Dynamically Delayed Postdictive Completeness and Consistency in Learning

John Case and Timo Kötzing

Department of Computer and Information Sciences,
University of Delaware, Newark, DE 19716-2586, USA
{case,koetzing}@cis.udel.edu

Abstract. In computational function learning in the limit, an algorithmic *learner* tries to find a program for a computable function g given successively more values of g, each time outputting a conjectured program for g. A learner is called *postdictively complete* iff all available data is correctly postdicted by each conjecture.

Akama and Zeugmann presented, for each choice of natural number δ, a relaxation to postdictive completeness: each conjecture is required to postdict only all *except the last δ seen* data points.

This paper extends this notion of delayed postdictive completeness from *constant* delays to *dynamically* computed delays. On the one hand, the delays can be different for different data points. On the other hand, delays no longer need to be by a fixed finite number, but any type of computable countdown is allowed, including, for example, countdown in a system of ordinal notations and in other graphs disallowing *computable* infinitely descending counts.

We extend many of the theorems of Akama and Zeugmann and provide some feasible learnability results. Regarding *fairness* in feasible learning, one needs to limit use of tricks that postpone output hypotheses until there is enough time to "think" about them. We see, for polytime learning, postdictive completeness (and delayed variants): 1. allows *some* but *not* all postponement tricks, *and* 2. there is a surprisingly tight boundary, for polytime learning, between what postponement is allowed and what is not. For *example*: 1. the set of polytime computable functions *is* polytime postdictively completely learnable employing some postponement, *but* 2. the set of exptime computable functions, while polytime learnable with a little more postponement, is *not* polytime postdictively completely learnable! We have that, for w a notation for ω, the set of exptime functions *is* polytime learnable with *w-delayed* postdictive completeness. Also provided are generalizations to further, small constructive limit ordinals.

1 Introduction

"Explanatory learning", or Ex-learning for short, is a standard model of limit learning of computable functions. In this model a learner is given successively longer initial segments of a target function. For each initial segment of the target function, the learner gives an hypothesis. The learner is said to successfully *Ex-learn* the target function iff the infinite sequence of hypotheses generated by the learner on the initial segments of the target functions converges in the limit to a (single) correct program for the target function [JORS99].

In some literature on limit learning this intuitively simple success criterion is used as a minimal requirement for success, and additional requirements are defined and examined. We call two such extra requirements *postdictive completeness* (the hypotheses correctly postdict the data seen so far) and *postdictive consistency* (the hypotheses do not explicitly contradict the given data) [Bār74, BB75, Wie76, Wie78].[1] There are Ex-learnable sets of functions that cannot be learned with the additional requirement of postdictive completeness or consistency.

Akama and Zeugmann [AZ07] presented success criteria that are a little less restrictive than postdictively complete Ex-learning. Their criteria delay the requirement to postdict a given datum by a fixed natural number δ of (not necessarily distinct) hypotheses output. For ordinary postdictive completeness, if a learner h has seen so far, on a computable g input, $g(0), \ldots, g(n-1)$, then h's corresponding hypothesis, p_n, must correctly compute $g(0), \ldots, g(n-1)$.[2] For delay δ, Akama and Zeugmann, require only that, on $g(0), \ldots, g(n-1)$, h's *later* hypothesis, $p_{n+\delta}$, must correctly compute $g(0), \ldots, g(n-1)$. Essentially, the delay δ learner could, after seeing $g(0), \ldots, g(n-1)$, run a counter down from δ to 0 to see which future hypothesis must correctly compute $g(0), \ldots, g(n-1)$.

In the present paper we extend this notion of delayed postdictive completeness from *constant* delays δ to *dynamically computed* delays. *One* of the ways we consider herein to do this involves counting down from notations for constructive ordinals. We explain. Everyone knows how to use the natural numbers for counting, including for counting *down*. Freivalds and Smith [FS93], as well as [ACJS04], employed in learning theory *notations for constructive ordinals* [Rog67, § 11.7] as devices for algorithmic counting down. Theorems 4 and 5 in Section 3 provide strong justification for studying the herein ordinal countdown variants of Postdictive Completeness.

[SSV04] gives a further generalized notion of counting down. They consider certain partial orders with no *computable* infinitely descending chains. In the present paper we consider arbitrary and also computable graphs with no infinite, computable paths, and we algorithmically count "down" along their paths. Theorem 11, in Section 4.2 below, gives a nice example of a linearly ordered, com-

[1] We use the terminology *postdictive completeness* because the the hypotheses must *completely* postdict the data seen to that point. We use the terminology *postdictive consistency* because the the hypotheses need only *avoid explicit inconsistencies* with the data seen to that point. Such an hypothesis may, then, on some input for which the data seen to that point tells the answer, go undefined (i.e., go into an infinite loop) and, thereby, not explicitly contradict the known data. In the literature on these requirements, except for [Ful88], what we call postdictively complete is called *consistent*, and what we call postdictively consistent is called *conformal*.

[2] Note that, for $n = 0$, the data seen is empty and the output hypothesis, p_0, is unconstrained.

putable such graph which nonetheless has infinite paths (just not computable ones). We call our graphs in the present paper, *countdown graphs*.

We make use of countdown graphs for delaying the requirement of postdictive completeness (respectively, consistency) by requiring a learner to start an *independent* countdown for each datum $g(i)$ seen and to be postdictively complete (respectively, consistent) regarding $g(i)$ as soon as the countdown for $g(i)$ terminates.[3]

Section 2 will introduce the notation and concepts used in this paper.

In the prior literature we also see further variants of postdictive completeness and consistency not based on delay. For example, [CJSW04] surveys with references these variants. Roughly, below, when we attach $\mathcal{R}$ to the front of a name of a criterion requiring postdictive completeness or consistency, this means that the associated learnability must be witnessed by a (total) computable learner as opposed to just a partial computable learner (defined at least where it minimally needs to be); when we attach a $\mathcal{T}$ to the front of a name of a criterion requiring postdictive completeness (respectively, consistency), this means that the associated learnability must be witnessed by a (total) computable learner which is postdictively complete (respectively, consistent) on *all* input functions regardless of whether the learner actually learns them.

Sections 3 and 4 present our results. All of our results in Section 3 provide information about polynomial time learners. Furthermore, some of our results in Section 4.1 entail learnability with linear time learners. These time bounds are uniform bounds on how much time it takes the learner to conjecture each hypothesis in terms of the total size of the input data it can use for making this conjecture. Suppose for discussion p is a polynomial time bound. Pitt [Pit89] notes that Ex-learning allows unfair postponement tricks, i.e., a learner h can put off outputting significant conjectures based on data σ until it has seen a much larger sequence of data τ so that $p(|\tau|)$ is enough time for h to think about σ as long as it needs.[4] In this way the polytime restriction on each output does not, by itself, have the desirable effect of constraining the total learning time. Pitt [Pit89] then lays out some additional constraints toward avoiding "cheating" by such postponement tricks. He discusses in this regard what we are calling postdictive completeness. He also considers further constraints since he wants to forbid enumeration techniques [JORS99]. For our complexity-bounded results in Section 4.1 we get by with an extremely fair, restricted kind of *linear-time* learner, we call *transductive*. A transductive learner has access only to its current datum.

