Difficulties in Forcing Fairness of Polynomial Time Inductive Inference

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. There are difficulties obtaining *fair* feasibility from polynomial time updated language learning in the limit from positive data. Pitt 1989 noted that *un*fair delaying tricks can achieve polynomial time updates but with no feasibility constraint on the whole learning process. In this context Yoshinaka 2009 makes a useful list of properties or restrictions towards true feasibility. He also provides interesting examples of *fair* polynomial time algorithms featuring particular uniformly polynomial time decidable hypothesis spaces, and each of his algorithms satisfies several of his properties.

Yoshinaka claims that the combination of the three restrictions on polynomial time learners of *consistency* (which we call herein *postdictive completeness*), *conservativeness* and *prudence* is restrictive enough to *stop* Pitt's delaying tricks from working.

The present paper refutes the claim of the previous paragraph in three settings. In the setting of uniformly polynomial time decidable hypothesis spaces with a few effective closure properties, the three restrictions allow maximal *un*fairness. The other two settings involve certain *other* uniformly decidable hypothesis spaces and general language learning hypothesis spaces. In each of these settings, the three restrictions forbid *some*, but *not all* Pitt-style delaying tricks.

Inside the proofs of each of our theorems asserting that the three restrictions do *not* forbid some or all delaying tricks, the witnessing learners can be seen to explicitly employ delaying tricks.

1 Introduction

For a class of (at least computably enumerable) languages $\mathcal{L}$ and an algorithmic learning function h, we say that h TxtEx-*learns* $\mathcal{L}$ [Gol67, JORS99] iff, for each $L \in \mathcal{L}$, for every function T enumerating (or presenting) all and only the elements of L (with or without pauses), as h is fed the succession of values $T(0), T(1), \ldots$, it outputs a corresponding succession of programs $p(0), p(1), \ldots$ from some hypothesis space, and, for some i_0, for all $i \geq i_0$, $p(i)$ is a correct program for L, and $p(i+1) = p(i)$. The function T as just above is called a *text* or *presentation* for L. TxtEx-learning is also called *learning in the limit from positive data*.

We say that h TxtEx-*learns* a $\mathcal{L}$ *in polynomial time* iff there is a polynomial Q such that, for each i, h computes $p(i)$ within time $Q(|T(0), T(1), \ldots, T(i-1)|)$.

R. Gavaldà et al. (Eds.): ALT 2009, LNAI 5809, pp. 263–277, 2009.

Pitt [Pit89] notes (in a slightly different context) that such a definition of polynomial time learning may not give one any feasibility restriction on the total time for successful learning. Here is informally why. Suppose h is any TxtEx-learner. Then, for suitable polynomial Q, a variant of learner h can *delay* outputting significant conjectures based on data σ until it has seen a much larger sequence of data τ so that $Q(|\tau|)$ is enough time for h to think about σ as long as it needs.

Pitt [Pit89] discusses some possible ways to forbid such *unfair delaying tricks*. More recently, Yoshinaka [Yos09] compiled a very useful list of properties to help toward achieving *fairness* and efficiency in polynomial time learners, *including* to avoid Pitt-style delaying tricks. In the second part of [Yos09], Yoshinaka provides a number of interesting example *fair* polynomial time learners each satisfying several of these properties. In each of his *example* algorithms, the associated hypothesis space is *uniformly* polynomial time decidable.[1] In the present paper, we focus, for polynomial time learners, on three of Yoshinaka's properties: Postdictive completeness[2], conservativeness, and prudence. *Postdictive completeness* [Bār74, BB75, Wie76, Wie78] requires that each hypothesis output by a learner correctly postdicts the input data on which that hypothesis is based. *Conservativeness* [Ang80] requires that each hypothesis may be changed only if it fails to *pre*dict a new datum. *Prudence* [Wei82, OSW86] requires each output hypothesis has to be for a target that the learner actually learns.

Yoshinaka [Yos09] claims that, for efficient learning in the limit from positive data, the combination of *postdictive completeness*, *conservativeness* and *prudence* is restrictive enough to prevent all Pitt-style delaying tricks.

In the present paper, in several different settings (settings mostly as to kind of hypothesis spaces), we refute the claim of the immediately above paragraph.

In *one* of our settings, uniformly polynomial time decidable hypothesis spaces with a few effective closure properties,[3,4] the three restrictions allow maximal

[1] These spaces are such that there is a polynomial Q and an algorithm so that, from both an hypothesis i and an object x, the algorithm returns, within time Q(|i|,|x|) a correct decision as to whether x is in the language defined by hypothesis i.

[2] In the prior literature, except for [Ful88] and [CK08a, CK08b], what we call postdictive completeness is called *consistency*.

[3] These effective closure properties pertain to obtaining finite languages and modifications of languages by finite languages.

[4] The particular uniformly polynomial time hypothesis spaces Yoshinaka employs in the second half of [Yos09] do not have our few effective closure properties, but his algorithms would work essentially unchanged were one to extend his hypothesis spaces to ones with our few effective closure properties. Then his algorithms would not search or use the new hypotheses and would not learn any more languages. The space of CFGs with Prohibition discussed below in this section and in Section 2.1 further below would work as such an extension of both Yoshinaka's hypothesis spaces. Yoshinaka does mention the possibility of extending his hypothesis spaces to provide an hypothesis for Σ^*. We did not examine whether we could, in some cases, work with such an extension instead of our few effective closure properties. We also did not examine whether we can modify our (to be mentioned shortly) Theorem 13 to cover *just* his particular hypothesis spaces.

