Information and Computation 220–221 (2012) 60–73

Contents lists available at SciVerse ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Learning secrets interactively. Dynamic modeling in inductive inference

John Case [a], Timo Kötzing [b,*]

[a] Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA
[b] Department 1: Algorithms and Complexity, Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

ARTICLE INFO

Article history:
Received 23 July 2012
Revised 1 October 2012
Available online 23 October 2012

Keywords:
Inductive inference
Interactive learning

ABSTRACT

Introduced is a new inductive inference paradigm, *dynamic modeling*. Within this learning paradigm, for *example*, function h *learns* function g iff, in the i-th iteration, h and g both produce output, h gets the sequence of all outputs from g in prior iterations as input, g gets all the outputs from h in prior iterations as input, and, from some iteration on, the sequence of h's outputs will be *programs for* the *output sequence* of g.

Dynamic modeling provides an idealization of, for example, a social interaction in which h seeks to discover program models of g's behavior it sees in interacting with g, and h *openly* discloses to g its sequence of candidate program models to see what g says back.

Sample results: every g can be so learned by some h; there are g that can only be learned by an h if g can also learn that h back; there are extremely secretive h which cannot be learned back by any g they learn, but which, nonetheless, succeed in learning infinitely many g; quadratic time learnability is strictly more powerful than linear time learnability. This latter result, as well as others, follows immediately from general correspondence theorems obtained from a *unified* approach to the paradigms within inductive inference.

Many proofs, some sophisticated, employ machine self-reference, a.k.a., recursion theorems.

1. Introduction and motivation

In **Computational limit learning of computable functions** mapping the non-negative integers to the same, an *algorithmic learner* is iteratively given more and more *finite* information generated by a *computable target function*. From this information, the learner, in each iteration (may) synthesize a (suitably interpreted) natural number as output. In the literature, many criteria of successful learning have been proposed. Each such learning criterion defines precisely, possibly among other things, in what way the information will be generated by the target function and how the sequence of iteratively generated outputs and the target function have to relate for the learning to be considered successful. Sometimes each output number will be interpreted as (numerically naming) a program, other times each number will represent a prediction for a yet unseen data point. It is helpful for the present paper to briefly consider some illustrative examples.

Ex-learning [17] exemplifies the category of *identification*. h Ex-learns target g iff, in the i-th iteration, h outputs a conjecture on input $g(0)\ldots g(i-1)$ and there are j, e such that $\forall k \geqslant j$: $h(g(0)\ldots g(k-1)) = e$ and e is a program for g.[1] Such a program e could be carried away and used *offline*.

Learning the next value (Nv-learning) [1,4] exemplifies the category of *extrapolation*. h Nv-learns target g iff, h is total and, in the i-th iteration, h outputs a conjecture on input $g(0)\ldots g(i-1)$ and there is a j such that $\forall k \geqslant j$: $h(g(0)\ldots g(k-1)) = g(k)$.[2] The successful extrapolants $h(g(0)\ldots g(k-1))$, $k \geqslant j$, can be used *online*.

* Corresponding author.
E-mail addresses: case@cis.udel.edu (J. Case), koetzing@mpi-inf.mpg.de (T. Kötzing).

[1] The term 'Ex' stands for *explanatory* [11].

[2] The term 'Nv' stands for *next value* [1].

0890-5401/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ic.2012.10.002

Learning to coordinate (Coord-learning) [22] exemplifies the category of *coordination*. h Coord-learns a target g iff, in the i-th iteration, h and g both produce output, h gets the sequence of all outputs from g in prior iterations as input, g gets all the outputs from h in prior iterations as input, and, from some iteration on, the sequence of h's outputs will be the same as the sequence of g's outputs. The finally successfully coordinated matching outputs can be used *online*.

In these examples we see that, while Coord-learning features a *reactive learnee* g, Ex- and Nv-learning feature a *passive learnee* g.

	Passive learnee	Reactive learnee
Learning programs	Identification	??
Predicting next value	Extrapolation	Coordination

In the just above table, there is a missing, not heretofore studied category entry for *offline* with *reactive learnee*. We refer to this category as *dynamic modeling*, and it is the subject of the present paper. **XBc**-learning exemplifies the category of *dynamic modeling*. h **XBc**-learns a target g iff, in the i-th iteration, h and g both produce output, h gets the sequence of all outputs from g in prior iterations as input, g gets all the outputs from h in prior iterations as input, and, from some iteration on, the sequence of h's outputs will be *programs for* the *output sequence* of g.[3]

In cognitive science, *theory of mind* refers to ones having a model (or models) of another's thoughts, emotions, and perspectives — including those different from ones own. Ideally, one might have a *program* (or programs) generating the behavior of the other, but — the behavior presented by the other would, in reality, be all and only that resulting from crossfeeding between oneself and the other. While one is attempting to synthesize program(s) for the other, *a* technique to employ is to pass on a sequence of remarks such as, "I think you are like ..." (where ... might be a program), and, then, attend to the resultantly elicited sequence of reactions of the other — as one formulates further programs/models of the other. Of course, in reality, one might, in seeking social understanding, carry out variants, including highly filtered variants, of the just above scenario. The unfiltered and very idealized scenario above is, nonetheless, covered by dynamic modeling.

Next we summarize the contents of the remaining sections.

In Section 2 below we present mathematical preliminaries.

Section 3 presents a *unified* approach to limiting learning criteria. This pays off in Section 5 where we can *then* provide general results applying to many criteria at once and, thereby, *quickly* obtain some nice corollaries.[4]

Section 4 involves cooperativeness vs. secretiveness in dynamic modeling. Considered are dynamic modelers which may or may not, in return, be dynamically modeled themselves. Proposition 4.1 implies that *no* computable g can keep models of its behavior totally secret; moreover, for any computable g, there are infinitely many constant functions h that **XBc**-learn g.[5]

Surprisingly, Theorem 4.3 implies that there is a computable g so that, *no* computable h *that* **XBc**-*learns* g can keep models of its behavior a secret from g, i.e., such h gives itself away: g, in turn, **XBc**-learns h. Positively, such a g is, then, *extremely cooperative*: informally, g can figure out the behavior of any computable h that figures out its behavior. Furthermore, such a g can be chosen to be linear time computable! The proof of Theorem 4.3 is particularly elegant.

We say that computable h is *extremely uncooperative* iff there is *no* computable g such that h **XBc**-learns g and g **XBc**-learns h. Theorem 4.7 implies there are extremely uncooperative computable h which, nonetheless, are infinitely successful, i.e., such that h **XBc**-learns infinitely many computable g.

Results in Section 4 feature open disclosure of certain learners' models of another while not disclosing their own models to the other. For comparison and contrast, a *zero-knowledge proof* [5] permits open, convincing disclosure of its existence without disclosing how it works.

Section 5 features two general and powerful correspondence theorems (Theorems 5.10 and 5.12) regarding many of the criteria discussed above and in Section 3. A further result is *dynamic modeling by enumeration* as given in Theorem 5.3. This theorem states that each set of uniformly computable total functions is learnable in the sense of dynamic modeling; this is a known and easy result for Ex-learning, but it requires some thought in the setting of dynamic modeling.

Theorem 5.10 *immediately* yields Corollary 5.11 which implies, for *example*, that quadratic time **XBc**-learnability is strictly more powerful than lintime **XBc**-learnability.[6]

Theorem 5.12 *immediately* yields Corollary 5.13 which provides a number of learning criteria hierarchies and separations. An *example*: the powers of **XBc**-learning and of Coord-learning are incomparable.

Many proofs, some sophisticated, employ machine self-reference techniques, including Kleene Recursion Theorem (**KRT**) [25, page 214, problem 11-4] and Case's Operator Recursion Theorem (**ORT**) [6,7]. The latter achieves infinitary self (and other) reference.

This paper is an extension of the conference paper [10] and was part of the second author's PhD thesis [19].

[3] We use the cross-shaped **X** to denote cross-feeding. Of course crossfeeding of data is common to both the categories of coordination and dynamic modeling. In Section 3 we use the **X** also in talking about the former category. **Bc** stands for *behaviorally correct* [11].

[4] In Section 3 Ex-learning will be called **GEx**-learning, where the **G** is for Gold [17]. Nv-learning will be called $\mathcal{R}$**GM**, where the $\mathcal{R}$ is for (total) computable learner and learnee, and the **M** is for Matching. Coord-learning will be called **XM**-learning.

[5] Actually, Proposition 4.1 is stated for a more restrictive criterion within the dynamic modeling category.

[6] Nothing like this happens for, for example, Ex-learning, since by an extension of Pitt's postponement tricks from [23] (otherwise unrestricted) Ex-learning with lintime learners is just as powerful as Ex-learning.

2. Mathematical preliminaries

Any unexplained complexity-theoretic notions are from [26]. All unexplained general computability-theoretic notions are from [25].

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, $\{0,1,2,\ldots\}$. $*$ is a symbol such that, for all $n \in \mathbb{N}$, $n < *$. For two functions f, g and $n \in \mathbb{N}$, $f =^n g$ means that f and g differ on at most n arguments, $f =^* g$ means that f and g differ only on finitely many arguments.