In Section 3 we see, from Theorems 4 and 5 and the proof of the first, that, for polytime learning, postdictive completeness (and delayed variants): 1. allows *some* but *not* all postponement tricks, *and* 2. there is a surprisingly tight boundary, for polytime learning, between what postponement is allowed and what is not. For *example*: 1. the set of polytime computable functions *is* polytime

[3] Below we refer to a vector of such individual counts as a *multicount*.

[4] Pitt talks in this context of *delaying tricks*. We changed this terminology due to the clash with Akama and Zeugmann's terminology for *delayed* postdictive completeness.

postdictively completely Ex-learnable (by a complexity-bounded enumeration technique) employing some postponement, *but* 2. the set of exptime computable functions, while polytime Ex-learnable with a little more postponement, is *not* polytime postdictively completely Ex-learnable! From Theorem 4, we see that, for w a notation for ω, the set of exptime functions *is* polytime Ex-learnable with w*-delayed* postdictive completeness. Theorems 4 and 5 also provide generalizations to further, small constructive limit ordinals.

Section 4.1 shows how the different variants of our criteria relate in learning power. Our main theorem in this section is Theorem 6. For *example*, it entails that there is a set of computable functions which is postdictively *consistently* learnable (with no delays) by a transductive, linear time learner *but* is *not* postdictively *completely* learnable with delays employing *any* countdown graph.

In Section 4.2, our main result, Theorem 13, entails (including with Corollaries) *complete characterizations* of learning power *in dependence on* associated (computable) *countdown graphs.* Corollary 16 extends the finite hierarchy given in [AZ07] into the constructive transfinite.

We omit most proofs due to space constraints. A more complete version of the present paper can be found online [CK08]. Many of our omitted proofs use Kleene's Recursion Theorem [Rog67, page 214, problem 11-4] or the Case's *Operator Recursion Theorem* [Cas74] (and are a bit combinatorially difficult).

2 Mathematical Preliminaries

Any unexplained complexity-theoretic notions are from [RC94]. All unexplained general computability-theoretic notions are from [Rog67].

Strings herein are finite and over the alphabet $\{0,1\}$. $\{0,1\}^*$ denotes the set of all such strings; ε denotes the empty string.

$\mathbb{N}$ denotes the set of natural numbers, {0,1,2,...}. We do not distinguish between natural numbers and their *dyadic* representations as strings.[5]

For each $w \in \{0,1\}^*$ and $n \in \mathbb{N}$, w^n denotes n copies of w concatenated end to end. For each string w, we define $\mathrm{size}(w)$ to be the length of w. As we identify each natural number x with its dyadic representation, for all $n \in \mathbb{N}$, $\mathrm{size}(n)$ denotes the length of the dyadic representation of n. For all strings w, we define $|w|$ to be $\max\{1, \mathrm{size}(w)\}$. [6]

The symbols $\subseteq, \subset, \supseteq, \supset$ respectively denote the subset, proper subset, superset and proper superset relation between sets.

For sets A, B, we let $A \setminus B := \{a \in A \mid a \notin B\}$, $\overline{A} := \mathbb{N} \setminus A$.

We sometimes denote a function f of $n > 0$ arguments $x_1, \ldots, x_n$ in lambda notation (as in Lisp) as $\lambda x_1, \ldots, x_n \centerdot f(x_1, \ldots, x_n)$. For example, with $c \in \mathbb{N}$, $\lambda x \centerdot c$ is the constantly c function of one argument.

[5] The *dyadic* representation of a natural number $x :=$ the x-th finite string over $\{0,1\}$ *in lexicographical order*, where the counting of strings starts with zero [RC94]. Hence, unlike with binary representation, lead zeros matter.

[6] This convention about $|\varepsilon| = 1$ helps with runtime considerations.

A function ψ is *partial computable* iff there is a Turing machine computing ψ. $\mathcal{R}$ and $\mathcal{P}$ denote the set of all (total) computable and partial computable functions $\mathbb{N} \to \mathbb{N}$, respectively. If ψ is not defined for some argument x, then we denote this fact by $\psi(x)\uparrow$, and we say that ψ on x *diverges*. The opposite is denoted by $\psi(x)\downarrow$, and we say that ψ on x converges.

We say that a partial function ψ *converges to* p iff $\forall^\infty x : \psi(x)\downarrow = p$.

[RC94, §3] describes an *efficiently* numerically named or coded[7] programming system for multi-tape Turing machines (TMs) which compute the partial computable functions $\mathbb{N} \to \mathbb{N}$. Herein we name this system φ. φ_p denotes the partial computable function computed by the TM-program with code number p in the φ-system, and Φ_p denotes the partial computable *runtime* function of the TM-program with code number p in the φ-system. In the present paper, we employ a number of complexity bound results from [RC94, §§ 3 & 4] regarding (φ, Φ). These results will be clearly referenced as we use them.

We fix the 1-1 and onto pairing function $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ from [RC94], which is based on dyadic bit-interleaving. Pairing and unpairing is computable in linear time. π_1 and π_2, respectively, denote the unpairing into the left and right component of a given coded pair, respectively.

For all $f, g \in \mathcal{R}$ we let $\langle f, g \rangle$ denote $\lambda i \centerdot \langle f(i), g(i) \rangle$.

Whenever we consider tuples of natural numbers as input to TMs, it is understood that the general coding function $\langle \cdot, \cdot \rangle$ is used to (left-associatively) code the tuples into appropriate TM-input.

A *finite sequence* is a mapping with a finite initial segment of $\mathbb{N}$ as domain (and range, $\mathbb{N}$). $\emptyset$ denotes the empty sequence (and, also, the empty set). The set of all finite sequences is denoted by $\mathbb{S}\mathrm{eq}$. For each finite sequence σ, we will denote the first element, if any, of that sequence by $\sigma(0)$, the second, if any, with $\sigma(1)$ and so on. $\mathrm{last}(\sigma)$ denotes the last element of σ, if any. $\#\mathrm{elets}(\sigma)$ denotes the number of elements in a finite sequence σ, that is, the cardinality of its domain.

We use $\diamond$ (with infix notation) to denote concatenation on sequences. For any natural number x, we let $\overline{x}$ denote the sequence of length one with only element x, and we let $\overline{x}^n$ be the code of the sequence of length n, each element being x.

From now on, by convention, f, g and h with or without decoration range over (partial) functions $\mathbb{N} \to \mathbb{N}$, x, y with or without decorations range over $\mathbb{N}$ and σ, τ with or without decorations range over finite sequences of natural numbers.

Following [LV97], we fix a coding $\langle \cdot \rangle_{\mathbb{S}\mathrm{eq}}$ of all sequences into $\mathbb{N}$ $(= \{0,1\}^*)$ – with the following properties.