*un*fairness (Theorem 13 in Section 3 below).[5] An example of our uniformly polynomial time decidable hypothesis spaces (with a few effective closure properties) employs efficiently coded DFAs. Another example employs an (also efficiently coded), interesting extension of context free grammars (CFGs), called *CFGs with Prohibition* [Bur05]. This latter example is treated in more detail after Definition 2 in Section 2.1 below.

In all of our settings, any combination of just the two restrictions of conservativeness and prudence allows for *arbitrary* delaying tricks (Theorem 18 in Section 5).

In each of our two settings *besides* the first setting of uniformly polynomial time decidable hypothesis spaces (with a few effective closure properties), postdictive completeness *does* strictly forbid *some*, but *not all* Pitt-style delaying tricks.[6]

The two residual settings are: 1. TxtEx-learning with *certain other* uniformly decidable hypothesis spaces (Section 4), e.g., the (efficiently coded), explicitly clocked, multi-tape Turing Machines which halt in linear time [RC94, Chapter 6][7] and 2. TxtEx-learning with a general purpose hypothesis space (Section 5).

The theorems that postdictive completeness forbids *some* delaying tricks in these last two settings are: Theorem 14 in Section 4 and Theorem 19 in Section 5.

The theorems that postdictive completeness does *not* forbid *all* delaying tricks in these last two settings are: Theorem 15 in Section 4 and Theorem 22 in Section 5.

Inside the proofs of each of our theorems asserting that the three restrictions do *not* forbid some or all delaying tricks, the witnessing learners can be seen to explicitly employ delaying tricks. Note that many of our delaying tricks involve "overlearning," i.e., learning a larger class of languages than required.

To avoid having to define successively each of a large number of criteria of successful learning (e.g., restricted variants of TxtEx-learning), we provide a modular approach to presenting such definitions. In our modular approach, we define names for "pieces" of our criteria (Section 2.1). Then, after that, each criterion needed is named by stringing together the relevant names of its pieces. For example, unrestricted TxtEx-learning in the present section will be later

[5] It is an interesting open question, though, for our uniformly polynomial time decidable hypothesis spaces, whether the combination of postdictive completeness, conservativeness and prudence is so restrictive, that, any class of languages TxtEx-learnable employing such an hypothesis space and with those three restrictions, is also TxtEx-learnable with an intuitively fair, *different* polynomial time learner respecting all three restrictions.

[6] That is, in our residual two settings, of the three restrictions, postdictive completeness *does* improve fairness, *but* there *can* still be some residual unfair delaying tricks.

For these residual settings, we did not examine the question of whether adding onto postdictive completeness, conservativeness and/or prudence, provides better degree of avoidance of delaying tricks than postdictive completeness alone. Again: we already know, though, that all three restrictions do *not* avoid *all* delaying tricks.

[7] The associated class is *not* uniformly *polynomial time* decidable, by [RC94, Theorem 6.5].

named, **TxtGEx**.[8] A similar modular approach appears already in [CK08a, CK08b].

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, $\{0,1,2,\ldots\}$. We do not distinguish between natural numbers and their *dyadic* representations as strings.[9]

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. Since 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 $\max\{1, \mathrm{size}(w)\}$.[10]

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 $A \setminus B := \{a \in A \mid a \notin B\}$, $\overline{A} := \mathbb{N} \setminus A$ and $\mathrm{Pow}(A)$ be the power set of A.

The quantifier $\forall^\infty x$ means "for all but finitely many x", the quantifier $\exists^\infty x$ means "for infinitely many x". For any set A, $\mathrm{card}(A)$ denotes the cardinality of A.

$\mathfrak{P}$ and $\mathfrak{R}$ denote, respectively, the set of all partial and of all total functions $\mathbb{N} \to \mathbb{N} \cup \{\#\}$. dom and range denote, respectively, domain and range of a given function.

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

A function ψ is *partial computable* iff there is a deterministic, multi-tape Turing machine computing ψ. $\mathcal{P}$ and $\mathcal{R}$ denote, respectively, the set of all partial computable and the set of all total (partial) computable functions $\mathbb{N} \to \mathbb{N}$. If $f \in \mathfrak{P}$ is defined for some argument x, then we denote this fact by $f(x)\downarrow$, and we say that f on x *converges*.

We say that $f \in \mathfrak{P}$ *converges to* p iff $\forall^\infty x : f(x)\downarrow = p$; we write $f \to p$ to denote this.[11]

φ^{TM} is the fixed programming system from [RC94, Chapter 3] for the partial computable functions $\mathbb{N} \to \mathbb{N}$. This system is based on deterministic, multi-tape

[8] In general, *standard* inductive inference criteria names will be changed to slightly different names in our modular approach.

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

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

[11] $f(x)$ converges should not be confused with f converges *to*.

Turing machines (TMs). In this system the TM-programs are *efficiently* given numerical names or codes.[12] Φ^{TM} denotes the TM step counting complexity measure also from [RC94, Chapter 3] and associated with φ^{TM}. In the present paper, we employ a number of complexity bound results from [RC94, Chapters 3 & 4] regarding $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$. These results will be clearly referenced as we use them. For simplicity of notation, hereafter, we write (φ, Φ) for $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$. 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.

The symbol # is pronounced *pause* and is used to symbolize "no new input data" in a text.