We do not distinguish between natural numbers and their *dyadic* representations as strings.[7]

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 $\max\{1, \mathrm{size}(w)\}$.[8]

The symbols $\subseteq, \subset, \supseteq, \supset$ respectively denote the subset, proper subset, superset and proper superset relation between sets. $\mathfrak{P}$ and $\mathfrak{R}$, respectively, denote the set of all partial ans total functions $\mathbb{N} \to \mathbb{N}$, respectively.

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

The quantifier $\forall^\infty x$ means "for all but finitely many x", the quantifier $\exists^\infty x$ means "for infinitely many x".

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.

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 $\forall^\infty x\colon\ \psi(x)\downarrow = p$.

Ref. [26, §3] describes an *efficiently* numerically named or coded[9] 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.

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, by $\sigma(1)$ and so on. $\mathrm{len}(\sigma)$ denotes the number of elements in a finite sequence σ, that is, the cardinality of its domain. $\mathrm{last}(\sigma)$ denotes the last element of σ (if any).

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

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

We take $\mathbb{N} = \{0,1\}^*$ for ease of measurement of the size of each natural number. Following [20], we fix an easily computed and inverted coding of all finite sequences of natural numbers into $\mathbb{N}$ so that the size of any sequence, defined as the size of its coding, is sensible for measuring the computational complexity of functions which take sequences as inputs. In particular the size of any sequence σ should be linear in $\mathrm{len}(\sigma)$ and the size of each natural number n linear in $\log(n)$.

Let **LinF**, **PF** and **EXPF** be the set of all linear time, polynomial time and exponential time computable functions, respectively.

We fix a computable coding for the set of all sequences into the natural numbers. *Henceforth, we will many times identify a finite sequence σ with its code number*. 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 this paper we will need a *padding function*, an s-m-n function and a version of **ORT**, introduced below. In particular, we will need non-standard complexity-restricted variants; details on those can be found in [19].

There are a 1-1 function pad and a function unpad_1, both $\in$ **LinF** such that

$$\forall e, n\colon\ \varphi_{\mathrm{pad}(e,n)} = \varphi_e; \tag{1}$$

$$\forall e, n\colon\ \mathrm{unpad}_2\big(\mathrm{pad}(e,n)\big) = n. \tag{2}$$

The function pad is called a *padding function*, intuitively padding a φ-program e with some auxiliary information coded in n without changing the semantics of the program.

[7] 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 [26]. Hence, unlike to binary representation, lead zeros matter.

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

[9] 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.

There is a function $s \in \mathbf{LinF}$ (called *s-m-n function*) such that

$$\forall e, x, y\colon\ \varphi_{s(e,x)}(y) = \varphi_e(x, y).$$

Intuitively, s provides for algorithmic storage of data in programs. We also need 1-1 linear time **ORT**, which states that, for all computable operators Θ, there is 1-1 $e \in \mathbf{LinF}$ such that

$$\forall x, n\colon\ \varphi_{e(n)}(x) = \Theta(e)(x, n).$$

3. Unified approach to limit learning criteria

In this section, in the interest of generality, we give many definitions for limit learning also involving *non*-algorithmic learning. Nonetheless, all the *results* given in the present paper concern only *algorithmic* (including complexity bounded) learning.

3.1. Definitions

Any set $\mathcal{C} \subseteq \mathfrak{P}$ is a *learner admissibility restriction*; intuitively, a learner admissibility restriction defines which functions are admissible as potential learners, e.g., $\mathcal{P}, \mathcal{R}, \mathbf{LinF}, \ldots$.

Every computable operator $\mathfrak{P} \times \mathfrak{R} \to \mathfrak{P}^2$ is called a *sequence generating operator*; intuitively, a sequence generating operator defines how learner and learnee interact to generate two infinite sequences, one for learner-outputs (we call this sequence the *learner-sequence*) and one for learnee-outputs.[10] For example, for Ex-learning (to be renamed in Section 3.2), for a (then, passive) learnee g, its learnee outputs would be $g(0), g(1), \ldots$. Below, in general, even for reactive learnees, we refer to the sequence of learnee-outputs as a *data-sequence*.

For $h \in \mathcal{P}$, $g \in \mathcal{R}$ and β a sequence generating operator, the first component of $\beta(h, g)$ (the learner-sequence of h given g) is denoted by $\beta_1(h, g)$; the second component (the data-sequence of g given h) is denoted by $\beta_2(h, g)$.

Every subset of $\mathfrak{P}^2$ is called a *sequence acceptance criterion*. Intuitively, a sequence acceptance criterion defines which learner-sequences are considered a successful learning of a data-sequence. 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 $\delta \cap \delta'$.

A *learning criterion* (or short *criterion*) is a 3-tuple consisting of a learner admissibility restriction, a sequence generating operator and a sequence acceptance criterion. Let $\mathcal{C}$, β, δ, respectively, be a learner admissibility restriction, a sequence generating operator and a sequence acceptance criterion, respectively. For $h \in \mathfrak{P}$, $g \in \mathfrak{R}$ we say that h $(\mathcal{C}, \beta, \delta)$-learns g iff $h \in \mathcal{C}$ and $\beta(h, g) \in \delta$. For $h \in \mathfrak{P}$ and $\mathcal{S} \subseteq \mathfrak{R}$ we say that h $(\mathcal{C}, \beta, \delta)$-learns $\mathcal{S}$ iff, for all $g \in \mathcal{S}$, h $(\mathcal{C}, \beta, \delta)$-learns g; the set of all computable learnees $(\mathcal{C}, \beta, \delta)$-learned by h is denoted by

$$\mathcal{C}\beta\delta(h) = \bigl\{g \in \mathcal{R} \bigm| h\ (\mathcal{C}, \beta, \delta)\text{-learns } g\bigr\}.$$

The set of $(\mathcal{C}, \beta, \delta)$-learnable sets of computable functions is defined as

$$\mathcal{C}\beta\delta = \bigl\{\mathcal{C}\beta\delta(h) \bigm| h \in \mathcal{C}\bigr\}. \tag{3}$$

We refer to the sets $\mathcal{C}\beta\delta$ as in (3) as *learnability classes*. To refer to a learning criterion, instead of writing the tuple $(\mathcal{C}, \beta, \delta)$, we will ambiguously write $\mathcal{C}\beta\delta$.

For any sequence generating operator β, we can turn a given sequence acceptance criterion δ into a learner admissibility restriction $\mathcal{T}_\beta\delta$ by admitting only those learners that obey δ *on all possible input*:

$$\mathcal{T}_\beta\delta = \bigl\{h \in \mathfrak{P} \bigm| \forall g \in \mathfrak{R}\colon\ \beta(h, g) \in \delta\bigr\}.$$

3.2. Examples

In this section we give many examples illustrating our definitions and give an overview as to how our notation covers criteria from the literature. Past this section, we will not be concerned with *every* example given in this section, but some of them will be employed.

Example 3.1. Two typical learner admissibility restrictions are $\mathcal{P}$ and $\mathcal{R}$. Furthermore, any set of functions computable with a resource restriction (such as the set of all lintime computable functions) may be used as a learner admissibility restriction. For each sequence generating operator β and each $\mathfrak{F} \subseteq \mathfrak{R}$, the set $\mathbf{Rel}_{\beta,\mathfrak{F}}$ of all functions reliable on $\mathfrak{F}$ [21,4] (defined below) is also a learner admissibility restriction.

$$\mathbf{Rel}_{\beta,\mathfrak{F}} := \bigl\{h \in \mathcal{P} \bigm| \forall g \in \mathfrak{F}\ \forall p, q \in \mathcal{P}\colon\ \bigl(\beta(h, g) = (p, q) \wedge p \text{ converges to some} p'\bigr) \Rightarrow \varphi_{p'} = q\bigr\}.$$

[10] Essentially, *these* computable operators are the recursive operators of [25] but with two arguments and two outputs and restricted to the indicated domain.

When denoting criteria with $\mathcal{P}$ as the learner admissibility restriction, we will omit $\mathcal{P}$.

Example 3.2. We define the following example sequence generating operators. All learners give an initial conjecture, say, of 0, based on no data.

- Goldstyle [17]: $\mathbf{G}: \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}$, $(h, g) \mapsto (\lambda i \centerdot h(g[i]), g)$.
- Iterative [27]: $\mathbf{It}: \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}$, $(h, g) \mapsto (p, g)$ such that $p(0) = 0$, $\forall n$: $p(n+1) = h(q(n), p(n))$.
- Transductive: $\mathbf{Td}: \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}$, $(h, g) \mapsto (p, g)$ such that $p(0) = 0$, $\forall n$: $p(n+1) = h(q(n))$.
- Crossfeeding [22] $\mathbf{X}: \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{P}$, $(h, g) \mapsto (p, q)$ such that $\forall n$ $p(n) = h(q[n]) \wedge q(n) = g(p[n])$.
- Learnee iterative: $\mathbf{Li}: \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{P}$, $(h, g) \mapsto (p, q)$ such that $\forall n p(n) = h(q[n]) \wedge q(0) = 0 \wedge \forall n$: $q(n+1) = g(p(n), q(n))$.