The set of all codes of sequences is decidable in linear time. The time to encode a sequence, that is, to compute $\lambda k, v_1, \ldots, v_k \centerdot \langle v_1, \ldots, v_k \rangle_{\mathbb{S}\mathrm{eq}}$ is $\mathcal{O}(\lambda k, v_1, \ldots, v_k \centerdot \sum_{i=1}^{k} |v_i|)$. Therefore, the size of the codeword is also linear in the size of the elements: $\lambda k, v_1, \ldots, v_k \centerdot |\langle v_1, \ldots, v_k \rangle_{\mathbb{S}\mathrm{eq}}|$ is

[7] This numerical coding guarantees that many simple operations involving the coding run in linear time. This is by contrast with historically more typical codings featuring prime powers and corresponding at least exponential costs to do simple things.

$\mathcal{O}(\lambda k, v_1, \ldots, v_k \centerdot \sum_{i=1}^{k} |v_i|)$.[8] We also have $\lambda\langle\sigma\rangle_{\mathbb{S}\mathrm{eq}} \centerdot \#\mathrm{elets}(\sigma)$ is linear time computable; $\lambda\langle\sigma\rangle_{\mathbb{S}\mathrm{eq}}, i \centerdot \begin{cases}\sigma(i), & \text{if } i < \#\mathrm{elets}(\sigma);\\ 0, & \text{otherwise,}\end{cases}$ is linear time computable; and

$$\forall\sigma : \#\mathrm{elets}(\sigma) \leq |\langle\sigma\rangle_{\mathbb{S}\mathrm{eq}}|. \tag{1}$$

Henceforth, we will many times identify a finite sequence σ with its code number $\langle\sigma\rangle_{\mathbb{S}\mathrm{eq}}$. However, when we employ expressions such as $\sigma(x)$, $\sigma = f$ and $\sigma \subset f$, we consider σ as a *sequence*, not as a number.

For a partial function g and $i \in \mathbb{N}$, if $\forall j < i : g(j)\downarrow$, then $g[i]$ is defined to be the finite sequence $g(0), \ldots, g(i-1)$.

A *pre-order* is a pair $(A, \leq_A)$ such that $\leq_A$ is a transitive and reflexive binary relation on A.

Church and Kleene introduced systems of ordinal notations. See Rogers [Rog67, § 11.7]. For us, a *system of ordinal notations* is a pair $(\mathcal{N}, \leq_\mathcal{N})$ and associated functions $k_\mathcal{N}, p_\mathcal{N}, q_\mathcal{N} \in \mathcal{P}$ and $\nu_\mathcal{N}$ mapping $\mathcal{N}$ into the set of all constructive ordinals, such that $\mathcal{N} \subseteq \mathbb{N}$, and, for all $u, v \in \mathcal{N}$, we have: $u \leq_\mathcal{N} v$ iff $\nu_\mathcal{N}(u) \leq \nu_\mathcal{N}(v)$; if $\nu_\mathcal{N}(u) = 0$, then $k_\mathcal{N}(u) = 0$; if $\nu_\mathcal{N}(u)$ is successor ordinal, then $k_\mathcal{N}(u) = 1$ and $\nu_\mathcal{N}(p_\mathcal{N}(u)) + 1 = \nu_\mathcal{N}(u)$; if $\nu_\mathcal{N}(u)$ is limit ordinal, then $k_\mathcal{N}(u) = 2$ and $\varphi_{q_\mathcal{N}(u)}$ is a monotonic increasing computable function such that $\nu_\mathcal{N} \circ \varphi_{q_\mathcal{N}(u)}$ converges to $\nu_\mathcal{N}(u)$.

Note that $\leq_\mathcal{N}$ is not necessarily computable. If it is, then $(\mathcal{N}, \leq_\mathcal{N})$ is called *computably related.*

For countdown in polynomial time, we use *feasibly related feasible systems of ordinal notations* [CKP07]. In such systems, many predicates and operations on notations are feasibly computable. For example, one can use the (efficiently) coded tuple $\langle a_n, \ldots, a_0\rangle$ as a notation for the ordinal $\omega^n \cdot a_n + \ldots \omega^0 \cdot a_0$. The resulting system of ordinal notations gives a notation to all ordinals $< \omega^\omega$ and allows for polytime comparing, adding and so on.

Note that, for any constructive ordinal α, there is a computably related system of ordinal notations which gives a notation to α [Rog67]; furthermore, there is also a feasibly related feasible system of ordinal notations giving a notation to α [CKP07].

In this paper we consider *several* indexed families of learning criteria. We proceed somewhat abstractly to avoid needless terminological repetitions.

For each $\mathcal{C} \subseteq \mathcal{P}$ and $\delta \subseteq \mathcal{R}^2$, we say that the pair $(\mathcal{C}, \delta)$ is a *learning criterion* (for short, *criterion*). The set $\mathcal{C}$ is called a *learner admissibility restriction*, and intuitively serves as a limitation on what functions will be considered as learners. Typical learner admissibility restrictions are $\mathcal{P}, \mathcal{R}$, as well as complexity classes. The predicate δ is called a *sequence acceptance criterion*, intuitively restricting what output-sequences by the learner are considered a successful learning of a given function. For $h \in \mathcal{P}, g \in \mathcal{R}$ we say that h $(\mathcal{C}, \delta)$-learns g iff $h \in \mathcal{C}$ and $(\lambda x \centerdot h(g[x]), g) \in \delta$. For $h \in \mathcal{P}, g \in \mathcal{R}$, we call $\lambda x \centerdot h(g[x])$ the *learning-sequence* of h given g. Here's an *example* δ, herein called **Ex**. Let $\mathbf{Ex} = \{(\langle p, d\rangle, q) \in$

[8] For these $\mathcal{O}$-formulas, $|\varepsilon| = 1$ helps.

$\mathcal{R}^2 \mid p$ converges to some $e \ \wedge \ \varphi_e = q\}$. Intuitively, $(\langle p, d\rangle, q) \in \mathbf{Ex}$ means that the learning-sequence $\langle p, d\rangle$ successfully learns the function q iff: for some i, $p(i)$ is a correct program number for q, and this hypothesized program number will never change after that point i. N.B. For *this* example, the learning-sequence is a sequence of coded pairs and **Ex** completely disregards the second component d. Some *other* sequence acceptance criteria below make use of d as an auxiliary output of the learner. In these cases, d will code countdowns until some events of interest must happen. For $h \in \mathcal{P}$ and $\mathcal{S} \subseteq \mathcal{R}$ we say that h $(\mathcal{C}, \delta)$-learns $\mathcal{S}$ iff, for all $g \in \mathcal{S}$, h $(\mathcal{C}, \delta)$-learns g. The set of $(\mathcal{C}, \delta)$-learnable sets of computable functions is $\mathcal{C}\delta := \{\mathcal{S} \subseteq \mathcal{R} \mid \exists h \in \mathcal{C} : h \ (\mathcal{C}, \delta)\text{-learns } \mathcal{S}\}$. Instead of writing the pair $(\mathcal{C}, \delta)$, we will ambiguously write $\mathcal{C}\delta$. We will omit $\mathcal{C}$ if $\mathcal{C} = \mathcal{P}$.[9] One way to *combine* two sequence acceptance criteria δ and δ' is to intersect them as sets. We write $\delta\delta'$ for the intersection, and we present examples featuring countdowns in the next section.

