omega} u$,
(2) $\mathbf{F}(f, 0^{t+1}) <_{O_\omega} \mathbf{F}(f, 0^t)$, for all $t < k$,
(3) $\mathbf{F}(f, 0^k) = \underline{0}(= 0)$, and
(4) $\varphi_{H(f,0^k)} = f$.

Definition 5. $\mathcal{S}$ ranges over classes of computable functions.
$\mathbf{Itr}_u\mathbf{BffEx} = \{\mathcal{S} | \mathcal{S}$ is $\mathbf{Itr}_u$-*feasibly learnable*$\}$.

When an $\mathbf{F}$ from above counts down from a notation in O_ω for ω, it is allowed to jump to a notation for any finite ordinal. Let $\mathcal{S}_0$ be the *example* set of functions f such that f has the following properties:
(a) $f(0) > 1$,
(b) $f(1) > 1$,
(c) $f(x) = 1$, for exactly one x, where $1 < x \leq 2^{|f(0)|} + |f(1)|$,
(d) $f(x) = 0$, everywhere else. We have the following

Theorem 6. $\mathcal{S}_0 \in (\mathbf{Itr}_w\mathbf{Bffex}$ - $\mathbf{Itr}_{\underline{n}}\mathbf{Bffex})$, where w is any notation in O_ω for the ordinal ω, and $\underline{n}$ is the notation in O_ω for a *finite* ordinal $n \in \mathbb{N}$.

The particular scheme of feasibly iterating basic feasible learning functionals in Definitions 4 and 5 above requires the count-down function to bottom out at $\underline{0} = 0$, so one can tell when the iterations are done (and can suppress all the programs output but the last). We were initially surprised that, even for a scheme like this, we get a learning hierarchy result like in the just above theorem. We can prove that, for finite ordinals $n \in \mathbb{N}$, the $\mathbf{Itr}_{\underline{n}}\mathbf{Bffex}$ hierarchy collapses. As noted above, we are interested in the investigation of more ways for feasibly iterating basic feasible learning functionals. We'd like variant results where one cannot suppress all the output programs but the last. Here is another feasible notation system, this one for O_{ω^2}, *where*, for the lineartime computable pairing function $< \cdot, \cdot >$ from [40], the notation for ω^2 is 0, and that for ordinals $\omega \times a + b < \omega^2$ is $1 + < a, b >$ — with $<_{O_{\omega^2}}$ defined in the obvious way. We can prove hierarchy results similar to the above for this system.

It is interesting to question the feasible learnability of the example class above, $\mathcal{S}_0 \in \mathbf{Itr}_w\mathbf{Bffex}$. Of course, the counting down and the ordinal notations were all feasible as well as was the basic feasible functional H. Nevertheless, we can prove that, for $\mathbf{Ex}$-learning of $\mathcal{S}_0$, the total learning time of infinitely many $f \in \mathcal{S}_0$ is *inherently exponential* in $|f(0)|$ — while being polynomial in $|f(1)|$. *However*, $\mathcal{S}_0$ is analyzable in terms of *parameterized complexity* [18]. For parameter $k > 1$, let $\mathcal{S}_0^k = (\mathcal{S}_0 \cap \{f \mid f(0) \leq k\})$. Then each $\mathcal{S}_0^k$ is infinite and feasibly learnable. We would like to see this sort of phenomenon more generally analyzed and understood — including in more sophisticated settings.

We would also like to see studied *probabilistic* variants of feasibly iterated feasible learners — this toward producing practical generalizations of Valiant's *PAC* learning [31] and Reischuk and Zeugmann's [38] *stochastically finite learning*. These latter involve, probabilistic, one-shot learners.

