

**ABSTRACTION AND COMPLEXITY IN
COMPUTATIONAL LEARNING IN THE LIMIT**

by

Timo Kötzing

A dissertation submitted to the Faculty of the University of Delaware in
partial fulfillment of the requirements for the degree of Doctor of Philosophy in
Computer Science

Summer 2009

© 2009 Timo Kötzing
All Rights Reserved

**ABSTRACTION AND COMPLEXITY IN
COMPUTATIONAL LEARNING IN THE LIMIT**

by

Timo Kötzing

Approved: _____

B. David Saunders, Ph.D.

Chair of the Department of Computer and Information Sciences

Approved: _____

Tom Apple, Ph.D.

Dean of the College of Arts and Sciences

Approved: _____

Debra Hess Norris, M.S.

Vice Provost for Graduate and Professional Education

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____
John Case, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____
B. David Saunders, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____
Daniel Chester, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____
James Royer, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Dr. John Case, for all he has done for me. He patiently taught me the many facets of research, especially those pertaining to rhetoric. He spent many more hours on my education than I would have dared asking for, pointing me to many subtleties that I might have missed without his guidance.

Further, I would like to thank the remaining members of my committee, Dr. David Saunders, Dr. Daniel Chester and Dr. James S. Royer, for their reading of this manuscript and the many comments they made.

Importantly, I want to thank Dr. Samuel E. Moelius, who served as my role-model Ph.D. student for many years, who sat with me through many research meetings and whose careful reading of my many paper drafts resulted frequently in helpful comments.

Finally, I also want to thank the other researchers whom I had the honor to exchange ideas with in the course of my studies, especially Dr. Sandra Zilles and Dr. Jeffrey Heinz.

TABLE OF CONTENTS

LIST OF FIGURES	vii
ABSTRACT	viii
Chapter	
1 INTRODUCTION	1
1.1 Classification of Criteria	1
1.2 Abstraction	6
1.3 Complexity	8
1.4 Overview	10
2 MATHEMATICAL PRELIMINARIES	11
2.1 General Definitions	11
2.2 Runtime Complexity	17
2.3 Useful Theorems	21
3 ABSTRACTION OF COMPUTATIONAL LEARNING IN THE LIMIT	28
3.1 Learning Criteria Modules	28
3.2 Examples	33
3.3 Abstract Theorems	40
4 DELAYED POSTDICTIVE COMPLETENESS AND CONSISTENCY	44
4.1 Feasible Systems of Ordinal Notation	49
4.2 Countdown Graphs and Countdown Functions	56
4.3 Complexity Results	64

4.4	Results mostly not Comparing Countdown Graphs	69
4.5	Dependencies on the Countdown Graphs	80
4.6	Characterization of Reliable Identification	89
5	DIFFICULTIES IN FORCING FAIRNESS	92
5.1	Learning with Uniformly Polynomial time Decidable Hypothesis Spaces	94
5.2	Learning with Uniformly Decidable Hypothesis Spaces	98
5.3	Learning of ce Sets	108
5.4	Learning of Functions	115
6	DYNAMIC MODELING	119
6.1	Cooperation and Secretiveness	121
6.2	General Crossfeeding	129
	BIBLIOGRAPHY	141

LIST OF FIGURES

1.1	Ex-learning process	2
1.2	Nv-learning process	4
1.3	Coord-learning process	5
1.4	Categories of Limit Learning Criteria	7

ABSTRACT

In Computational Learning in the Limit, many criteria of successful learning have been proposed in the literature. For example, an algorithmic learner h tries to find a program for a computable function g (learner) given successively more values of g , each time outputting a conjectured program for g ; learning is considered successful, iff, from some point on, the learner always conjectures the same program p , and p is a program for g .

In the present thesis we will give a unifying framework, an abstraction of these criteria, and discuss its many merits. For example, a new learning paradigm, called *Dynamic Modeling*, is introduced. It actually arose out of the above abstraction.

Dynamic Modeling provides an idealization of, for example, a social interaction in which learner h seeks to discover program models of learner 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. The learner g , then, could try to *learn back* the behavior of learner h . We say that h *cooperatively* learns g iff h learns g and g learns h back. We say that h *secretively* learns g iff h learns g and g *cannot* learn h back. We show that there are non-trivial cases of both cooperative and secretive learning.