We can turn a given sequence acceptance criterion δ into a learner admissibility restriction $\mathcal{T}\delta$ by admitting only those learners that obey δ *on all input*: $\mathcal{T}\delta := \{h \in \mathcal{P} \mid \forall g \in \mathcal{R} : (\lambda x \,\textbf{.}\, h(g[x]), g) \in \delta\}$.

The following two definitions formalize the intuitive discussion about countdown graphs as given above in Section 1.

A *graph* is a pair $(G, \rightarrow)$, where $G \subseteq \mathbb{N}$ and $\rightarrow$ is a binary relation on G. We will use infix notation for $\rightarrow$. For each graph $(G, \rightarrow)$, we say that τ is a *G-path* iff $\#\text{elets}(\tau) > 0$, $\forall i < \#\text{elets}(\tau) : \tau(i) \in G$ and $\forall i < \#\text{elets}(\tau) - 1 : \tau(i) \rightarrow \tau(i+1)$. For each graph G, let $\vec{G}$ denote the set of all G-paths. (S, R) is a *subgraph* of $(G, \rightarrow)$, iff $S \subseteq G$ and R is $\rightarrow$ restricted to $(S \times S)$.

For all $m, n \in \mathbb{N}$, we write $m \rightarrow^* n$ (respectively, $m \rightarrow^+ n$) iff there is a G-path τ such that $\tau(0) = m$, $\text{last}(\tau) = n$ (respectively, additionally $\#\text{elets}(\tau) > 1$). We sometimes write G for $(G, \rightarrow)$. A graph $(G, \rightarrow)$ is said to be *computable* iff G and $\rightarrow$ are computable. Note that $G \in \mathcal{G}$ is computable iff $\vec{G}$ is computable. For a graph $(G, \rightarrow)$ we sometimes identify $m \in G$ with $\{n \in G \mid m \rightarrow^+ n\}$. With every pre-order $(A, \leq_A)$ we associate the graph $(A, >_A)$, where, for all $a, b \in A$, $a >_A b$ iff ($b \leq_A a$ and $a \not\leq_A b$).

A graph $(G, \rightarrow)$ is called a *countdown graph*, iff $\neg\exists r \in \mathcal{R} \forall i \in \mathbb{N} : r(i) \rightarrow r(i+1)$. Note that if G is a countdown graph, then so is every subgraph of G. Let $\mathcal{G}$, $\mathcal{G}_{\text{comp}}$, respectively, denote the set of all and all computable countdown-graphs, respectively.

Example countdown graphs can be obtained from systems of ordinal notations. Let $(\mathcal{N}, \leq_{\mathcal{N}})$ be a system of ordinal notations. Then, $(\mathcal{N}, \leq_{\mathcal{N}})$ is a pre-order without infinite descending chains, so the graph associated with $(\mathcal{N}, \leq_{\mathcal{N}})$ is a countdown graph. If $(\mathcal{N}, \leq_{\mathcal{N}})$ is computably related, then the associated graph will be computable.

[9] Thus, every sequence acceptance criterion denotes at the same time a learning criterion and the set of learnable sets. It will be clear from context which meaning is intended. An example: **Ex**, then, denotes sequence acceptance criterion **Ex**, learning criterion $(\mathcal{P}, \mathbf{Ex})$ and set $\mathcal{P}\mathbf{Ex}$ of $(\mathcal{P}, \mathbf{Ex})$-learnable sets.

In Theorem 11 below we give one example of a countdown graph not based on a system of ordinal notations.

Soon we define what postdictive completeness, respectively consistency, with respect to $G \in \mathcal{G}$ means. Intuitively, every learner is required to have two outputs: a hypothesis, and a countdown output. For any learnee $g \in \mathcal{R}$, if the learner sees $g[i]$, the countdown output will need to encode one countdown for each $j < i$. As soon as the countdown for a given data item is over, the hypothesis has to be postdictively complete, respectively consistent, *for that data item.* We will refer to the countdown output of a learner as a *multicount* (as it represents more than one countdown). We refer to a learning-output of hypothesis and multicount as a *hypothesis-multicount.*

The set of all *multicountdown sequences* is defined as $\mathbb{M} := \{\sigma \in \mathbb{S}\mathrm{eq} \mid \forall i < \#\mathrm{elets}(\sigma) : (\sigma(i) \in \mathbb{S}\mathrm{eq} \wedge \#\mathrm{elets}(\sigma(i)) = i)\}$.[10]

An example multicountdown sequence is $\sigma_0 := \langle\rangle_{\mathbb{S}\mathrm{eq}}, \langle 3\rangle_{\mathbb{S}\mathrm{eq}}, \langle 2,3\rangle_{\mathbb{S}\mathrm{eq}}, \langle 1,2,2\rangle_{\mathbb{S}\mathrm{eq}}, \langle 0,1,2,1\rangle_{\mathbb{S}\mathrm{eq}}, \langle 0,0,2,2,2\rangle_{\mathbb{S}\mathrm{eq}}, \langle 2,0,2,3,1,1\rangle_{\mathbb{S}\mathrm{eq}}, \langle 0,0,2,7,0,0,5\rangle_{\mathbb{S}\mathrm{eq}}$. σ_0 can be displayed as a matrix like this:

$$\sigma_0 = \begin{array}{c|cccccccc} {}_{n}\,{}^{x} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & & 3 & 2 & 1 & 0 & 0 & 2 & 0 \\ 1 & & & 3 & 2 & 1 & 0 & 0 & 0 \\ 2 & & & & 2 & 2 & 2 & 2 & 2 \\ 3 & & & & & 1 & 2 & 3 & 7 \\ 4 & & & & & & 2 & 1 & 0 \\ 5 & & & & & & & 1 & 0 \\ 6 & & & & & & & & 5 \end{array}. \qquad (2)$$

In (2) each column is a multicount. For example, column $x = 4$ represents the multicount $\sigma_0(4) = \langle 0,1,2,1\rangle$. Each row of (2) provides the successive values of a particular countdown. For example, the n-th row of (2) (without initial empty entries) is the n-th countdown of σ_0. As we will see below, for an associated learnee g, the n-th row will be relevant to $g(n)$.

For each $\sigma \in \mathbb{M}$ and $n < \#\mathrm{elets}(\sigma) - 1$ we define $\mathrm{row}(n,\sigma) := \langle \sigma(n+1)(n), \ldots, \sigma(\#\mathrm{elets}(\sigma)-1)(n)\rangle_{\mathbb{S}\mathrm{eq}}$. For σ_0 as presented above in (2), we have, for example, $\mathrm{row}(4,\sigma_0) = \langle 2,1,0\rangle_{\mathbb{S}\mathrm{eq}}$. Each $\mathrm{row}(n,\sigma)$ is a countdown.