Note that all (partial) computable functions are $\mathbb{N} \to \mathbb{N}$. Whenever we want to consider (partial) computable functions on objects like finite sequences or finite sets, we assume those objects to be efficiently coded as natural numbers. We give such codings for finite sequences and finite sets below.

For all p, W_p denotes the computably enumerable (ce) set $\text{dom}(\varphi_p)$. $\mathcal{E}$ denotes the set of all ce sets. We say that e is an index (in W) for W_e.

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.

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.

We identify any function $f \in \mathfrak{P}$ with its graph $\{\langle x, f(x) \rangle \mid x \in \mathbb{N}\}$.

A *finite sequence* is a mapping with a finite initial segment of $\mathbb{N}$ as domain (and range, $(\mathbb{N} \cup \{\#\})$). $\emptyset$ denotes the empty sequence (and, also, the empty set). The set of all finite sequences is denoted by $\mathbb{S}\text{eq}$. 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. $\#\text{elets}(\sigma)$ denotes the number of elements in a finite sequence σ, that is, the cardinality of its domain.

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 $\mathbb{N}$. D with or without decorations ranges over finite subsets of $\mathbb{N}$.

Following [LV97], we fix a coding $\langle \cdot \rangle_{\mathbb{S}\text{eq}}$ of all sequences into $\mathbb{N} \cup \{\#\}$ $(= \{0,1\}^* \cup \{\#\})$ – 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

$$\lambda k, v_1, \ldots, v_k \,.\, \langle v_1, \ldots, v_k \rangle_{\mathbb{S}\text{eq}}$$

is

$$\mathcal{O}(\lambda k, v_1, \ldots, v_k \,.\, \sum_{i=1}^{k} |v_i|).$$

[12] 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.

Therefore, the size of the codeword is also linear in the size of the elements: $\lambda k, v_1, \ldots, v_k . |\langle v_1, \ldots, v_k \rangle_{\mathbb{S}\mathrm{eq}}|$ is $\mathcal{O}(\lambda k, v_1, \ldots, v_k . \sum_{i=1}^{k} |v_i|)$.[13]

Furthermore,

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

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 $g(0), \ldots, g(i-1)$.

D, with and without decorations, ranges over finite sets. We fix the following 1-1 coding for all finite subsets of $\mathbb{N}$. For each non-empty finite set $D = \{x_0 < \ldots < x_n\}$, $\langle x_0, \ldots, x_n \rangle_{\mathbb{S}\mathrm{eq}}$ is the code for D and $\langle \rangle_{\mathbb{S}\mathrm{eq}}$ is the code for $\emptyset$.

Henceforth, we will many times identify a finite set D with its code number. However, when we employ expressions such as $x \in D$, $\mathrm{card}(D)$, $\max(D)$ and $D \subset D'$, we consider D and D' as *sets*, not as numbers.

For each (possibly infinite) sequence q, let $\mathrm{content}(q) = (\mathrm{range}(q) \setminus \{\#\})$.

We define $\mathrm{LinPrograms} = \{e \mid \exists a, b, \forall x : \Phi_e(x) \leq a|x| + b\}$ and $\mathrm{PolyPrograms} = \{e \mid \exists p \text{ polynomial } \forall x \in \mathbb{N} : \Phi_e(x) \leq p(|x|)\}$. Furthermore, for let $\mathbf{LinF} = \{\varphi_e \mid e \in \mathrm{LinPrograms}\}$ and $\mathbf{PF} = \{\varphi_e \mid e \in \mathrm{PolyPrograms}\}$.

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

With log we denote the floor of the base-2 logarithm, with the exception of $\log(0) = 0$.

For all e, x, t, we write $\varphi_e(x)\downarrow_t$ iff $\Phi_e(x) \leq t$. Furthermore, we write

$$\forall e, x, t : \varphi_e(x)\downarrow_t = \begin{cases} \varphi_e(x), & \text{if } \Phi_e(x) \leq t; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The following lemma is used in many of our detailed proofs. The present paper, because of space limitations, omits many details of proofs. Nonetheless, we still include this lemma herein to give the reader some intuitions as to how to manage some missing details.

Lemma 1. Regarding time-bounded computability, we have the following.

- Equality checks and log are computable in linear time [RC94, Lemma 3.2].
- Conditional definition is computable in a time polynomial in the runtimes of its defining programs [RC94, Lemma 3.14].
- Bounded minimizations, and, hence, bounded maximizations are computable in a time polynomial in the runtimes of its defining programs [RC94, Lemma 3.15].
- Boolean combinations of predicates computable in polytime are computable in polytime [RC94, Lemma 3.18].
- From [RC94, Corollary 3.7], we have that $\lambda e, x, t . \varphi_e(x)\downarrow_{|t|}$ and $\lambda e, x, t, z . \varphi_e(x)\downarrow_{|t|} = z$ are computable in polynomial time.

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

– Our coding of finite sequences easily gives that the following functions are linear time computable. $\forall x : 1 \leq \text{length}(\bar{x})$, $\lambda\langle\sigma\rangle_{\mathbb{S}\text{eq}}.\#\text{elets}(\sigma)$ and $\lambda\langle\sigma\rangle_{\mathbb{S}\text{eq}}, i.\begin{cases}\sigma(i), & \text{if } i < \#\text{elets}(\sigma);\\ 0, & \text{otherwise.}\end{cases}$
– Our coding above of finite sets enables content to be computable in polynomial time.[14]

2.1 Learning Criteria Modules

In this section we give our modular definition of what a learning criterion is. After that we will put the modules together to obtain the actual criteria we need. As noted above, all *standard* inductive inference criteria names will be changed to slightly different names in our modular approach.