"**G**" is a reference to Gold [17]. Intuitively, **G** takes a learner h and a learnee g, and feeds longer and longer initial segments of g into h, considering the successive outputs as coding an infinite sequence of hypotheses. The second output is just g, meaning that the target concept to be learned is all of g. In this setting, the learner gets a lot of information about the learnee, while the learnee does not react at all to the learning process. For **It** and **Td** defined above, a learner for the latter has less information at its disposal than for the former.

Regarding **X**, learner and learnee have symmetrical information in each iteration. **Li** lessens the information that the learnee has in a similar way that iterative learning lessens the information of the learner.

The first three bullets given in Example 3.2 involve passive learnees, while the last two examples involve reactive learnees.

We note the following three important properties relating **G** and **X**, which are of relevance to this paper. Let $f, g, h, p, q \in \mathcal{P}$.

$$\mathbf{X}(h, g) = (p, q) \quad \Rightarrow \quad \mathbf{G}(h, q) = (p, q); \tag{4}$$

$$\mathbf{X}(h, g) = (p, q) \quad \Rightarrow \quad \mathbf{G}(g, p) = (q, p); \tag{5}$$

$$\mathbf{X}\big(h, \lambda\sigma \centerdot f\big(\mathrm{len}(\sigma)\big)\big) = \mathbf{G}(h, f). \tag{6}$$

The last item says, intuitively, that G-learning a function f is the same as crossfeeding with a function that ignores all the conjectures made and only considers the current iteration number n (which equals the number of conjectures made) and returns $f(n)$; we will use this for an observation at the end of this section.

Example 3.3. We define the following sequence acceptance criteria.

- Explanatory: $\mathbf{Ex} = \{(p, q) \in \mathfrak{P}^2 \mid p$ converges to some p' and $\varphi_{p'} = q\}$.
- Explanatory with up to $a \in \mathbb{N} \cup \{*\}$ errors [11,4]:

$$\mathbf{Ex}^a = \big\{(p, q) \in \mathfrak{P}^2 \bigm| p \text{ converges to some } p' \text{ and } \varphi_{p'} =^a q\big\}.$$

- Behaviorally correct [11,3]: $\mathbf{Bc} = \{(p, q) \in \mathfrak{P}^2 \mid \forall^\infty n\ \varphi_{p(n)} = q\}$.
- Behaviorally correct with up to $a \in \mathbb{N} \cup \{*\}$ errors [11]:

$$\mathbf{Bc}^a = \big\{(p, q) \in \mathfrak{P}^2 \bigm| \forall^\infty n\ \varphi_{p(n)} =^a q\big\}.$$

- Matching [1,4,22]: $\mathbf{M} = \{(p, q) \in \mathfrak{P} \times \mathfrak{R} \mid p =^* q\}$.

All the above criteria include global restrictions on the path to successful learning. The following defines several criteria only involving local restrictions.

- Postdictively complete [2,4,27]: $\mathbf{Pcp} = \{(p, q) \in \mathfrak{R}^2 \mid \forall n \forall i < n\colon\ \varphi_{p(n)}(i) = q(i)\}$.
- Hypotheses are programs for total functions [11]: $\mathbf{T} = \{(p, q) \in \mathfrak{R}^2 \mid \forall n\colon\ \varphi_{p(n)} \in \mathcal{R}\}$.
- Always giving hypotheses: $\mathfrak{R}^2$.

Note that, for any sequence acceptance criterion δ, we use δ^{-1} to denote the inverse relation, i.e., $(p, q) \in \delta$ iff $(q, p) \in \delta^{-1}$.

The idea of dividing a learning criterion into different components is not entirely new. For example, Freivalds et al. [16] defined *admissible sequences for a given function*, which basically defines a binary predicate on a pair of infinite sequences, in effect similar to sequence acceptance criteria.

We can now express several learning criteria as defined in the prior literature (left-hand-side below) with our notation system (right-hand-side below). Recall that the default learner admissibility restriction is $\mathcal{P}$; hence, all learning criteria displayed just below are for *algorithmic* learners.

$$\begin{aligned}
\text{Ex} &\leftrightarrow \mathbf{GEx}\\
\text{Bc} &\leftrightarrow \mathbf{GBc}\\
\text{Nv} &\leftrightarrow \mathcal{R}\mathbf{GM}\\
\text{Nv}' &\leftrightarrow \mathbf{G}\mathcal{R}^2\mathbf{M}\\
\text{Nv}'' \text{ [24]} &\leftrightarrow \mathbf{GM}\\
\text{Cons} &\leftrightarrow \mathbf{GPcpEx}\\
\text{R-Cons} &\leftrightarrow \mathcal{R}\mathbf{GPcpEx}\\
\text{T-Cons} &\leftrightarrow \mathcal{T}_{\mathbf{G}}\mathbf{PcpGEx}\\
\text{reliable on}\mathcal{R} &\leftrightarrow \mathbf{Rel}_{\mathcal{R}}\mathbf{GEx}\\
\text{It} &\leftrightarrow \mathbf{ItEx}\\
\text{learnable by a player [22]} &\leftrightarrow \mathbf{XM}\\
\text{learnable by a total player [22]} &\leftrightarrow \mathcal{R}\mathbf{XM}
\end{aligned}$$

A sequence acceptance criterion δ is said to be *degenerate* iff $\exists(p,q) \in \delta$: $p =^* \lambda x.\uparrow$. All sequence acceptance criteria given above are non-degenerate, and the authors don't know of any degenerate sequence acceptance criteria implicit in the prior literature. We conjecture that any degenerate sequence acceptance criterion would be useless to model learning. Hence, the present paper will solely focus on non-degenerate such criteria.

The following remark is easy to see.

Remark 3.4. Let δ be non-degenerate and $\mathcal{C} \subseteq \mathcal{P}$. Then the learnability classes $\mathcal{C}\mathbf{X}\delta$ and $\mathcal{C}\mathbf{X}\mathcal{R}^2\delta$ are *equal*.

Similarities between extrapolation (like **GM**) and coordination (like **XM**) have been pointed out in [9]. In particular, *blind* learnees are defined as functions where each output only depends on the *length* of it's input, and with each function $g \in \mathcal{R}$, a blind learnee $g' = \lambda\sigma.g(\text{len}(\sigma))$ is associated. The mapping $\Theta = \lambda g.g'$ is then a natural embedding of learnees in the **G**-sense to learnees in the **X**-sense; more formally, for all sequence acceptance criteria δ and $\mathcal{S} \subseteq \mathcal{R}$,

$$\mathcal{S} \in \mathbf{G}\delta \quad \Leftrightarrow \quad \Theta(\mathcal{S}) \in \mathbf{X}\delta. \tag{7}$$

The special case of (7) with $\delta = \mathbf{M}$ is used in [9].

4. Cooperation and secretiveness

The main emphasis of the present paper, as seen in Section 1, features **XBc**-learning, but, based on the thinking of Section 3, one might wonder why we didn't talk about **XEx**-learning. We'll talk about it now. Suppose h **XEx**-learns g. The learner-sequence of h interacting with g, is, then, a total, almost everywhere constant function. Suitable g, then, easily **XEx**-learn h.[11]

The proposition just below implies that *any* computable g gets **XBc**-learned by some computable h. Hence, any g can have its secrets learned by some learner. The interesting thing, then, is whether, when h **XBc**-learns g, h can keep models of itself secret from g. This is considered in Theorem 4.3 further below.

Proposition 4.1. *Let $g \in \mathcal{R}$. Then there are infinitely many (total) constant functions h* **XBc**- *(in fact,* **XEx**-*) learning g.*

Proof. Let $n \in \mathbb{N}$. There is, by **KRT**, e_n such that, with

$$p = \lambda x.\text{pad}(e_n, n), \tag{8}$$

$$\forall x\colon\ \varphi_{e_n}(x) = g\big(p[x]\big). \tag{9}$$

Let $h_n \in \mathcal{R}$ be such that

$$\forall\sigma\colon\ h_n(\sigma) = \text{pad}(e_n, n). \tag{10}$$

There is $q \in \mathcal{R}$ such that $\mathbf{X}(h_n, g) = (p, q)$. Then we have, for all t and x,

$$\varphi_{p(t)}(x) \underset{(8)}{=} \varphi_{\text{pad}(e_n,n)}(x) \underset{(1)}{=} \varphi_{e_n}(x) \underset{(9)}{=} g\big(p[x]\big) \underset{\text{choice of } q}{=} q(x). \tag{11}$$

Hence, h_n **XBc**-learns g. Trivially, using (1), we have for all $l \neq m$, $h_l \neq h_m$. This shows that there are infinitely many different (constant) functions **XEx**-learning g. □

[11] We have, more generally, that

$$\exists g \in \mathcal{R}\ \forall h \in \mathcal{R}\colon\ h\ \mathbf{XEx}\text{-learns } g \quad \Rightarrow \quad g\ \mathbf{XEx}\text{-learns } h.$$

The Stanford psychologist Herb Clark specializes in treating human language communication as serving a/the primary purpose of coordinating activities [14,12,13]. He also knows about the influential inductive inference work first introduced by Gold in [17]. When the first author told him about his plan just above to extend coordination as in [22], etc., he responded:

> I worry a lot about the Gold type models because they make such simplistic, by which I mean empirically unrealistic, assumptions about feedback, interaction, evidence of understanding, etc. So if you can move the model into one that takes account of coordination with others in the proper way, terrific!