We will consider a given countdown sequence τ as *terminated* with respect to a given countdown graph $G \in \mathcal{G}$, iff $\tau \notin \vec{G}$. We then say that "τ has terminated" or "τ has bottomed out". For a given multicountdown sequence we will define the set of all n such that the n-th countdown has (started and) bottomed out just below. For all σ and all $G \in \mathcal{G}$, define $\perp_G(\sigma) = \{n < \#\mathrm{elets}(\sigma) - 1 \mid \sigma \notin \mathbb{M} \vee \mathrm{row}(n,\sigma) \notin \vec{G}\}$. We omit the subscript G whenever no confusion can arise.

We pronounce $\perp$ as "bottom". For $\sigma \in \mathbb{M}$, $\perp(\sigma)$ is the set of all countdown numbers where the countdown has terminated.

Let us, for example, consider the finite countdown graph G on $\{0,1,2,3\}$ with the natural $>$-order on $\mathbb{N}$. For σ_0 depicted above in (2), we have $\perp_G(\sigma_0) =$

[10] Of course, $\sigma(i) \in \mathbb{S}\mathrm{eq}$ means that the number $\sigma(i)$ is the code of a sequence.

$\{0, 1, 2, 3, 6\}$. The example of rows $n = 4$ and $n = 5$ shows that reaching a minimal element (in this case 0) of G does not imply immediate termination of the countdown. The example of rows $n = 2$ and $n = 3$ shows how countdowns terminate when not obeying the graph relation. Note that the countdown for row $n = 6$ has terminated immediately when it started, as it started with 5, and $\langle 5\rangle_{\mathbb{S}\mathrm{eq}}$ is not a G-path. From rows $n = 4$ and $n = 6$ we see that the different countdowns do not have to terminate in row order.

Next we define two families of sequence acceptance criteria, employing countdowns as described above. The rest of the paper will be concerned with studying these criteria in various settings.

Definition 1. For $G \in \mathcal{G}$ let, for all $p, d, q \in \mathcal{R}$,

- $\mathbf{Pcp}_G(\langle p, d\rangle, q) :\Leftrightarrow \forall x \forall n \in \bot_G(d[x]) : \varphi_{p(x)}(n)\downarrow = q(n)$; and
- $\mathbf{Pcs}_G(\langle p, d\rangle, q) :\Leftrightarrow \forall x \forall n \in \bot_G(d[x]) : \varphi_{p(x)}(n)\downarrow \Rightarrow \varphi_{p(x)}(n) = q(n)$.

For all $g \in \mathcal{R}$ and $h, f \in \mathcal{P}$, we say that $\langle h, f\rangle$ *works postdictively completely (respectively, consistently) on* g *with* G*-delay* iff $(\lambda i \centerdot (\langle h(g[i]), f(g[i])\rangle), g) \in \mathbf{Pcp}_G$ (respectively, $\mathbf{Pcs}_G$). We omit "with G-delay", if no confusion can arise.

3 Complexity Results

For this section only, let $\mathcal{N}$ be a feasibly related feasible system of ordinal notations for at least the ordinals $< \omega^2$. Let w be a notation for ω in $\mathcal{N}$. For each $n \in \mathbb{N}$, $\underline{n}$ denotes a notation for n in $\mathcal{N}$, such that $\lambda n \centerdot \underline{n}$ is computable in polytime. We will assume for all constructive ordinals α,

$$\forall n \in \mathbb{N}, u \in \mathcal{N} : (u \text{ is notation in } \mathcal{N} \text{ for } \alpha + n) \Rightarrow n \leq u.[11] \tag{3}$$

Definition 2. Let exp denote the function $\lambda x \centerdot 2^x$. Furthermore, for all n, we write $\exp^n$ for the n-times application of exp. In particular, $\exp^0$ denotes the identity. For all k let $\mathrm{Exp}_k\mathrm{Programs} := \{e \mid e \in \mathbb{N} \ \wedge \ \exists p \text{ polynomial } \forall n \in \mathbb{N} : \Phi_e(n) \leq \mathit{exp}^k(p(|n|))\}$ and $\mathbf{EXP}_k\mathbf{F} := \{\varphi_e \mid e \in \mathrm{Exp}_k\mathrm{Programs}\}$. Also, we let $\mathrm{ExpPrograms} := \mathrm{Exp}_1\mathrm{Programs}$, $\mathbf{EXPF} := \mathbf{EXP}_1\mathbf{F}$, $\mathrm{PolyPrograms} := \mathrm{Exp}_0\mathrm{Programs}$ and $\mathbf{PF} := \mathbf{EXP}_0\mathbf{F}$.

For $g \in \mathbf{PF}$ we say that g is *computable in polytime*, or also, *feasibly computable*. Recall that we have, by (1), $\forall \sigma : \#\mathrm{elets}(\sigma) \leq |\sigma|$.

Definition 3. Let S, T be such that

$$\forall p, x, t : S(p, x, t) = \begin{cases} \varphi_p(x), & \text{if } \Phi_p(x) \leq |t|; \\ 0, & \text{otherwise}; \end{cases} \tag{4}$$

$$\forall p, x, t : T(p, x, t) = \begin{cases} 1, & \text{if } \Phi_p(x) \leq |t|; \\ 0, & \text{otherwise}. \end{cases} \tag{5}$$

[11] Specific systems of ordinal notations seen in the literature typically, perhaps always, satisfy (3).

We now use the notion introduced above for subscripting our criteria with ordinal notations instead of countdown graphs.

Theorem 4
(a) $\mathbf{PF} \in \mathbf{PFPcp}_{\underline{0}}\mathbf{Ex}$.
(b) $\mathbf{EXPF} \in \mathbf{PFPcp}_{w}\mathbf{Ex}$.
(c) $\forall n : \mathbf{EXP}_n\mathbf{F} \in \mathbf{PFPcp}_{w \cdot \underline{n}}\mathbf{Ex}$.

Furthermore, each of (a), (b) and (c) is witnessed by a respective learner $\langle h, f\rangle$ such that range$(h) \subseteq$ PolyPrograms, $\subseteq$ ExpPrograms and $\subseteq$ Exp$_n$Programs, respectively.

Proof of (a). This proof employs a complexity-bounded enumeration technique [JORS99]. By [RC94, Theorem 3.13], there is a linear time computable patch$_0$ such that,

$$\forall\sigma\forall x : \varphi_{\mathrm{patch}_0(\sigma)}(x) = \begin{cases} \sigma(x), & \text{if } x < \#\mathrm{elets}(\sigma); \\ 0, & \text{otherwise;} \end{cases} \tag{6}$$

and all outputs of patch$_0$ are programs computable in linear time.