Definition 2. An *effective numbering of* `ce` *languages* is a function $V : \mathbb{N} \to \mathcal{E}$ such that there is a function $f \in \mathcal{P}$ with $\forall e, x : f(e, x)\downarrow \Leftrightarrow x \in V(e)$.[15] For such numberings V, for each e, we will write V_e instead of $V(e)$, and call e an *index* or an *hypothesis* (in V) for V_e. Recall that we identify functions with their graphs. Therefore, φ and any other indexing for partial computable functions is considered an effective numbering. We use effective numberings as hypothesis spaces for our learners.

We will sometimes require that $\{V_e \mid e \in \mathbb{N}\}$ is effectively closed under some finite modifications; precisely we will sometimes require

$$\begin{array}{l}\exists s_\cap \in \mathcal{R} : \forall e, D : V_{s_\cap(e,D)} = V_e \cap D \wedge\\ \exists s_\cup \in \mathcal{R} : \forall e, D : V_{s_\cup(e,D)} = V_e \cup D \wedge\\ \exists s_\setminus \in \mathcal{R} : \forall e, D : V_{s_\setminus(e,D)} = V_e \setminus D.\end{array} \tag{3}$$

Note that, in practice, many effective numberings of `ce` languages allow $s_\cap, s_\cup$ and $s_\setminus$ as in (3) to be computable in polynomial or even linear time.

Effective numberings include the following important examples.

– W ((3) holds).
– A canonical numbering of all regular languages, represented by efficiently coded DFAs (where membership is trivially uniformly polynomial time decidable and (3) holds).
– For each pair of context free grammars (CFGs) G_0, G_1, we efficiently code (G_0, G_1) to be an index for $(L(G_0) \setminus L(G_1))$. Then the resulting numbering, in particular, has an index for each context free language. Furthermore,

[14] This computation involves sorting. Selection sort can be done in quadratic time in the RAM model [Knu73], and adding an extra linear factor to translate from RAM complexity to deterministic multi-tape TM complexity [vEB90], we get selection sort in cubic (and, hence, polynomial) time measured by Φ^{TM}.

[15] Note that such a numbering does not necessarily need to be onto, i.e., a numbering might only number some of the `ce` languages, leaving out others.

membership is uniformly polynomial time decidable [HU79, Sch91], and (3) holds. As noted above, these grammars are called CFGs with Prohibition in [Bur05].[16]

Definition 3. Any set $\mathcal{C} \subseteq \mathcal{P}$ is a *learner admissibility restriction*. Intuitively, a learner admissibility restriction defines which functions are admissible as potential learners.

Two typical learner admissibility restrictions are $\mathcal{P}$ and $\mathcal{R}$. When denoting criteria with $\mathcal{P}$ as the learner admissibility restriction, we will omit $\mathcal{P}$.

Definition 4. Any function from $\mathcal{E}$ to $\mathrm{Pow}(\mathfrak{R})$ is called a *target presenter* for the ce languages. The only target presenter used in this paper is $\mathbf{Txt} : \mathcal{E} \to \mathrm{Pow}(\mathfrak{R}), L \mapsto \{\rho \in \mathfrak{R} \mid \mathrm{content}(\rho) = L\}$.

Definition 5. Every computable operator $\mathfrak{P} \times \mathfrak{R} \to \mathfrak{P}^2$ is called a *sequence generating operator*.[17] Intuitively, a sequence generating operator defines how learner and presentation interact to generate two infinite sequences, one for learner-outputs (we call this sequence the *learner-sequence*) and one for learnee-outputs.

For any sequence generating operator β, we define β_1 and β_2 such that $\beta = \lambda h, g\,.\,(\beta_1(h,g), \beta_2(h,g))$.

We define the following sequence generating operators.

- Goldstyle: $\mathbf{G} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}, (h,g) \mapsto (\lambda i\,.\,h(g[i]), g)$.
- [JORS99] Set-driven: $\mathbf{Sd} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}, (h,g) \mapsto (\lambda i\,.\,h(\mathrm{content}(g[i])), g)$.
- [JORS99] Partly set-driven: $\mathbf{Psd} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}, (h,g) \mapsto (\lambda i\,.\,h(\mathrm{content}(g[i]), i), g)$.

Definition 6. Every subset of $\mathcal{P}^2$ is called a *sequence acceptance criterion*. Intuitively, a sequence acceptance criterion defines what identification-sequences are considered a successful identification of a target. Any two such sequence acceptance criteria δ and δ' can be combined by intersecting them. For ease of notation we write $\delta\delta'$ instead of $\delta \cap \delta'$.

For each effective numbering of some ce languages V, we define the following sequence acceptance criteria.

- Explanatory: $\mathbf{Ex}_V = \{(p,q) \in \mathcal{P}^2 \mid \exists p' : p \to p' \wedge V_{p'} = \mathrm{content}(q)\}$.
- Postdictive Completeness: $\mathbf{Pcp}_V = \{(p,q) \in \mathfrak{R}^2 \mid \forall i : \mathrm{content}(q[i]) \subseteq V_{p(i)}\}$.
- Conservativeness: $\mathbf{Conv}_V = \{(p,q) \mid \forall i : p(i) \neq p(i{+}1) \Rightarrow \mathrm{content}(q[i{+}1]) \not\subseteq V_{p(i)}\}$.