In this section, we try to get a little bit closer to what Herb Clark suggested analyzing.

Definition 4.2. We define the following sequence acceptance criteria.

- Cooperative Bc: $\mathbf{Bcc} = \{(p,q) \in \mathcal{R}^2 \mid \forall^\infty n\ \varphi_{p(n)} = q \wedge \forall^\infty n\ \varphi_{q(n)} = p\}$ $(= \mathbf{BcBc}^{-1})$.
- Secretive Bc: $\mathbf{Bcs} = \{(p,q) \in \mathcal{R}^2 \mid \forall^\infty n\ \varphi_{p(n)} = q \wedge \neg\forall^\infty n\ \varphi_{q(n)} = p\}$ $(= \mathbf{Bc}\overline{\mathbf{Bc}^{-1}})$.

Clearly, for all $h, g \in \mathcal{R}$, h **XBcc**-learns g iff, h **XBc**-learns g and g **XBc**-learns h; similarly, h **XBcs**-learns g iff, h **XBc**-learns g and g does *not* **XBc**-learn h.

It is easy to see that there are computable functions which are not **XBcc**-learnable, for example $\lambda\sigma\,.\,\mathrm{len}(\sigma)$.[12] At first glance, it seems likely that all computable functions can be **XBcs**-learned, as, by Proposition 4.1 above, for any given function g, there are infinitely many functions h **XBc**-learning g. We were, then, surprised that not all computable functions can be **XBcs**-learned, as seen below in Theorem 4.3. Intuitively, this theorem means that there is a $g \in \mathcal{R}$ such that, for all $h \in \mathcal{P}$, if h **XBc**-learns g, then h has to give away enough information about itself so that g will be able to **XBc**-learn h. Even more surprisingly, such a g can be chosen to be linear time computable! On the other hand, Theorem 4.3 also has a positive interpretation: it *is* possible to find a function g that will **XBc**-learn every function h that **XBc**-learns g – in other words, there are *extremely cooperative functions* that will cooperate with *any* function **XBc**-learning them. We denote the set of extremely cooperative functions with $EC := \{h \in \mathcal{R} \mid \forall g \in \mathcal{R}\colon\ g$ **XBc**-learns $h \Rightarrow h$ **XBc**-learns $g\}$.[13]

Theorem 4.3 *(Secretiveness Fails).*

$$\exists g \in \mathbf{LinF}\colon \{g\} \notin \mathbf{XBcs}.$$

Proof. By 1-1, linear time **ORT** there is 1-1 $g \in \mathbf{LinF}$ such that

$$\forall \tau, x\colon \varphi_{g(\tau)}(x) = \mathrm{last}\big(g^{-1}\big(\varphi_{\mathrm{last}(\tau)}(x+1)\big)\big).^{14} \tag{12}$$

Let $h \in \mathcal{P}$ be such that $g \in \mathbf{XBc}(h)$. Let $p, q \in \mathcal{R}$ be such that $\mathbf{X}(h,g) = (p,q)$. Since $(p,q) \in \mathbf{Bc}$, there is n_0 such that

$$\forall n \geqslant n_0\colon \varphi_{p(n)} = q. \tag{13}$$

Claim 1. $\forall^\infty n\colon \varphi_{q(n)} = p.$

Proof. We have

$$\forall n \geqslant n_0 + 1, x\colon \varphi_{p(n-1)}(x+1) \underset{(13)}{=} q(x+1) \underset{\text{choice of } q}{=} g\big(p[x+1]\big). \tag{14}$$

Hence, for all $n \geqslant n_0 + 1$ and all x,

$$\begin{aligned}\varphi_{q(n)}(x) &\underset{\text{choice of } q}{=} \varphi_{g(p[n])}(x) \underset{(12)}{=} \mathrm{last}\big(g^{-1}\big(\varphi_{p(n-1)}(x+1)\big)\big)\\ &\underset{(14)}{=} \mathrm{last}\big(g^{-1}\big(g\big(p[x+1]\big)\big)\big) = \mathrm{last}\big(p[x+1]\big) = p(x). \qquad \square\end{aligned} \tag{15}$$

Hence, by the claim, $g \in \mathbf{XBcc}(h)$; therefore, $\{g\} \notin \mathbf{XBcs}$. $\square$

[12] For all $g \in \mathcal{R}$, and $p, q \in \mathcal{R}$ such that $\mathbf{X}(\lambda\sigma\,.\,\mathrm{len}(\sigma), g) = (p,q)$, we have that p is the identity on $\mathbb{N}$; hence, $\lambda\sigma\,.\,\mathrm{len}(\sigma)$ does not **XBc**-learn g.

[13] It is easy to see that $EC = \{h \in \mathcal{R} \mid \mathbf{XBcs}^{-1}(h) = \emptyset\}$.

[14] Note that $\varphi_{g(\tau)}(x)$ might be undefined for various reasons, for example last is not total. Furthermore, note that accessing g^{-1} is a valid use of **ORT**, in the sense that $g^{-1}(y)$ is the first x found such that $g(x) = y$; x will be unique, as g is 1-1.

Ref. [22] examined uncooperativeness of coordinators. In particular, two sets of total computable functions are constructed such that any learner learning *all* of the functions from one of the sets cannot coordinate with *any* function from the other set. Furthermore, Ref. [9] extended this result showing that for all $k \geqslant 2$, one can find k such sets of uncooperative learners. Below, we give an analog of this result for cooperation in the **XBcc**-sense, where we give an infinite family of uncooperative sets, so that any learner that can **XBc**-learn *any* of the functions in one of the sets cannot **XBc**-learn *any* of the functions of any other set.

Theorem 4.4 *(Incompatible Mutual Cooperation Camps). There is a* 1-1 $e \in \mathcal{R}$ *such that for all* m, n, $\varphi_{e(m,n)}$ *total and, defining* $\mathcal{S}_n = \{\varphi_{e(m,n)} \in \mathcal{R} \mid m \in \mathbb{N}\}$, *for each* n *all members of* $\mathcal{S}_n$ **XBcc***-learn each other, while each function* $h \in \mathcal{P}$ **XBcc***-learns functions from at most one of the sets in* $\{\mathcal{S}_n \mid n \in \mathbb{N}\}$.

Proof. Let $f \in \mathcal{P}$ be such that $\forall e, e', x$: $f(e, e', x) = \mathbf{X}_2(\varphi_e, \varphi_{e'})(x)$. Obviously, for all e, e' such that $\varphi_e, \varphi_{e'} \in \mathcal{R}$ and for all x, $f(e, e', x)\downarrow$. By the Generalized Delayed Recursion Theorem [6, Theorem 23] we find a 1-1 $e \in \mathcal{R}$ such that, for all m, n, $\varphi_{e(m,n)}$ total and

$$\forall \tau, x\colon\ \varphi_{\varphi_{e(m,n)}(\tau)}(x) = \begin{cases} n, & \text{if } \tau = \emptyset; \\ \varphi_{e(m,n)}(x), & \text{else if } \mathrm{len}(\tau) = 1 \text{ and } \varphi_{\tau(0)}(0)\downarrow = n; \\ f(e(m,n), \tau(1), x), & \text{else if } \varphi_{\tau(0)}(0)\downarrow = n; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{16}$$

Intuitively, each $\varphi_{e(m,n)}$ declares it's "group" on input $\emptyset$ to be n. If the input suggests that $\varphi_{e(m,n)}$ is being fed another function from its "group" then it delivers helpful output (a program number for itself) as the next output, followed by trying to model it's co-learner, which will be successful, if the co-learner has also output a program number for itself as second output. If the input does not suggest that the co-learner is from the same team, $\varphi_{e(m,n)}$ will act uncooperatively.

For each $n \in \mathbb{N}$, let $\mathcal{S}_n = \{\varphi_{e(m,n)} \in \mathcal{R} \mid m \in \mathbb{N}\}$.

Let $h \in \mathcal{P}$. Our first claim implies that h can dynamically model functions from at most one of the above ce sets of functions.

Claim 1. *Let* m, n *be such that* $n \neq \varphi_{h(\emptyset)}(0)$. *Then* h *does not* **XBcc***-learn* $g := \varphi_{e(m,n)}$.

Proof. Suppose h does **XBc**-learn g. Then there are $p, q \in \mathcal{R}$ such that $\mathbf{X}(h, g) = (p, q)$; hence $\mathbf{X}(g, h) = (q, p)$. We show that g does not **XBc**-learn h. For all $\tau \subset p$ with $\mathrm{len}(\tau) \geqslant 1$ we have $\varphi_{\tau(0)}(0) = \varphi_{p(0)} \underset{\text{def } \mathbf{X}}{=} \varphi_{h(\emptyset)}(0) \neq n$; thus, by (16) $\forall x$: $\varphi_{g(\tau)}(x) = \varphi_{\varphi_{e(m,n)}(\tau)}(x)\uparrow$. Hence, $\forall^\infty t$: $\varphi_{q(t)} \underset{\text{def } \mathbf{X}}{=} \varphi_{g(p[t])} \neq p$. □

Claim 2 just below implies that, for each n, all functions in $\mathcal{S}_n$ **XBcc**-learn each other.