By [RC94, Theorems 4.13(b) & 4.17] there is a linear time computable e such that $\mathbf{PF} = \{\varphi_{e(j)} \mid j \in \mathbb{N}\}$ and $\forall j \in \mathbb{N} : e(j) \in$ PolyPrograms. From [RC94, Corollary 3.7] S and T from (4) and (5) above are polytime computable.[12] Then, by [RC94, Lemmas 3.15 & 3.16], it is easy to see that there is $h \in \mathbf{PF}$ such that

$$\forall\sigma : h(\sigma) = \begin{cases} e(j), & \begin{array}{l}\text{if there is a minimal} \\ j \leq |\sigma| : \forall x < \#\mathrm{elets}(\sigma) : \\ (T(e(j), x, \sigma) \wedge\ S(e(j), x, \sigma) = \sigma(x));\end{array} \\ \mathrm{patch}_0(\sigma), & \text{otherwise.} \end{cases} \tag{7}$$

To show that h converges on all $g \in \mathbf{PF}$: Let $g \in \mathbf{PF}$. Let j_0 be minimal such that $\varphi_{e(j_0)} = g$. Let p be a polynomial such that $\forall x : \Phi_{e(j_0)}(x) \leq p(|x|)$. We then have the following.

– $\forall^\infty n, j_0 \leq n \leq |g[n]|$ (by (1)).
– $\forall^\infty n \forall j < j_0 : g[n] \not\subseteq \varphi_{e(j)}$ (as j_0 minimal such that $\varphi_{e(j_0)} = g$).
– We have $\forall^\infty x : \Phi_{e(j_0)}(x) \leq x$.[13] Hence, $\forall^\infty n \forall x \leq n : \Phi_{e(j_0)}(x) \leq n$.[14] Therefore, using (1), $\forall^\infty n \forall x < n : T(e(j_0), x, g[n])$; hence, also $\forall^\infty n \forall x < n : S(e(j_0), x, g[n]) = \varphi_{e(j_0)}(x) = g(x)$.

By the three items above, we have $\forall^\infty n : h(g[n]) = e(j_0)$. Let $f = \lambda\sigma\centerdot\underline{0}$. Obviously, $\langle h, f\rangle$ witnesses $\mathbf{PF} \in \mathbf{Pcp}_{\underline{0}}\mathbf{Ex}$. The furthermore clause follows from the choice of e and patch$_0$. □ (OF (a))

[12] N.B. S and T above are variants of the S and T featured in [RC94, Corollary 3.7].

[13] By [RC94, §2.5, (9)], there are $a, b \in \mathbb{N}$ such that $\forall x : 2^{|x|} \leq a \cdot x + b$; thus, there is $d > 0$ such that $\forall^\infty x : 2^{|x|} \leq d \cdot x$. Clearly, $\forall^\infty x : p(|x|) \leq \frac{1}{d}2^{|x|}$. Thus, $\forall^\infty x : p(|x|) \leq x$.

[14] Let n_0, n_1 be such that $\forall x \geq n_0 : \Phi_{e(j_0)}(x) \leq x$ and $\forall x < n_0 : \Phi_{e(j_0)}(x) \leq n_1$. Then, for all $n \geq \max\{n_0, n_1\}$ and for all $x \leq n$, we have (if $x < n_0$) $\Phi_{e(j_0)}(x) \leq n_1 \leq n$, and (otherwise) $\Phi_{e(j_0)}(x) \leq x \leq n$.

We will not give proofs for (b) and (c), as they are only slight modifications of the proof for (a). Note that (a) and (b) are *both* special cases of (c).

Theorem 5

(a) $\forall n \in \mathbb{N} : \mathbf{EXPF} \notin \mathbf{PFPcp}_{\underline{n}}\mathbf{Ex}$.
(b) $\forall k, n \in \mathbb{N} : \mathbf{EXP}_{k+1}\mathbf{F} \notin \mathbf{PFPcp}_{w \cdot k + \underline{n}}\mathbf{Ex}$.

We will not prove (b), but only its simpler to prove special case (a).
Proof of (a). Suppose by way of contradiction otherwise as witnessed by n and $\langle h, f \rangle$. Note that $\{\sigma \mid \exists g \in \mathbf{EXPF}, \sigma \subset g\} = \mathbb{S}\mathrm{eq}$; thus, $\langle h, f\rangle \in \mathcal{T}\mathbf{Pcp}_{\underline{n}}$.

Define $g \in \mathcal{R}$ according to the following informal definition in stages. g_s denotes g as defined until before stage s.

```
$g_0 = \varepsilon$
stage $s = 0$ to $\infty$
    if $h(g_s \diamond \overline{0} \diamond \overline{0}^n) = h(g_s)$
        then $g_{s+1} = g_s \diamond \overline{1} \diamond \overline{0}^n$
        else $g_{s+1} = g_s \diamond \overline{0} \diamond \overline{0}^n$
```

Claim 1: h does not converge on g.
We show the claim by showing $\forall s : h(g_{s+1}) \neq h(g_s)$. As $\langle h, f\rangle \in \mathcal{T}\mathbf{Pcp}_{\underline{n}}$, we have for all $s \in \mathbb{N}$ and each $j \in \{0,1\}$, $\lambda i \leq n \,\text{.}\, f(g_s \diamond \overline{j} \diamond \overline{0}^i)$ is not a $\underline{n}$-path, as there is no path of length $n+1$ in $\underline{n}$; hence, $\varphi_{h(g_s \diamond \overline{j} \diamond \overline{0}^n)}(\#\text{elets}(g_s)) = j$.

If now $h(g_s \diamond \overline{0} \diamond \overline{0}^n) = h(g_s)$, then $\varphi_{h(g_{s+1})}(\#\text{elets}(g_s)) = \varphi_{h(g_s \diamond 1 \diamond \overline{0}^n)}(\#\text{elets}(g_s)) = 1 \neq \overline{0} = \varphi_{h(g_s \diamond \overline{0} \diamond \overline{0}^n)}(\#\text{elets}(g_s)) = \varphi_{h(g_s)}(\#\text{elets}(g_s))$; thus, $h(g_{s+1}) \neq h(g_s)$.

If $h(g_s \diamond \overline{0} \diamond \overline{0}^n) \neq h(g_s)$, then $h(g_{s+1}) = h(g_s \diamond \overline{0} \diamond \overline{0}^n) \neq h(g_s)$.

□ (of Claim 1)

Claim 2: $g \in \mathbf{EXPF}$.
By the construction of g, we have $\forall s : g_s \in \{0,1\}^{s \cdot (n+1)}$. Hence, to compute $g(x)$ for any given x, it suffices to execute stages 0 through x of the above algorithm to get g_{x+1}, from which $g(x)$ can then be extracted. Therefore, it suffices to show that, for all s, the stages 0 through s of the above algorithm can be done with an appropriate timebound.