For any given target presenter α and a sequence generating operator β, 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*:

[16] Intuitively, G_0 may "generate" an element, and G_1 can correct it or exclude it.
The concept of Prohibition Grammars is generalized in [CCJ09, CR09] and, there, they are called Correction Grammars.

[17] Essentially, *these* computable operators are the recursive operators of [Rog67] but with two arguments and two outputs and restricted to the indicated domain.

$\mathcal{T}\delta := \{h \in \mathcal{P} \mid \forall T \in \text{range}(\alpha) : \beta(h,T) \in \delta\}$. We then speak of "total ... ." For example, *total* postdictive completeness, i.e., $\mathcal{T}\mathbf{Pcp}_V$, requires postdictive completeness on *any* input data, including input data not necessarily taken from a target to be learned.[18]

Definition 7. A *learning criterion* (for short, *criterion*) is a 4-tuple consisting of a learner admissibility restriction, a target presenter, a sequence generating operator and a sequence acceptance criterion. Let $\mathcal{C}$, α, β, δ be, respectively, a learner admissibility restriction, a target presenter, a sequence generating operator and a sequence acceptance criterion. For $h \in \mathfrak{P}, L \in \text{dom}(\alpha)$, we say that h $(\mathcal{C},\alpha,\beta,\delta)$-*learns* L iff: $h \in \mathcal{C}$ and, for all $T \in \alpha(L)$, $\beta(h,T) \in \delta$. For $h \in \mathfrak{P}$ and $\mathcal{L} \subseteq \text{dom}(\alpha)$ we say that h $(\mathcal{C},\alpha,\beta,\delta)$-*learns* $\mathcal{L}$ iff, for all $L \in \mathcal{L}$, h $(\mathcal{C},\alpha,\beta,\delta)$-learns L. The set of $(\mathcal{C},\alpha,\beta,\delta)$-learnable sets of computable functions is

$$\mathcal{C}\alpha\beta\delta := \{\mathcal{L} \subseteq \mathcal{E} \mid \exists h \in \mathfrak{P} : h \ (\mathcal{C},\alpha,\beta,\delta)\text{-learns } \mathcal{L}\}. \tag{4}$$

We refer to the sets $\mathcal{C}\alpha\beta\delta$ as in (4) as *learnability classes*. Instead of writing the tuple $(\mathcal{C},\alpha,\beta,\delta)$, we will ambiguously write $\mathcal{C}\alpha\beta\delta$. For $h \in \mathfrak{P}$, the set of all computable learnees $(\mathcal{C},\alpha,\beta,\delta)$-learned by h is denoted by $\mathcal{C}\alpha\beta\delta(h) := \{L \in \mathcal{E} \mid h \ (\mathcal{C},\alpha,\beta,\delta)\text{-learns } L\}$.

Definition 8. We let Id be the function mapping a learning criterion $(\mathcal{C},\alpha,\beta,\delta)$ to the set $\mathcal{C}\alpha\beta\delta$, as defined in (4). We define two versions of prudent learning as follows. For all $\mathcal{C}$, α, β, δ, V, respectively, a learner admissibility restriction, a target presenter, a sequence generating operator, a sequence acceptance criterion and an effective numbering of `ce` languages, we let

$$\mathbf{Prud}_V(\mathcal{C},\alpha,\beta,\delta) = \{\mathcal{L} \subseteq \text{dom}(\alpha) \mid \begin{array}{l}\exists h \in \mathcal{C} : \mathcal{L} \subseteq \alpha\beta\delta(h) \wedge \\ \forall t \in \mathcal{L}, \forall T \in \alpha(t), \forall i : V_{\beta_1(h,T)(i)} \in \mathcal{L}\},\end{array}$$

and

$$\mathcal{T}\mathbf{Prud}_V(\mathcal{C},\alpha,\beta,\delta) = \{\mathcal{L} \subseteq \text{dom}(\alpha) \mid \begin{array}{l}\exists h \in \mathcal{C} : \mathcal{L} \subseteq \alpha\beta\delta(h) \wedge \\ \forall e \in \text{range}(h) : V_e \in \mathcal{L}\}.\end{array}$$

For $\mathcal{D} \in \{\text{Id}, \mathbf{Prud}, \mathcal{T}\mathbf{Prud}\}$, a learning criterion C and a learner h, we write $\mathcal{D}C$ instead of $\mathcal{D}(C)$; further, we let $\mathcal{D}C(h)$ denote the set of all targets learnable by h for criterion $\mathcal{D}C$.

We subscript an entire learning criterion with an effective numbering V to change all restrictions to expect hypotheses from V.

For example, we write the criterion of TxtEx-learning, with V-indices for the hypothesis space, as $\mathbf{TxtGEx}_V$. However, $\mathbf{TxtGEx}_V$ *with* the three restrictions of total postdictive completeness, total conservativeness *and* prudence (not total prudence) in our modular notation is written $\mathbf{Prud}\mathcal{T}\mathbf{Pcp}\mathcal{T}\mathbf{Conv}\mathbf{TxtGEx}_V$, which we abbreviate as $\mathbf{Prud}\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtGEx}_V$. If, instead, we wanted this criterion *but* with *total* prudence in the place of prudence, it could be written $\mathcal{T}\mathbf{Prud}\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtGEx}_V$.