Claim 2. *Let* $n \in \mathbb{N}$ *and* $h, g \in \mathcal{S}_n$. *Then* $g \in$ **XBc**(h).

Proof. Let p, q be such that $\mathbf{X}(h, g) = (p, q)$. We have that

$$\varphi_{p(0)}(0) \underset{\text{def } \mathbf{X}}{=} \varphi_{h(\emptyset)}(0) \underset{(16)}{=} n; \tag{17}$$

$$\varphi_{q(0)}(0) \underset{\text{def } \mathbf{X}}{=} \varphi_{g(\emptyset)}(0) \underset{(16)}{=} n. \tag{18}$$

Hence,

$$\varphi_{q(1)} \underset{\text{def } \mathbf{X}}{=} \varphi_{g(p[1])} \underset{(16)\ \&\ (17)}{=} g. \tag{19}$$

Let m be such that $h = \varphi_{e(m,n)}$. Now we have, for all $t > 1$ and all x,

$$\begin{aligned}\varphi_{p(t)}(x) &\underset{\text{def } \mathbf{X}}{=} \varphi_{g(q[t])}(x) \underset{(16)\ \&\ (18)}{=} f\big(e(m,n), q(1), x\big) \\ &\underset{\text{def } f}{=} \mathbf{X}_2(\varphi_{e(m,n)}, \varphi_{q(1)})(x) \underset{(19)}{=} \mathbf{X}_2(h, g)(x) \underset{\text{def } \mathbf{X}_2}{=} q(x). \quad \square\end{aligned} \tag{20}$$

As a contrast to the extremely cooperative functions as defined above, we say that $h \in \mathcal{R}$ is *extremely uncooperative* iff **XBcc**$(h) = \emptyset$ (i.e., h cooperates with no function). The set of all extremely uncooperative functions is denoted by *EU*. Trivially, $EU \neq \emptyset$, as *EU* contains each function h that doesn't **XBc**-learn any function. Interestingly, *EU* is rather big in the sense that the closure under **LinF** composition of *EU is equal to* $\mathcal{R}$.[15] However, many of the functions $h \in EU$ will not **XBc**-model

[15] Let $f \in \mathcal{R}$, let p be such that $\varphi_p = \lambda x \centerdot \uparrow$. Then $f' = \lambda x \centerdot \mathrm{pad}(p, f(x)) \in EU$. Define $a = \lambda x \centerdot \mathrm{unpad}_2(x) \in$ **LinF**. Then $a \circ f' = f$.

anything. We define a (computable) operator below, turning a given learner h into an extremely uncooperative learner h', which intuitively doesn't lose too much of the learning power of h.

Furthermore, Theorem 4.7 below states the existence of $h \in EU$ with $\mathbf{XBc}(h)$ infinite.

Definition 4.5. For all $h \in \mathcal{R}$ there is, by linear time **ORT**, $h' \in \mathbf{LinF}$ such that

$$\forall \sigma, x\colon\ \varphi_{h'(\sigma)}(x) = \begin{cases} \varphi_{h(\sigma)}(x), & \text{if } \sigma = \emptyset \vee \exists s \geqslant \text{len}(\sigma), t\colon\ \varphi_{\varphi_{h(\sigma)}(s)}(t)\downarrow \neq h'(\varphi_{h(\sigma)}[t])\downarrow; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{21}$$

Intuitively, h' makes conjectures mostly behaviorally equivalent to those of h, but modified so that the conjectures are definitely wrong as soon as the input seems to learn the outputs of h'.

Define $\Psi = \lambda h \in \mathcal{R}.h'$. N.B. For each $h \in \mathcal{R}$, $\Psi(h) \in \mathbf{LinF}$.

Lemma 4.6. *Let $h \in \mathcal{R}$. Then $\mathbf{XBcc}(\Psi(h)) = \emptyset$.*

Proof. Let $h' = \Psi(h)$. Suppose, by way of contradiction, there is $g \in \mathbf{XBcc}(h')$. Let $p, q \in \mathcal{R}$ be such that $\mathbf{X}(h', g) = (p, q)$. Now we have

$$\forall^\infty n\colon\ \varphi_{p(n)} = q; \tag{22}$$

$$\forall^\infty n\colon\ \varphi_{q(n)} = p. \tag{23}$$

Thus,

$$\forall^\infty n\colon\ \varphi_{h'(q[n])} \underset{\text{def } \mathbf{X}}{=} \varphi_{p(n)} \underset{(22)}{=} q \in \mathcal{R}. \tag{24}$$

For each $n \in \mathbb{N}$, $\varphi_{h'(q[n])} \in \mathcal{R}$ implies that in (21) for $\sigma = q[n]$ we have that the first case holds; that is,

$$\forall^\infty n\ \exists s > n, t\colon\ \varphi_{\varphi_{h(q[n])}(s)}(t)\downarrow \neq h'\big(\varphi_{h(q[n])}[t]\big)\downarrow. \tag{25}$$

Thus,

$$\forall^\infty n\colon\ \varphi_{h(q[n])} \underset{(21)\ \&\ (25)}{=} \varphi_{h'(q[n])} \underset{(24)}{=} q. \tag{26}$$

We have $\forall^\infty n\ \forall s > n, t$:

$$\varphi_{\varphi_{h(q[n])}(s)}(t) \underset{(26)}{=} \varphi_{q(s)}(t) \underset{(23)}{=} p(t) \underset{\text{def } \mathbf{X}}{=} h'\big(q[t]\big) \underset{(24)\ \&\ (26)}{=} h'\big(\varphi_{h(q[n])}[t]\big), \tag{27}$$

a contradiction to (25). □

Theorem 4.7 *(Extremely Uncooperative Infinitely Successful Learners). There are functions $h \in \mathcal{R}$ such that $\mathbf{XBcs}(\Psi(h))$ is infinite, but $\mathbf{XBcc}(\Psi(h)) = \emptyset$ (i.e., no function **XBc**-learned by $\Psi(h)$ can **XBc**-learn $\Psi(h)$).*

Proof. By s-m-n there is $h \in \mathcal{R}$ such that

$$\forall \sigma, x\colon\ \varphi_{h(\sigma)}(x) = \begin{cases} 0, & \text{if } \sigma = \emptyset; \\ \sigma(0), & \text{otherwise.} \end{cases} \tag{28}$$

We fix an infinite set S such that for all $e \in S$, $\varphi_e(0) = \Psi(h)(\emptyset) + 1$. We show that $\Psi(h)$ **XBcs**-learns $\{\lambda x.e \mid e \in S\}$; to this end, let $e \in S$ and let $g = \lambda x.e$. Let $(p, q) = \mathbf{X}(h, g)$. We have that g does not **XBc**-learn $\Psi(h)$, as, for all n, $q(n) = e$ and

$$\varphi_e(0) \neq \Psi(h)(\emptyset) = p(0).$$

We have, for all $n > 0$, $\varphi_{h(q[n])} = \lambda x.e$; thus, we get that h **XBc**-learns g; it remains to show that $\Psi(h)$ **XBc**-learns g. This follows from the following inequality, which holds for all $n > 0$.

$$\varphi_{\varphi_{h(q[n])}(0)}(0) = \varphi_e(0) \neq \Psi(h)(\emptyset) = \Psi(h)\big(\varphi_{h(q[n])}[0]\big). \qquad \square$$

The next theorem intuitively implies that requiring a learner to be extremely uncooperative will decrease its learning power with respect to plain uncooperative learning.

Corollary 4.8. *EU***XBcs** $\subset$ *$\mathcal{R}$***XBcs**.[16]

[16] Less surprisingly, one can also show EC**XBcc** $\subset \mathcal{R}$**XBcc**.

Proof. [17] N.B. This proof uses notions and concepts defined and explained in Section 5. **Bcs** is *non-trivial* (see Definition 5.4 in Section 5). By Proposition 5.8, it suffices to show that there is a **XBcs**-maximal learner in $\mathcal{R} \setminus EU$. Let $h \in \mathcal{R}$ be such that for all σ with $\text{len}(\sigma) > 0$ we have $h(\sigma) = \text{unpad}_2(\text{last}(\sigma))$. Let $r \in \mathcal{R}$ be such that $\forall e, p$: $\varphi_{r(e,p)} = \mathbf{X}_1(\varphi_e, \varphi_p)$.

Claim 1. *$h \notin EU$.*

Proof. Let $p \in \mathbb{N}$ be such that $\varphi_p = h$. By **KRT** there is $e \in \mathbb{N}$ such that

$$\forall x\colon\ \varphi_e(x) = \text{pad}\big(r(p,e), r(e,p)\big). \tag{29}$$

We now have, for all $n > 0$, $\mathbf{X}_1(h, \varphi_e)(n) = r(e,p)$, which is a program for $\mathbf{X}_1(\varphi_e, \varphi_p) = \mathbf{X}_2(h, \varphi_e)$ as desired; furthermore, for all n, $\mathbf{X}_2(h, \varphi_e)(n) = \text{pad}(r(p,e), r(e,p))$, which is a program for $\mathbf{X}_1(\varphi_p, \varphi_e) = \mathbf{X}_1(h, \varphi_e)$. □

By Theorem 5.9, it suffices to show $[\mathbf{XBcs}(h)]_h = \mathbb{S}\text{eq}$ to apply Proposition 5.8.