References

1. Ambainis, A., Case, J., Jain, S., Suraj, M.: Parsimony hierarchies for inductive inference. Journal of Symbolic Logic 69, 287–328 (2004)
2. Ash, C., Knight, J.: Recursive structures and Eshov's hierarchy. Mathematical Logic Quarterly 42, 461–468 (1996)
3. Baliga, G., Case, J., Merkle, W., Stephan, F., Wiehagen, W.: When unlearning helps, Journal submission (2007)
4. Bārzdiņš, J.: Two theorems on the limiting synthesis of functions. Theory of Algorithms and Programs, Latvian State University, Riga. 210, 82–88 (1974)
5. Bārzdiņš, J., Freivalds, R.: Prediction and limiting synthesis of recursively enumerable classes of functions. Latvijas Valsts Univ. Zinatn. Raksti 210, 101–111 (1974)
6. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM Journal on Computing 26, 1411–1473 (1997)
7. Carlucci, L., Case, J., Jain, S., Stephan, F.: Non U-shaped vacillatory and team learning. In: Jain, S., Simon, H.U., Tomita, E. (eds.) ALT 2005. LNCS (LNAI), vol. 3734, pp. 241–255. Springer, Heidelberg (2005)
8. Carlucci, L., Case, J., Jain, S., Stephan, F.: Memory-limited U-shaped learning. In: Lugosi, G., Simon, H. (eds.) COLT 2006. LNCS (LNAI), vol. 4005, pp. 244–258. Springer, Heidelberg (2006)
9. Case, J.: The power of vacillation in language learning. SIAM Journal on Computing 28(6), 1941–1969 (1999)
10. Case, J., Chen, K., Jain, S., Merkle, W., Royer, J.: Generality's price: Inescapable deficiencies in machine-learned programs. Annals of Pure. and Applied Logic 139, 303–326 (2006)
11. Case, J., Jain, S., Lange, S., Zeugmann, T.: Incremental concept learning for bounded data mining. Information and Computation 152, 74–110 (1999)
12. Case, J., Lynes, C.: Machine inductive inference and language identification. In: Nielsen, M., Schmidt, E.M. (eds.) Automata, Languages, and Programming. LNCS, vol. 140, pp. 107–115. Springer, Heidelberg (1982)
13. Case, J., Moelius, S.: U-shaped, iterative, and iterative-with-counter learning, Submitted (2007)
14. Case, J., Paddock, T., Kötzing, T.: Feasible iteration of feasible learning functionals, Work in progress (2007)
15. Case, J., Smith, C.: Comparison of identification criteria for machine inductive inference. Theoretical Computer Science 25, 193–220 (1983)
16. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge, MA (2001)
17. Daley, R., Smith, C.: On the complexity of inductive inference. Information and Control 69, 12–40 (1986)
18. Downey, R., Fellows, M.: Parameterized Complexity. In: Monographs in Computer Science, Springer, Heidelberg (1998)
19. Ershov, Y.: A hierarchy of sets, I. Algebra i Logika, 7(1):47–74, 1968. In Russian (English translation in Algebra and Logic, 7:25–43 1968) (1968)
20. Ershov, Y.: A hierarchy of sets II. Algebra and Logic 7, 212–232 (1968)
21. Freivalds, R., Smith, C.: On the role of procrastination in machine learning. Information and Computation 107(2), 237–271 (1993)
22. Fulk, M., Jain, S., Osherson, D.: Open problems in Systems That Learn. Journal of Computer and System Sciences 49(3), 589–604 (1994)

23. Gold, E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
24. Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)
25. Hildebrand, F.: Introduction to Numerical Analysis. McGraw-Hill, New York (1956)
26. Hopcroft, J., Ullman, J.: Introduction to Automata Theory Languages and Computation. Addison-Wesley Publishing Company, London, UK (1979)
27. Irwin, R., Kapron, B., Royer, J.: On characterizations of the basic feasible functional, Part I. Journal of Functional Programming 11, 117–153 (2001)
28. Jain, S., Osherson, D., Royer, J., Sharma, A.: Systems that Learn: An Introduction to Learning Theory, 2nd edn. MIT Press, Cambridge, MA (1999)
29. Jain, S., Sharma, A.: Elementary formal systems, intrinsic complexity, and procrastination. Information and Computation 132, 65–84 (1997)
30. Kapron, B., Cook, S.: A new characterization of type 2 feasibility. SIAM Journal on Computing 25, 117–132 (1996)
31. Kearns, M., Vazirani, U.: An Introduction to Computational Learning Theory. MIT Press, Cambridge, MA (1994)
32. Kinber, E., Stephan, F.: Language learning from texts: Mind changes, limited memory and monotonicity. Information and Computation 123, 224–241 (1995)
33. Lange, S., Zeugmann, T.: Incremental learning from positive data. Journal of Computer and System Sciences 53, 88–103 (1996)
34. Marcus, G., Pinker, S., Ullman, M., Hollander, M., Rosen, T.J., Xu, F.: Overregularization in Language Acquisition. In: Monographs of the Society for Research in Child Development (Includes commentary by H. Clahsen), vol. 57(4), University of Chicago Press, Chicago (1992)
35. Mehlhorn, K.: Polynomial and abstract subrecursive classes. Journal of Computer and System Sciences 12, 147–178 (1976)
36. Odifreddi, P.: Classical Recursion Theory, volume II. Elsivier, Amsterdam (1999)
37. Plunkett, K., Marchman, V.: U-shaped learning and frequency effects in a multilayered perceptron: implications for child language acquisition. Cognition 86(1), 43–102 (1991)
38. Reischuk, R., Zeugmann, T.: An average-case optimal one-variable pattern language learner. Journal of Computer and System Sciences (Special Issue for COLT'98) 60(2), 302–335 (2000)
39. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw Hill, New York, 1967. Reprinted, MIT Press (1987)
40. Royer, J., Case, J.: Subrecursive Programming Systems: Complexity and Succinctness. Research monograph in: Progress in Theoretical Computer Science. Birkhäuser Boston (1994)
41. Sipser, M.: Private communication (1978)
42. Strauss, S., Stavy, R. (eds.): U-Shaped Behavioral Growth. Academic Press, NY (1982)
43. Taatgen, N., Anderson, J.: Why do children learn to say broke? A model of learning the past tense without feedback. Cognition 86(2), 123–155 (2002)
44. Wiehagen, R.: Limes-erkennung rekursiver funktionen durch spezielle strategien. Electronische Informationverarbeitung und Kybernetik 12, 93–99 (1976)