[18] Note that, while Yoshinaka [Yos09] essentially *defines* for his postdictive completeness, conservativeness and prudence the *total* kinds, his interesting algorithms for which he claims these three restrictions satisfy only the non-total versions.

3 Uniformly Polytime Decidable Hypothesis Spaces

For this section, we let U be an arbitrary, fixed effective numbering of some ce languages and such that $\lambda e, x . x \in U_e$ is computable in polynomial time. We call such a numbering *uniformly polynomial time computable*. Further suppose there is an $r \in \mathbf{PF}$ such that $\forall D : U_{r(D)} = D$ and suppose (3) in Section 2.1 holds for U. Codings for DFAs or CFGs with Prohibition are example such Us (see Definition 2 in Section 2.1).

Interestingly, Theorem 10 below says that, every conservative learner employing hypothesis space U can, without loss of generality, be assumed to be polynomial time, postdictively complete, prudent and set-driven. This leads to the main theorem in this section (Theorem 13), that, for hypothesis space U, no combination of the three restrictions of postdictive completeness, conservativeness and prudence will forbid *arbitrary* delaying tricks.

First, we show with a lemma how we can delay set-driven learning and preserve postdictive completeness and conservativeness. We use this lemma for the succeeding theorem.

Lemma 9. We have

$$\mathbf{PF}\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtSdEx}_U = \mathcal{T}(\mathbf{PcpConv})\mathbf{TxtSdEx}_U.$$

Proof: "$\subseteq$" is immediate. Let $h \in \mathcal{R}$ and $\mathcal{L} = \mathcal{T}(\mathbf{PcpConv})\mathbf{TxtSdEx}_U(h)$. Fix a φ-program for h.

Let P be a computable predicate such that

$$\forall D', D : P(D', D) \Leftrightarrow [D' \subseteq D \ \wedge \ h(D')\downarrow_{|D|} \ \wedge \ D \subseteq U_{h(D')}]. \tag{5}$$

By Lemma 1, there is $h' \in \mathbf{PF}$ such that

$$\forall D : h'(D) = \begin{cases} h(D'), & \text{if there is } \leq\text{-max } D' \leq |D| \text{ such that } P(D', D); \\ r(D), & \text{otherwise.} \end{cases} \tag{6}$$

We omit the proof that this delaying construction works. □

Theorem 10. We have

$$\mathcal{T}\mathbf{PrudPF}\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtSdEx}_U = \mathbf{TxtGConvEx}_U.$$

Proof: "$\subseteq$" is trivial. Regarding "$\supseteq$": First we apply Proposition 16 to get total conservativeness. Then we use Theorem 17 to obtain total postdictive completeness. We use Theorem 20 to make the learner set-driven. By Lemma 9, such a learner can be delayed to be computable in polynomial time. By Proposition 21, this learner is automatically totally prudent. □

Proposition 11 just below shows that any learner can be assumed partially set-driven, and, importantly, the transformation of a learner to a partially set-driven

learner preserves prudence. The proposition and its proof are somewhat analogous to [JORS99, Proposition 5.29] and its proof. However, our proof, unlike that of [JORS99, Proposition 5.29], does not require the hypothesis space to be paddable.

Proposition 11. Let $\mathcal{D} \in \{\mathrm{Id}, \mathbf{Prud}, \mathcal{T}\mathbf{Prud}\}$. We have

$$\mathcal{D}\mathbf{TxtPsdEx}_U = \mathcal{D}\mathbf{TxtGEx}_U.$$

We can delay partially set-driven learning just as we delayed set-driven learning in Lemma 9, resulting in Lemma 12 just below.

Lemma 12. We have

$$\mathbf{PrudPF}\mathcal{T}\mathbf{PcpTxtPsdEx}_U = \mathbf{PrudTxtGEx}_U \text{ and}$$
$$\mathcal{T}\mathbf{PrudPF}\mathcal{T}\mathbf{PcpTxtPsdEx}_U = \mathcal{T}\mathbf{PrudTxtGEx}_U.$$

The next theorem is the main result of the present section. As noted in Section 1 above, it says that the three restrictions of postdictive completeness, conservativeness and prudence allow maximal *un*fairness — within the current setting of polynomial time decidable hypothesis spaces.

Theorem 13. Let $\delta \in \{\mathcal{R}^2, \mathbf{Pcp}, \mathbf{Conv}, \mathbf{PcpConv}\}$, $\mathcal{D} \in \{\mathrm{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathrm{Id}, \mathcal{T}\mathbf{Prud}\}$. Then

$$\mathcal{D}\mathbf{PFTxtG}\delta\mathbf{Ex}_U = \mathcal{D}\mathbf{TxtG}\delta\mathbf{Ex}_U \text{ and}$$
$$\mathcal{D}'\mathbf{PF}\mathcal{T}\delta\mathbf{TxtGEx}_U = \mathcal{D}'\mathcal{T}\delta\mathbf{TxtGEx}_U.$$

Proof: Use Theorem 10, as well as Theorem 18 and Lemma 12. □

4 Other Uniformly Decidable Hypothesis Spaces

For this section, we let $V : \mathbb{N} \to \mathcal{E}$ range over effective numberings of some ce languages such that $\lambda e, x\,.\,x \in V_e$ is computable (we call such a numbering *uniformly decidable*). Further suppose, for each such V, there is $r \in \mathcal{R}$ such that $\forall D : V_{r(D)} = D$.[19]

Example such numberings V include the classes of all linear time, polynomial time, ... decidable languages (not *uniformly* linear time, polynomial time, ... decidable), each represented by efficiently numerically coded programs in a suitable subrecursive programming system for deciding languages [RC94].