Claim 2. $[\mathbf{XBcs}(h)]_h = \mathbb{S}\text{eq}$.

Proof. Let $\sigma \in \mathbb{S}\text{eq}$, let p be such that $\varphi_p = \lambda x \centerdot \uparrow$. We now construct a function which will first give outputs according to σ and then make sure that it does not **XBc**-learn h while being **XBc**-learned by h. By **KRT** there is $e \in \mathbb{N}$ such that

$$\forall \tau\colon\ \varphi_e(\tau) = \begin{cases} \sigma(\text{len}(\tau)), & \text{if } \text{len}(\tau) < \text{len}(\sigma); \\ \text{pad}(p, r(e,p)), & \text{otherwise.} \end{cases} \tag{30}$$

We have that φ_e does not **XBc**-learn h, as all outputs after $\text{len}(\sigma)$ are for the everywhere undefined function. On the other hand, h does **XBc**-learn φ_e as follows. Let $(p,q) = \mathbf{X}(h, \varphi_e)$. We have, for all $n > \text{len}(\sigma)$,

$$h\big(q[n]\big) = \text{unpad}_2\big(\text{last}\big(q[n]\big)\big) = \text{unpad}_2\big(\text{pad}\big(p, r(e,p)\big)\big) = r(e,p).$$

By the definition of r, we have that $r(e,p)$ is a program for q, as desired. □

5. General crossfeeding

Most lemmas, propositions and theorems of this section carry over with slightly modified hypotheses to the case of **Li** (from Section 3.1) in the place of **X**.

Just below is a proposition with corollary regarding which sequence acceptance criteria allow dynamical modeling all of $\mathcal{R}$.

Proposition 5.1. *Let δ be a sequence acceptance criterion, let $\mathcal{C} \subseteq \mathcal{P}$. Then we have*

$$\mathcal{R} \in \mathcal{C}\mathbf{X}\delta \quad \Leftrightarrow \quad \mathcal{R} \in \mathcal{C}\mathbf{G}\delta.$$

Proof. "⇒": Let h witness $\mathcal{R} \in \mathcal{C}\mathbf{X}\delta$. Let $g \in \mathcal{R}$. Let $p \in \mathcal{P}$ be such that $\mathbf{G}(h,g) = (p,g)$. Now we have $\mathbf{G}(h,g) \underset{(6)}{=} \mathbf{X}(h, \lambda\sigma \centerdot g(\text{len}(\sigma))) \in \delta$.

"⇐": Let h witness $\mathcal{R} \in \mathcal{C}\mathbf{G}\delta$. Let $g \in \mathcal{R}$. Let $p, q \in \mathcal{P}$ be such that $\mathbf{X}(h,g) = (p,q)$. Now we have $\mathbf{X}(h,g) \underset{(4)}{=} \mathbf{G}(h,q) \in \delta$. □

We get the following corollary by a theorem of Harrington, cited in [11].

Corollary 5.2.

$$\mathcal{R} \in \mathbf{XBc}^*.$$

Gold [17] introduced learning by enumeration. Analogous to, but harder to prove than the case for **G**-style learning, we have, by the next theorem that any computably enumerable set of (total) computable functions is **XEx**-modelable.

Theorem 5.3 *(Dynamic Modeling by Enumeration). Let $r \in \mathcal{R}$ be an enumeration of program numbers of (total) computable functions. We have*

$$\{\varphi_{r(n)} \mid n \in \mathbb{N}\} \in \mathbf{XEx}.^{18}$$

[17] One could try to use the Learner Correspondence Theorem (Theorem 5.10) to prove the above Corollary 4.8. However, *EU* is not closed under **LinF** composition, and the closure of *EU* under **LinF** composition is equal to $\mathcal{R}$; see the paragraph just before Definition 4.5.

[18] In fact, **Ex** could be replaced by any δ from a wide set of sequence acceptance criteria.

Proof. For a number-theoretic (partial) predicate P and $n \in \mathbb{N}$, let

$$\mu x \leqslant n\,\textbf{.}\,P(x) = \begin{cases} x, & \text{if } \forall y < x\colon\ P(y)\downarrow \neq \text{true and } P(x)\downarrow = \text{true;} \\ n+1, & \text{if } \forall y < n+1\colon\ P(y)\downarrow \neq \text{true;} \\ \uparrow, & \text{otherwise.} \end{cases} \tag{31}$$

By **ORT**, there are function $h, u \in \mathcal{P}$ and $s \in \mathcal{R}$ such that

$$\forall \sigma\colon\ u(\sigma) = \mu k \leqslant \text{len}(\sigma)\,\textbf{.}\,\sigma \subseteq \mathbf{X}_2(h, \varphi_{r(k)}); \tag{32}$$

$$\forall m\colon\ \varphi_{s(m)} = \mathbf{X}_2(h, \varphi_{r(m)}); \tag{33}$$

$$\forall \sigma\colon\ h(\sigma) = s\big(u(\sigma)\big). \tag{34}$$

Intuitively, u finds the least index k such that the observed data is consistent with the "possible world" $r(k)$; and s computes a corresponding conjecture.

It is easy to see by induction that $h, u \in \mathcal{R}$. Let $n \in \mathbb{N}$ and $g = \varphi_{r(n)}$. We show that h **XEx**-learns g. Let $p, q \in \mathcal{R}$ be such that $\mathbf{X}(h, g) = (p, q)$. Obviously, for all j, $u(q[j])\downarrow \leqslant n$. Also, u is monotone in the sense that $\forall i, j\colon\ i \leqslant j \Rightarrow u(q[i]) \leqslant u(q[j])$. Thus, there is m such that

$$\forall^\infty j\colon\ u\big(q[j]\big) = m. \tag{35}$$

Hence, p converges. By (32) and (35),

$$\forall^\infty j\colon\ q[j] \subseteq \mathbf{X}_2(h, \varphi_{r(m)}); \quad \text{thus,} \tag{36}$$

$$q = \mathbf{X}_2(h, \varphi_{r(m)}). \tag{37}$$

The following completes the proof.

$$\forall^\infty j\colon\ \varphi_{p(j)} \underset{\text{def } p}{=} \varphi_{h(q[j])} \underset{(34)\ \&\ (35)}{=} \varphi_{s(m)} \underset{(33)}{=} \mathbf{X}_2(h, \varphi_{r(m)}) \underset{(37)}{=} q. \qquad \square \tag{38}$$

The enumeration technique used in our proof of the theorem just above can be modified with techniques from [26] so as to obtain a linear time computable learner for any given computably enumerable set of functions. However, in general it is not the case that any **XEx**-learnable set is also **XEx**-learnable by a linear time learner. This will be stated formally in Corollary 5.11 below, for which we now give definitions to set it up.

Definition 5.4. Let δ be a sequence acceptance criterion.

- δ is called *non-trivial* iff $\forall \sigma\colon\ \sigma \diamond \mathcal{R} \notin \mathbf{G}\delta$.
- Let $\mathcal{D} \subseteq \mathcal{R}$. δ *allows for linear time path finding for* $\mathcal{D}$ iff there is $r \in \textbf{LinF}$ such that, for all σ, τ of equal length and $q \in \mathcal{D}$ and e with $\sigma \subseteq q = \varphi_e$, we have $(\tau \diamond \lambda x\,\textbf{.}\,r(x, e, \sigma), q) \in \delta$.[19]
- Let $h \in \mathcal{R}, \mathcal{S} \subseteq \mathcal{R}$. Define $[\mathcal{S}]_h := \{\sigma \mid \exists g \in \mathcal{S}\colon\ \sigma \subseteq \mathbf{X}_2(h, g)\}$.

We illustrate the first two definitions by giving the following examples. The first deals with trivial acceptance criteria, the second with path finding.

Example 5.5. We have the following.

- $\mathbf{Bc}^*$ is a trivial sequence acceptance criterion, while **Ex**, **Bc**, **Bcs**, **Bcc** and **M** are non-trivial.
- **Ex**, **Bc** and $\mathbf{Bc}^*$ allow for linear time path finding for $\mathcal{R}$ as witnessed by $r = \lambda x, e, \sigma\,\textbf{.}\,e$. **M** allows for linear time path finding for total finite variants of constant functions.[20]

Note that non-triviality is inherited by subsets, and allowing for linear time path finding by supersets.

Lemma 5.6. *Let δ, δ' be such that δ' is non-trivial. Let $h, h' \in \mathcal{P}$ and $\mathcal{S} = \mathbf{X}\delta(h)$. Suppose there is σ such that $h(\sigma) \neq h'(\sigma)$. Then, if $\mathcal{S} \subseteq \mathbf{X}\delta'(h')$, $\sigma \notin [\mathcal{S}]_h$.*

[19] Intuitively, linear time path finding means the following. Suppose τ is a sequence of conjectures and σ a corresponding sequence of replies by the target. If the learner knows a program for the target, then it can identify it in the sense of δ (by outputting hypotheses according to $\lambda x\,\textbf{.}\,r(x, e, \tau)$).