Let p be a polynomial upper-bounding the runtime of h such that $\forall x : x \leq p(x)$. For any stage s, the time to execute stage s is in $\mathcal{O}(\lambda s \,\text{.}\, p(|g_s \diamond \overline{0}^{n+1}|) + p(|g_s|)) = \mathcal{O}(\lambda s \,\text{.}\, p(|g_s| + n + 1)) =^{15} \mathcal{O}(\lambda s \,\text{.}\, p(s \cdot (n+1) + n + 1)) = \mathcal{O}(\lambda s \,\text{.}\, p(s))$. Therefore, for all s, the time to execute all stages 0 to s is bounded above by $\mathcal{O}(\lambda s \,\text{.}\, (s+1) \cdot p(s)) \subseteq \mathcal{O}(\lambda s \,\text{.}\, 2^{p'(|s|)})$ for some polynomial p'.[16]

□ (of Claim 2) □ (of (a))

[15] $\mathcal{O}(|g_s|) = \mathcal{O}(\#\text{elets}(g_s))$.
[16] Find k such that $\mathcal{O}(p) = \mathcal{O}(\lambda x \,\text{.}\, x^k)$. By [RC94, §2.5, (9)], there are a, b such that $x \leq a \cdot 2^{|x|} + b$. Thus, there is are c, d, c', d' such that $\forall x : p(x) \leq c \cdot x^k + d \leq c \cdot (a \cdot 2^{|x|} + b)^k + d \leq c' \cdot 2^{k \cdot |x|} + d'$.

4 General Results

4.1 Results Mostly Not Comparing Graphs

The following theorem shows the relationship between the different learning criteria as defined in this paper.

Theorem 6. We have the following.

$$\forall G \in \mathcal{G}_{\mathrm{comp}} : \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex} = \mathcal{T}\mathbf{Pcs}_G\mathbf{Ex}. \tag{8}$$

$$\mathcal{R}\mathbf{Pcs}_{\emptyset}\mathbf{Ex} \setminus (\bigcup_{G \in \mathcal{G}} \mathbf{Pcp}_G\mathbf{Ex}) \neq \emptyset. \tag{9}$$

$$\mathcal{R}\mathbf{Pcp}_{\emptyset}\mathbf{Ex} \setminus (\bigcup_{G \in \mathcal{G}} \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}) \neq \emptyset. \tag{10}$$

$$\mathbf{Pcp}_{\emptyset}\mathbf{Ex} \setminus (\bigcup_{G \in \mathcal{G}_{\mathrm{comp}}} \mathcal{R}\mathbf{Pcs}_G\mathbf{Ex}) \neq \emptyset. \tag{11}$$

Furthermore, the separations (9) and (10) are witnessed by sets of functions such that the positive part of the separation is witnessed by a (fair) learner computable in linear time working transductively.

Our proof of (8) above is an extension of Fulk's proof of the $G = \emptyset$ case [Ful88].

Proof of (10). [17] Let $\mathcal{S} := \{g \in \mathcal{R} \mid (\overline{0} \diamond (\pi_1 \circ g), g) \in \mathbf{Pcp}_{\emptyset}\mathbf{Ex}\}$. Obviously, $\mathcal{S} \in \mathbf{LinFTdPcp}_{\emptyset}\mathbf{Ex} \subseteq \mathcal{R}\mathbf{Pcp}_{\emptyset}\mathbf{Ex}$. Let $G \in \mathcal{G}$. Suppose, by way of contradiction, $\mathcal{S} \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$ as witnessed by $\langle h_0, f_0\rangle$. Define, for all $e \in \mathbb{N}$, $S_e := \{\sigma \mid \forall i < \#\mathrm{elets}(\sigma) : \pi_1(\pi_1(\sigma(i))) = e\}$. Note that S_e is uniformly computable in e. By **KRT**, there is e such that φ_e is defined as the union over an infinite, with respect to sequence-extension strictly increasing, family of finite sequences $(\sigma_s)_{s \in \mathbb{N}}$ recursively specified as follows.

$$\sigma_0 := \emptyset; \tag{12}$$

$$\forall s : \sigma_{s+1} := \mu\sigma \in S_e \centerdot \sigma_s \subset \sigma \ \wedge \ h_0(\sigma) \neq h_0(\sigma_s). \tag{13}$$

We reason by induction that, for all s, σ_s is defined. Clear for σ_0. Let s be such that σ_s is defined. Let τ, $\tau' \in S_e$ be two extensions of σ_s such that τ and τ' differ at position $\#\mathrm{elets}(\sigma_s)$ (the first position not in σ_s), and $\#\mathrm{elets}(\sigma_s)$ is in the bottomed-out set of f_0 after f_0 gets any one of the sequences τ or τ' as input. Then, as $\langle h_0, f_0\rangle \in \mathcal{T}\mathbf{Pcp}_G$, $\varphi_{h_0(\tau)}(\#\mathrm{elets}(\sigma_s)) = \tau(\#\mathrm{elets}(\sigma_s)) \neq \tau'(\#\mathrm{elets}(\sigma_s)) = \varphi_{h_0(\tau')}(\#\mathrm{elets}(\sigma_s))$. Hence, for at least one $\sigma \in \{\tau, \tau'\}$, $h_0(\sigma) \neq h_0(\sigma_s)$.

We have a contradiction, as trivially $\varphi_e \in \mathcal{S}$ and $\langle h_0, f_0\rangle$ does not $\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$-identify φ_e. □ (FOR (10))

[17] An anonymous referee pointed out that (10) can be proven by showing that all sets in $\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$ can be reliably learned, as it is known that not all reliably learnable sets are $\mathcal{R}\mathbf{Pcp}_{\emptyset}\mathbf{Ex}$-learnable [CJSW04]. We give this proof as a particularly short examplar of many proofs omitted in Section 4.

Theorem 7. Let $G \in \mathcal{G}$. Then $\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$ is closed under computably enumerable unions.

Our proof for Theorem 7 makes use of the notion of *reliability* [Min76, BB75].

Theorem 8. We have

$$\bigcup_{G \in \mathcal{G}} \mathbf{Pcs}_G\mathbf{Ex} \subset \mathbf{Ex}. \tag{14}$$

Furthermore, the separation is witnessed by a (fair) learner computable in linear time working transductively.

4.2 Dependencies on the Countdown Graphs

Next we define a pre-order, $\leq_{CD}$, on $\mathcal{G}$. We will see that $\leq_{CD}$ characterizes relative learning-power in dependence on countdown graphs.

Definition 9. For two graphs G, G' we write $G \leq_{CD} G'$ (read: G is countdown reducible to G') iff there is a $k \in \mathcal{R}$, such that

(i) for all $y \in G$: $k(\overline{y}) \in G'$;
(ii) for all $\tau \diamond \overline{y} \in \vec{G}$ such that $\#\text{elets}(\tau) > 0$, we have $k(\tau) \rightarrow_{G'} k(\tau \diamond \overline{y})$.

Intuitively, k maps any G-path into a vertex of G'.[18] Clearly, $\leq_{CD}$ is a pre-order.

Proposition 10. Let $G, G' \in \mathcal{G}$. Let $k \in \mathcal{R}$. The following are equivalent.

(a) $G \leq_{CD} G'$ as witnessed by k;
(b) $\forall \tau \in \vec{G} : (\lambda i < \#\text{elets}(\tau)\,\boldsymbol{.}\, k(\tau[i+1])) \in \vec{G}'$.

Next we exhibit nice example countdown graphs and indicate how they compare by $\leq_{CD}$.

ω denotes the order-type of the natural numbers ordered by $\leq$, ω^{-1} denotes the order-type of the natural numbers ordered by $\geq$.