For uniformly decidable hypothesis spaces, we get mixed results. We have already seen from Theorem 13 in Section 3 above that there are uniformly decidable hypothesis spaces where we have arbitrary delaying for all combinations of postdictive completeness, conservativeness and prudence. Next is the first main

[19] Note that, in practice, many effective numberings of some ce languages allow r to be computable in polynomial or even linear time.

theorem of the present section. It states that there are *other* uniformly decidable hypothesis spaces such that postdictive completeness, with or without any of conservativeness and prudence, forbids *some* delaying tricks. By contrast, according to Theorem 18 in Section 5, any combination of just the two restrictions of conservativeness and prudence allows for *arbitrary* delaying tricks.

Theorem 14. There *exists* a uniformly decidable numbering V such that, for each $\delta \in \{\mathcal{R}^2, \mathbf{Pcp}, \mathbf{Conv}, \mathbf{PcpConv}\}$, $\mathcal{D} \in \{\mathrm{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathrm{Id}, \mathcal{T}\mathbf{Prud}\}$,

$$\mathcal{D}\mathbf{PFTxtG}\delta\mathbf{Ex}_V \subset \mathcal{D}\mathcal{R}\mathbf{TxtG}\delta\mathbf{Ex}_V \Leftrightarrow \delta \subseteq \mathbf{Pcp} \text{ and}$$
$$\mathcal{D}'\mathbf{PF}\mathcal{T}\delta\mathbf{TxtGEx}_U \subset \mathcal{D}'\mathcal{T}\delta\mathbf{TxtGEx}_U \Leftrightarrow \delta \subseteq \mathbf{Pcp}.$$

We can, and sometimes do, think of total function learning as a special case of TxtEx-learning thus. Suppose f is any (possibly, but not necessarily total) function mapping non-negative integers into the same. Recall that we identify f with its graph, $\{\langle x, y\rangle \mid f(x) = y\}$, where $\langle x, y\rangle$ is the numeric coding of (x, y) (Section 2). Then $\{\langle x, y\rangle \mid f(x) = y\}$ is a sublanguage of the non-negative integers. Furthermore, programs for f are generally trivially intercompilable with programs or grammars for $\{\langle x, y\rangle \mid f(x) = y\}$. We sometimes refer to languages of the form $\{\langle x, y\rangle \mid f(x) = y\}$ as *single-valued languages*.

Next is our second main result of the present section. It asserts the polynomial time learnability *with restrictions of postdictive completeness, conservativeness and prudence* of a uniformly decidable class of total single-valued languages which are (the graphs of) the linear time computable functions. Importantly, our proof of this theorem employs a Pitt-style delaying trick on an enumeration technique [Gol67, BB75], and our result, then, entails, as advertised in Section 1 above, that *some* delaying tricks are *not* forbidden in the setting of the present section.

Let $\theta^{\mathcal{L}time}$ be an efficiently coded programming system from [RC94, Chapter 6] for **LinF**. $\theta^{\mathcal{L}time}$ is based on multi-tape TM-programs each explicitly clocked to halt in linear time (in the length of its input). Let $V^{\mathcal{L}time}$ be the corresponding effective numbering of all and only those `ce` languages (whose graphs are) $\in$ **LinF**. Note that $V^{\mathcal{L}time}$ does *not* satisfy the condition at the beginning of the present section on Vs for obtaining codes of *finite* languages — since we have only infinite languages in $V^{\mathcal{L}time}$. Instead, for $V^{\mathcal{L}time}$, we have (and use) a linear time algorithm, which on any finite function F, outputs a $V^{\mathcal{L}time}$-index for the zero-extension of F.

Theorem 15

$$\mathbf{LinF} \in \mathcal{T}\mathbf{PrudPF}\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtGEx}_{V^{\mathcal{L}time}}.$$

The remainder of this section presents two results that are used elsewhere. They are put here to present them in more generality. They each hold for *any* V.

The following proposition says that, for any uniformly decidable V, conservative learnability implies total conservative learnability. It is used for proving Theorem 10 in Section 3.

Proposition 16. We have

$$\mathcal{T}\mathbf{ConvTxtGEx}_V = \mathbf{TxtGConvEx}_V.$$

The following theorem holds for all V and states that we can assume total postdictive completeness when learning with total conservativeness.

Theorem 17. We have

$$\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtGEx}_V = \mathcal{T}\mathbf{ConvTxtGEx}_V.$$

5 Learning ce Languages

For the remainder of this section, let V be any effective numbering of some `ce` languages.

For the present section, next (and mentioned in Section 1 above) is our first main result which says that any combination of just the two restrictions of conservativeness and prudence allows for *arbitrary* delaying tricks.

Theorem 18. Let $\delta \in \{\mathcal{R}^2, \mathbf{Conv}\}$, $\mathcal{D} \in \{\mathrm{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathrm{Id}, \mathcal{T}\mathbf{Prud}\}$. Then

$$\mathcal{D}\mathbf{PFTxtG}\delta\mathbf{Ex}_V = \mathcal{D}\mathbf{TxtG}\delta\mathbf{Ex}_V \text{ and}$$
$$\mathcal{D}'\mathbf{PF}\mathcal{T}\delta\mathbf{TxtGEx}_V = \mathcal{D}'\mathcal{T}\delta\mathbf{TxtGEx}_V.$$

Our proof of the just above theorem uses delaying tricks similar to those in the proof of Lemma 9 in Section 3.