[20] **M** allows for not necessarily linear time path finding for $\mathcal{R}$.

Proof. Suppose $\mathcal{S} \subseteq \mathbf{X}\delta'(h')$. By way of contradiction, suppose there is σ $\subseteq$-minimal such that

$$h(\sigma) \neq h'(\sigma) \tag{39}$$

and

$$\exists g \in \mathcal{S}\colon \sigma \subseteq \mathbf{X}_2(h, g). \tag{40}$$

Fix one such g. Let $c = \mathrm{len}(\sigma)$.

We show that h' witnesses $\sigma \diamond \mathcal{R} \in \mathbf{G}\delta'$. For all $f \in \mathcal{R}$, let $f' \in \mathcal{R}$ be such that

$$\forall \tau\colon f'(\tau) = \begin{cases} g(\tau), & \text{if } \tau \subseteq \mathbf{X}_1(h, g); \\ 0, & \text{else, if } \mathrm{len}(\tau) \leqslant c; \\ f(\mathrm{len}(\tau) - (c+1)), & \text{otherwise.} \end{cases} \tag{41}$$

For all $f \in \mathcal{R}$ we have $f' \in \mathcal{S}$, as f' mimics g on inputs consistent with h; thus, h cannot distinguish between g and f' and will learn both equally. Therefore,

$$\forall f \in \mathcal{R}\colon \mathbf{X}\big(h', f'\big) \in \delta'. \tag{42}$$

We have, for some $r \in \mathcal{R}$,

$$\forall f \in \mathcal{R}\colon \mathbf{X}\big(h', f'\big) \underset{(39)}{=} (r, \sigma \diamond f) \underset{(4)}{=} \mathbf{G}\big(h', \sigma \diamond f\big). \tag{43}$$

From (42) and (43), we get

$$\forall f \in \mathcal{R}\colon h\ \mathbf{G}\delta'\text{-learns } \sigma \diamond f. \tag{44}$$

Therefore, h' witnesses $\sigma \diamond \mathcal{R} \in \mathbf{G}\delta'$, a contradiction. □

Below we develop the concept of maximal learners.

Definition 5.7. Let δ be a sequence acceptance criterion and $h \in \mathcal{P}$. h is called $\mathbf{X}\delta$-*maximal* iff $\forall h' \in \mathcal{P}\colon \mathbf{X}\delta(h) \subseteq \mathbf{X}\delta(h') \Rightarrow h = h'$. Let $\mathcal{M}_{\mathbf{X}\delta}$ be the set of all $\mathbf{X}\delta$-maximal learners.

It is a consequence of the results in [8], that there are no maximal learners in the sense $\mathbf{G}\delta$, for $\delta \in \{\mathbf{Ex}^a, \mathbf{Bc}^n \mid a \in \mathbb{N} \cup \{*\}, n \in \mathbb{N}\}$ (the results given are even stronger, stating that every learner can be improved by an infinite set of targets). On the other hand, there are maximal $\mathbf{GBc}^*$ and $\mathbf{XM}$ learners ([8] and [22], respectively). The proof of Corollary 5.10 below employs the existence of $\mathbf{X}\delta$-maximal learner, for various δ including $\mathbf{M}$. The following proposition implicitly shows how to use maximal learners for separations.

Proposition 5.8. *Let δ be non-trivial. Let $\mathcal{C}, \mathcal{C}' \subseteq \mathcal{P}$.*

$$\mathcal{C}\mathbf{X}\delta \subseteq \mathcal{C}'\mathbf{X}\delta \quad \Rightarrow \quad \mathcal{M}_{\mathbf{X}\delta} \cap \mathcal{C} \subseteq \mathcal{C}'.$$

The following theorem characterizes maximal learners. In particular, the equivalence of (i) and (ii) just below is useful to show a given learner to be maximal.

Theorem 5.9. *Let δ be non-trivial such that $\forall \tau, \sigma\ \exists (p, q) \in \delta\colon \tau \subseteq p \wedge \sigma \subseteq q$. Let $h \in \mathcal{P}$. The following are equivalent.*

(i) *h is $\mathbf{X}\delta$-maximal.*
(ii) *$[\mathbf{X}\delta(h)]_h = \mathbb{S}\mathrm{eq}$.*
(iii) *$\forall \delta'$ non-trivial, $\forall h' \in \mathcal{P}\colon \mathbf{X}\delta(h) \nsubseteq \mathbf{X}\delta'(h') \Rightarrow h = h'$.*

Proof. "(i) ⇒ (ii)": Straightforward by showing the contrapositive.
"(ii) ⇒ (iii)": Use Lemma 5.6.
"(iii) ⇒ (i)": Trivial. □

Note that Theorem 5.9 applies to $\delta = \mathbf{M}$.

The two main results in this section are the Learner Correspondence Theorem (Theorem 5.10 below) and the Sequence Acceptance Correspondence Theorem (Theorem 5.12 below). Together, they characterize the relative learning power of many dynamic modeling criteria such as **LinFXEx** and **XBc**, and they even apply to coordination (**XM**). As a result, a comparison of the learning power of two learning criteria will usually be an immediate consequence.

Note that Theorems 5.10 and 5.12 do not inform about trade-offs between more restricted learner admissibilities and less restricted sequence acceptance criteria, or vice versa.

Theorem 5.10 *(Learner Correspondence). Let δ be non-trivial such that δ allows for linear time path finding for total finite variants of constant functions. Let $\mathcal{C}, \mathcal{C}' \subseteq \mathcal{R}$ be closed under generalized composition with* **LinF** *and* **LinF** $\subseteq \mathcal{C}, \mathcal{C}'$. *Then*

$$\mathcal{C}\mathbf{X}\delta \subseteq \mathcal{C}'\mathbf{X}\delta \quad \Leftrightarrow \quad \mathcal{C} \subseteq \mathcal{C}'.$$

Proof. "$\Leftarrow$": Immediate.

"$\Rightarrow$": Suppose, by way of contradiction, otherwise, as witnessed by $f \in \mathcal{C} \setminus \mathcal{C}'$. We set up to use Theorem 5.9. By linear time s-m-n, there is $e \in \mathbf{LinF}$ such that, for all σ, $\varphi_{e(\sigma)} = \sigma \diamond \lambda x\centerdot 0$. Define, for all σ, $\hat{\sigma}$ as the largest initial segment of σ that does not end in 0. Let $u \in \mathbf{LinF}$ such that $\forall \sigma\colon\ u(\sigma) = \max(2, \mathrm{len}(\hat{\sigma}))$. Let $r \in \mathbf{LinF}$ witness that δ allows for linear time path finding for total finite variants of constant functions. Define $h \in \mathcal{R}$ by

$$\forall \sigma\colon\ h(\sigma) = \begin{cases} 0, & \text{if } \sigma = \emptyset; \\ f(\sigma(0)), & \text{else if } \mathrm{len}(\sigma) = 1; \\ r(\mathrm{len}(\sigma) - u(\sigma), e(\hat{\sigma}), \sigma[u(\sigma)]), & \text{otherwise.} \end{cases} \tag{45}$$

It is straightforward that $h \in \mathcal{C}$, as $f \in \mathcal{C}$ and $\mathcal{C}$ is closed under generalized composition with **LinF** and **LinF** $\subseteq \mathcal{C}$. Also, $h \notin \mathcal{C}'$, as otherwise $f = h \circ \lambda n\centerdot \bar{n} \in \mathcal{C}'$. Using Proposition 5.8 and Theorem 5.9, it suffices to show $[\mathbf{X}\delta(h)]_h = \mathbb{S}\mathrm{eq}$ in order to derive a contradiction. Let σ be any sequence. Define $g = \lambda \rho\centerdot(\sigma \diamond \lambda x\centerdot 0)(\mathrm{len}(\rho))$. We have $\mathbf{X}_2(h, g) \underset{(6)}{=} \sigma \diamond \lambda x\centerdot 0$. From (45) and the choice of r it is now straightforward to verify that h $\mathbf{X}\delta$-learns g: Let $p, q \in \mathcal{R}$ be such that $\mathbf{X}(h, g) = (p, q)$. Then

$$q = \sigma \diamond \lambda x\centerdot 0. \tag{46}$$

For all $n \geqslant u(\sigma)$,

$$\widehat{q[n]} = \hat{\sigma}; \tag{47}$$

hence, as

$$\forall \tau, \tau'\colon\ \hat{\tau} = \hat{\tau}' \quad \Rightarrow \quad u(\tau) = u\big(\tau'\big), \tag{48}$$

we have

$$p(n) = h\big(q[n]\big) = r\big(n - u(\sigma), e(\hat{\sigma}), \sigma\big[u(\sigma)\big]\big). \tag{49}$$

Obviously, $\varphi_{e(\hat{\sigma})} = q$. Thus, $\sigma \in [\mathbf{X}\delta(h)]_h$. $\square$

Suppose for discussion Q is a polynomial time bound. Pitt [23] notes that polynomial time (update) 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 $Q(|\tau|)$ is enough time for h to think about σ as long as it needs. In fact, by an extension of Pitt's postponement tricks, **LinFGEx** = **GEx**. A direct application of Theorem 5.10, e.g., Corollary 5.11, implies that *dynamic modeling* does *not* allow for those kinds of postponement tricks *in general*.[21] Let α, as from [15, §21.4], be a *very slow growing*, unbounded, linear time computable function $\leqslant$ an inverse of Ackermann's function; let $\mathbf{LinF}^{+\varepsilon} := \{\varphi_e \in \mathcal{R} \mid \exists k \forall n\colon\ \Phi_e(n) \leqslant k \cdot |n| \cdot \log(|n|) \cdot \alpha(|n|) + k\}$.[22] The classes **LinF** and $\mathbf{LinF}^{+\varepsilon}$ have long been known to separate [18].