Theorem 11. There is a computable total ordering $\leq_R$ on $\mathbb{N}$ of order-type $\omega + \omega^{-1}$ such that there are no computable infinitely descending chains with respect to $\leq_R$; hence, $(\mathbb{N}, >_R)$ is a countdown graph.

For the rest of this section, let $\leq_R$ be as in Theorem 11, and let R denote the countdown graph $(\mathbb{N}, >_R)$.

Example 12. *Let $(\mathcal{N}, \leq_{\mathcal{N}}), (\mathcal{N}', \leq_{\mathcal{N}'})$ be computably related systems of ordinal notations. Then we have*

(a) $\mathcal{N} \leq_{CD} \mathcal{N}' \Rightarrow \mathcal{N}'$ gives a notation to at least all the ordinals $\mathcal{N}$ gives a notation to;

[18] Neither of mapping G vertices into G' vertices nor mapping G paths into G' paths will give us the same characterization results that we have in Theorem 13 below.

(b) $\mathcal{N} \leq_{CD} R \Leftrightarrow \mathcal{N}$ *gives a notation to all and only the ordinals* $< \omega \cdot i + j$ *for some* $i \in \{0, 1\}, j \in \mathbb{N}$*; and*
(c) $R \not\leq_{CD} \mathcal{N}$.

Theorem 13. Let $G \in \mathcal{G}_{\mathrm{comp}}$, $G' \in \mathcal{G}$. We have

$$\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex} \subseteq \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex} \Leftrightarrow G \leq_{CD} G'.$$

Next are three corollaries to Theorem 13 (or its proof). The first two are regarding the other restricted learnability notions of the present paper. The third is our hierarchy theorem for ordinal notations.

Corollary 14. *Let* $G \in \mathcal{G}_{\mathrm{comp}}$. *We have*

$$\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex} \setminus \bigcup_{\substack{G' \in \mathcal{G}_{\mathrm{comp}} \\ G \not\leq_{CD} G'}} \mathbf{Pcs}_{G'}\mathbf{Ex} \neq \emptyset.$$

Next is a characterization of the graph dependence of relative learning power for the restricted learning criteria not covered by Theorem 13.

Corollary 15. *For all* $G, G' \in \mathcal{G}_{\mathrm{comp}}$ *we have*

$$\begin{aligned} G \leq_{CD} G' &\Leftrightarrow \mathcal{R}\mathbf{Pcp}_G\mathbf{Ex} \subseteq \mathcal{R}\mathbf{Pcp}_{G'}\mathbf{Ex} && (15)\\ &\Leftrightarrow \mathcal{R}\mathbf{Pcs}_G\mathbf{Ex} \subseteq \mathcal{R}\mathbf{Pcs}_{G'}\mathbf{Ex} && (16)\\ &\Leftrightarrow \mathbf{Pcp}_G\mathbf{Ex} \subseteq \mathbf{Pcp}_{G'}\mathbf{Ex} && (17)\\ &\Leftrightarrow \mathbf{Pcs}_G\mathbf{Ex} \subseteq \mathbf{Pcs}_{G'}\mathbf{Ex}. && (18) \end{aligned}$$

Recall that, from Section 2, for a graph $G \in \mathcal{G}$ and $m \in G$, we ambiguously use m to refer to the countdown-graph $\{n \in G \mid m \to^+ n\}$. For two sets M, N we write $M \# N$ iff $(M \not\subseteq N \ \wedge\ N \not\subseteq M)$.

Corollary 16. *Let* $(\mathcal{N}, \leq_{\mathcal{N}})$ *be a computably related system of ordinal notations. Let* $u, v \in \mathcal{N}$. *Then we have*

$$\begin{aligned} u <_{\mathcal{N}} v &\Leftrightarrow u <_{CD} v && (19)\\ &\Leftrightarrow \mathcal{T}\mathbf{Pcp}_u\mathbf{Ex} \subset \mathcal{T}\mathbf{Pcp}_v\mathbf{Ex}. && (20) \end{aligned}$$

Furthermore, if $\mathcal{N}$ *gives a notation to at least all ordinals* $< \omega \cdot 2$*, then*

$$\mathcal{T}\mathbf{Pcp}_{\mathcal{N}}\mathbf{Ex} \# \mathcal{T}\mathbf{Pcp}_R\mathbf{Ex}. \qquad (21)$$

References

[ACJS04] Ambainis, A., Case, J., Jain, S., Suraj, M.: Parsimony hierarchies for inductive inference. Journal of Symbolic Logic 69, 287–328 (2004)

[AZ07] Akama, Y., Zeugmann, T.: Consistent and coherent learning with δ-delay. Technical Report TCS-TR-A-07-29, Hokkaido Univ. (October 2007)

[Bār74] Bārzdiņš, J.: Inductive inference of automata, functions and programs. In: Int. Math. Congress, Vancouver, pp. 771–776 (1974)

[BB75] Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. Information and Control 28, 125–155 (1975)

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

[CJSW04] Case, J., Jain, S., Stephan, F., Wiehagen, R.: Robust learning – rich and poor. Journal of Computer and System Sciences 69, 123–165 (2004)

[CK08] Case, J., Kötzing, T.: Dynamically delayed postdictive completeness and consistency in machine inductive inference (2008), `http://www.cis.udel.edu/~case/papers/PcpPcsDelayTR.pdf`

[CKP07] Case, J., Kötzing, T., Paddock, T.: Feasible iteration of feasible learning functionals. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 34–48. Springer, Heidelberg (2007)

[FS93] Freivalds, R., Smith, C.: On the role of procrastination in machine learning. Information and Computation 107(2), 237–271 (1993)

[Ful88] Fulk, M.: Saving the phenomena: Requirements that inductive machines not contradict known data. Inform. and Comp. 79, 193–209 (1988)

[JORS99] Jain, S., Osherson, D., Royer, J., Sharma, A.: Systems that Learn: An Introduction to Learning Theory, 2nd edn. MIT Press, Cambridge (1999)

[LV97] Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, Heidelberg (1997)

[Min76] Minicozzi, E.: Some natural properties of strong identification in inductive inference. In: Theoretical Computer Science, pp. 345–360 (1976)

[Pit89] Pitt, L.: Inductive inference, DFAs, and computational complexity. In: Jantke, K.P. (ed.) AII 1989. LNCS, vol. 397, pp. 18–44. Springer, Heidelberg (1989)

[Rog67] Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw Hill, New York (1967) (Reprinted by MIT Press, Cambridge, Massachusetts, 1987)

[RC94] Royer, J., Case, J.: Subrecursive Programming Systems. In: Progress in Theoretical Computer Science, Birkhäuser (1994)

[SSV04] Sharma, A., Stephan, F., Ventsov, Y.: Generalized notions of mind change complexity. Information and Computation 189, 235–262 (2004)

[Wie76] Wiehagen, R.: Limes-erkennung rekursiver Funktionen durch spezielle Strategien. Elek. Informationverarbeitung und Kyb. 12, 93–99 (1976)

[Wie78] Wiehagen, R.: Zur Theorie der Algorithmischen Erkennung. Dissertation B. Humboldt University of Berlin (1978)