Our next main result of the present section says, for the general effective numbering of all `ce` languages, W, combinations of postdictive completeness, conservativeness and prudence forbid *some* delaying tricks iff postdictive completeness is part of the combination.

Theorem 19. Let $\delta \in \{\mathcal{R}^2, \mathbf{Pcp}, \mathbf{Conv}, \mathbf{PcpConv}\}$, $\mathcal{D} \in \{\mathrm{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathrm{Id}, \mathcal{T}\mathbf{Prud}\}$. Then

$$\mathcal{D}\mathbf{PFTxtG}\delta\mathbf{Ex}_W \subset \mathcal{D}\mathcal{R}\mathbf{TxtG}\delta\mathbf{Ex}_W \Leftrightarrow \delta \subseteq \mathbf{Pcp} \text{ and}$$
$$\mathcal{D}'\mathbf{PF}\mathcal{T}\delta\mathbf{TxtGEx}_W \subset \mathcal{D}'\mathcal{T}\delta\mathbf{TxtGEx}_W \Leftrightarrow \delta \subseteq \mathbf{Pcp}.$$

Our proof of the just above theorem makes crucial use of [CK08b, Theorem 5(a)] as well as Theorem 18 above.

Theorem 20 just below says that certain kinds of learners can be assumed without loss of generality to be set-driven. This is interesting on its own, and is also of important technical use for proving Theorem 10 in Section 3.

Theorem 20. Let V be such that (3) holds. We have

$$\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtSdEx}_V = \mathcal{T}(\mathbf{PcpConv})\mathbf{TxtGEx}_V. \tag{7}$$

The following proposition shows that total postdictive complete and total conservative, set-driven learners are automatically totally prudent. This, again, is of important technical use for proving Theorem 10 in Section 3.

Proposition 21. Let δ be a sequence acceptance criterion, let $\mathcal{C} \subseteq \mathcal{P}$. Let $h \in \mathcal{P}$. We have

$$\mathcal{T}\mathbf{Prud}\mathcal{C}\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtSd}\delta\mathbf{Ex}_V(h) = \mathcal{C}\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtSd}\delta\mathbf{Ex}_V(h).$$

Next is our last main result. As noted above in Section 1, this theorem says that, in the *general* setting of the present section, postdictive completeness does *not* forbid *all* delaying tricks.

Theorem 22. We have

$$\mathbf{LinF} \in \mathcal{T}\mathbf{PrudPF}\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtGEx}_W.$$

Proof: The effective numbering $V^{\mathcal{L}time}$ from Theorem 15 can be translated into the W-system in linear (and, hence, in polynomial) time. □

References

[Ang80] Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45, 117–135 (1980)

[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)

[Bur05] Burgin, M.: Grammars with prohibition and human-computer interaction. In: Proceedings of the 2005 Business and Industry Symposium and the 2005 Military, Government, and Aerospace Simulation Symposium, pp. 143–147. Society for Modeling and Simulation (2005)

[CCJ09] Carlucci, L., Case, J., Jain, S.: Learning correction grammars. Journal of Symbolic Logic 74(2), 489–516 (2009)

[CK08a] Case, J., Kötzing, T.: Dynamic modeling in inductive inference. In: Freund, Y., Györfi, L., Turán, G., Zeugmann, T. (eds.) ALT 2008. LNCS (LNAI), vol. 5254, pp. 404–418. Springer, Heidelberg (2008)

[CK08b] Case, J., Kötzing, T.: Dynamically delayed postdictive completeness and consistency in learning. In: Freund, Y., Györfi, L., Turán, G., Zeugmann, T. (eds.) ALT 2008. LNCS (LNAI), vol. 5254, pp. 389–403. Springer, Heidelberg (2008)

[CR09] Case, J., Royer, J.: Program size complexity of correction grammars, Working draft (2009)

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

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

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

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

[Knu73] Knuth, D.: The Art of Computer Programming, Volume III: Sorting and Searching. Addison-Wesley, Reading (1973)

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

[OSW86] Osherson, D., Stob, M., Weinstein, S.: Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists. MIT Press, Cambridge (1986)

[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)

[RC94] Royer, J., Case, J.: Subrecursive Programming Systems: Complexity and Succinctness. Research monograph in Progress in Theoretical Computer Science. Birkhäuser, Boston (1994)

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

[Sch91] Schabes, Y.: Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. In: Proceedings of the 29th annual meeting on Association for Computational Linguistics, pp. 106–113. Association for Computational Linguistics (1991)

[vEB90] van Emde Boas, P.: Machine models and simulations. In: Van Leeuwen, J. (ed.) Handbbook of Theoretical Computer Science. Algorithms and Complexity, vol. A, pp. 3–66. MIT Press/Elsevier (1990)

[Wei82] Weinstein, S.: Private communication at the Workshop on Learnability Theory and Linguistics, University of Western Ontario (1982)

[Wie76] Wiehagen, R.: Limes-erkennung rekursiver funktionen durch spezielle strategien. Elektronische Informationverarbeitung und Kybernetik 12, 93–99 (1976)

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

[Yos09] Yoshinaka, R.: Learning efficiency of very simple grammars from positive data. Theoretical Computer Science 410, 1807–1825 (2009); In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 227–241. Springer, Heidelberg (2007)