The following corollary gives a sample of the universal power of Theorem 5.10.

Corollary 5.11 *(Learner Complexity Matters). Let $\delta \in \{\mathbf{Ex}, \mathbf{Bc}, \mathbf{M}\}$.*

1. $\mathbf{LinFX}\delta \subset \mathbf{LinF}^{+\varepsilon}\mathbf{X}\delta$.
2. $\mathbf{PFX}\delta \subset \mathbf{EXPFX}\delta$.

Theorem 5.10 can be generalized so as to show $\mathcal{R}\mathbf{X}\delta \subseteq \mathcal{P}\mathbf{X}\delta$ for δ as in Corollary 5.11.

Theorem 5.12 *(Sequence Acceptance Correspondence). Let δ, δ' be non-trivial such that $\delta, \delta' \subseteq \mathcal{R}^2$. We have*

$$\mathbf{X}\delta \subseteq \mathbf{X}\delta' \quad \Leftrightarrow \quad \delta \subseteq \delta'.$$

Proof. "$\Leftarrow$": Immediate.

"$\Rightarrow$": Suppose, by way of contradiction, otherwise, as witnessed by $(p, q) \in \delta \setminus \delta'$.

[21] However, as we saw from Theorem 5.3 above, such extended postponement tricks *do*, nonetheless, apply to the *special case of* the **XEx**-learning of *any computably enumerable set of computable functions*.

However, we believe we can show that Theorem 5.3 doesn't hold for **LinFXPcpEx** in place of **LinFXEx**; hence, postdictive completeness prevents some postponement tricks.

[22] Trivially, $\mathbf{LinF}^{+\varepsilon}$ is contained in the set of quadratic time computable functions.

Let $h \in \mathcal{R}$ be such that $\forall \sigma\colon\ h(\sigma) = p(\mathrm{len}(\sigma))$. Let $\mathcal{S} = \mathbf{X}\delta(h)$. Obviously, $\mathcal{S} \in \mathbf{X}\delta$. Let $g \in \mathcal{R}$ be such that $\forall \sigma\colon\ g(\sigma) = q(\mathrm{len}(\sigma))$. Then $\mathbf{X}(h, g) = (p, q)$, and, therefore, $g \in \mathcal{S}$.

We have $\mathcal{S} \in \mathbf{X}\delta'$, as witnessed by, say, h'. As h' witnesses $\mathcal{S} \in \mathbf{X}\delta'$ and $\delta' \subseteq \mathcal{R}^2$, we have, for all c, $h'(q[c])\downarrow$.

Case 1: $\forall n\colon\ h'(q[n])\downarrow = p(n)$.

As $(p, q) \notin \delta'$, h' does not $\mathbf{X}\delta'$-learn $g \in \mathcal{S}$, a contradiction.

Case 2: There is c such that $h'(q[c])\downarrow \neq p(c)$.

Let $\sigma = q[c]$. We have $h(\sigma) \neq h'(\sigma)$ and $\sigma \in [\mathcal{S}]_h$ as witnessed by $g \in \mathcal{S}$. We use the contrapositive of the conclusion of Lemma 5.6 to infer that $\mathcal{S} \not\subseteq \mathbf{X}\delta'(h')$, a contradiction. □

The following corollary gives a sample of the universal power of Theorem 5.12.

Corollary 5.13 *(Hierarchies and Separations).*

1. *For all* $a, b \in \mathbb{N} \cup \{*\}$: $\mathbf{XBc}^a \not\subseteq \mathbf{XEx}^b$.
2. *For all* $a, b \in \mathbb{N} \cup \{*\}$: $\mathbf{XEx}^a \subseteq \mathbf{XBc}^b \Leftrightarrow a \leqslant b$.
3. *For all* $n \in \mathbb{N}$: $\mathbf{XM} \not\subseteq \mathbf{XEx}^*, \mathbf{XBc}^n$.
4. *For all* $n \in \mathbb{N}$: $\mathbf{XEx}^*, \mathbf{XBc}^n \not\subseteq \mathbf{XM}$.

Proof. By Remark 3.4 and Theorem 5.12. □

Acknowledgments

The authors would like to thank Samuel E. Moelius III for various fruitful discussions, especially on reactive learnees, as well as the anonymous reviewers for both the conference version and the journal version of this paper, which helped improve the current version.

References

[1] J. Bārzdiņš, Prognostication of automata and functions, Information Processing 1 (1971) 81–84.
[2] J. Bārzdiņš, Inductive inference of automata, functions and programs, in: Proc. of the International Congress of Mathematicians, 1974, pp. 455–560. English translation in American Mathematical Society Translations, Series 2, vol. 109, 1977, pp. 107–112.
[3] J. Bārzdiņš, Two theorems on the limiting synthesis of functions, in: Theory of Algorithms and Programs, vol. 210, Latvian State University, Riga, 1974, pp. 82–88.
[4] L. Blum, M. Blum, Toward a mathematical theory of inductive inference, Information and Control 28 (1975) 125–155.
[5] M. Blum, A. De Santis, S. Micali, G. Persiano, Noninteractive zero-knowledge, SIAM Journal on Computing 20 (6) (1991) 1084–1118.
[6] J. Case, Periodicity in generations of automata, Mathematical Systems Theory 8 (1974) 15–32.
[7] J. Case, Infinitary self-reference in learning theory, Journal of Experimental and Theoretical Artificial Intelligence 6 (1994) 3–16.
[8] J. Case, M. Fulk, Maximal machine learnable classes, Journal of Computer and System Sciences 58 (1999) 211–214.
[9] J. Case, S. Jain, F. Montagna, G. Simi, A. Sorbi, On learning to coordinate: Random bits help, insightful normal forms, and competency isomorphisms, Journal of Computer and System Sciences 71 (3) (2005) 308–332. Special issue for selected learning theory papers from COLT'03, FOCS'03, and STOC'03.
[10] J. Case, T. Kötzing, Dynamic modeling in inductive inference, in: Proc. of ALT (Algorithmic Learning Theory), 2008, pp. 404–418.
[11] J. Case, C. Smith, Comparison of identification criteria for machine inductive inference, Theoretical Computer Science 25 (1983) 193–220.
[12] H. Clark, Arenas of Language Use, University of Chicago Press, 1992.
[13] H. Clark, Using Language, Cambridge University Press, 1996.
[14] H. Clark, D. Wilkes-Gibbs, Referring as a collaborative process, Cognition 22 (1986) 1–39.
[15] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, second edition, MIT Press, 2001.
[16] R. Freivalds, E. Kinber, C. Smith, On the intrinsic complexity of learning, Information and Computation 123 (1995) 64–71.
[17] E. Gold, Language identification in the limit, Information and Control 10 (1967) 447–474.
[18] J. Hartmanis, R. Stearns, On the computational complexity of algorithms, Transactions of the American Mathematical Society 117 (1965) 285–306.
[19] T. Kötzing, Abstraction and complexity in computational learning in the limit, PhD thesis, University of Delaware, 2009. Available online at http://pqdtopen.proquest.com/#viewpdf?dispub=3373055.
[20] M. Li, P. Vitanyi, An Introduction to Kolmogorov Complexity and Its Applications, second edition, Springer Verlag, Heidelberg, 1997.
[21] E. Minicozzi, Some natural properties of strong identification in inductive inference, Theoretical Computer Science (1976) 345–360.
[22] F. Montagna, D. Osherson, Learning to coordinate: A recursion theoretic perspective, Synthese 118 (1999) 363–382.
[23] L. Pitt, Inductive inference, DFAs, and computational complexity, in: Proc. of AII (Analogical and Inductive Inference), 1989, pp. 18–44.
[24] K. Podnieks, Comparing various concepts of function prediction, in: Theory of Algorithms and Programs, vol. 210, 1974, pp. 68–81.
[25] H. Rogers, Theory of Recursive Functions and Effective Computability, McGraw–Hill, New York, 1967. Reprinted by MIT Press, Cambridge, MA, 1987.
[26] J. Royer, J. Case, Subrecursive Programming Systems: Complexity and Succinctness, Research Monograph in Progress in Theoretical Computer Science, Birkhäuser, Boston, 1994.
[27] R. Wiehagen, Limes-Erkennung rekursiver Funktionen durch spezielle Strategien, Elektronische Informationverarbeitung und Kybernetik 12 (1976) 93–99.