Furthermore, we will analyze efficient use of time in learning. Many notions of time restrictions suffer from unfair postponement tricks, i.e., the possibility to bypass runtime restrictions by postponing necessary computations. An important question to address is, then, how to avoid these postponement tricks and to achieve fair runtime restricted learning in the limit.

A learner is called *postdictively complete* iff all available data is correctly postdicted by each conjecture. The present thesis will show how postdictive completeness (and variants thereof) introduce some degree of fairness into runtime restricted learning in the limit.

Contrasting these latter results, we will point out many difficulties for introducing fairness into polynomial time restricted learning, for example by showing how several restrictions previously thought to forbid postponement tricks *fail* to do so.

Finally, some criteria in Dynamic Modeling are shown to naturally include some degree of fairness; however, some settings still allow for arbitrary postponement tricks.

Chapter 1

INTRODUCTION

We consider an *algorithmic learner* h , which is iteratively given more and more *finite* information generated by a *target function* g . From this information, the learner, in each iteration, (may) synthesize a (suitably interpreted) natural number as output. 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. The interplay of h and g is called a *learning process*. Depending on the context, some such learning processes constitute successful learning, others don't.

In the prior literature [JORS99], many *learning criteria* have been proposed. Each learning criterion specifies precisely, possibly among other things, how learner and target interplay, and which learning processes constitute successful learning.

1.1 Classification of Criteria

Many of the criteria given in the prior literature can be classified to fall into one of the categories of *identification*, *extrapolation* and *coordination*. We briefly consider some illustrative examples, giving one example per category named just above; later, we will give more examples for these categories.

Ex-learning [Gol67] in Definition 1.1.1 just below exemplifies the category of identification.¹

¹ The term 'Ex' stands for *explanatory* [CS83].

Definition 1.1.1. For a partial computable function h and a total computable function g , we say learner h *Ex-learns* learnee (or target) g iff, for all i (we think of i as the iteration number), h outputs a conjecture on input² $g(0) \dots g(i-1)$ and there are j_0, e such that $\forall j \geq j_0 : [h(g(0) \dots g(j-1)) = e]$ and e is a program for g .³

A program e as in the definition just above could be carried away and used *offline*.

Figure 1.1 gives an *informal* example of (the beginning of) an Ex-learning process. As can be seen in Figure 1.1, with each new datum the example learner h

input to h	program output of h
	“constantly 0”
1	“constantly 1”
1,2	“start with 1, add 1 each step”
1,2,3	“start with 1, add 1 each step”
1,2,3,5	“start with 1, add 1 each step, but leave out 4”
1,2,3,5,6	“start with 1, add 1 each step, but leave out 4”
1,2,3,5,6,7	“start with 1, add 1 each step, but leave out 4”
1,2,3,5,6,7,9	“start with 1, add 1 each step, but leave out multiples of 4”
⋮	⋮

Figure 1.1: Ex-learning process

sometimes refines the hypothesis. Let g be such that, for all n , $g(n) = n + 1 + \lfloor \frac{n}{3} \rfloor$, where, for all real numbers \mathbf{x} , $\lfloor \mathbf{x} \rfloor$ is the integer part of \mathbf{x} . Let us assume that the

² This input is a sequence, which is presented to the learner appropriately coded as given in Chapter 2.

³ Note that, starting in Chapter 3 below and, then, throughout the remainder of this Thesis, we treat Ex-learning technically a little differently from the present definition (Definition 1.1.1). Instead of presenting to an intended Ex-learner the data about a learnee g at iteration i as $g(0) \dots g(i-1)$, we will present it “canonically” as $(0, g(0) \dots ((i-1), g(i-1)))$. For each of these approaches to Ex-learning, the classes of computable functions g learnable by the class of partial computable h s will be the same.

input to h actually comes from taking initial segments of g (it is easy to verify that the first 7 values of g are indeed 1, 2, 3, 5, 6, 7, 9). h successfully Ex-learns g iff it stops refining its hypotheses after finitely much data and its final hypothesis is a correct program for g . Note that the bottom-most program stated in the second column of Figure 1.1 above *is* a correct program for g .

One motivation for Ex-learning is to model science: for example, on [BB75, Page 125] the following can be found.

Consider the physicist who looks for a law to explain a growing body of physical data. His data consist of a set of pairs (x, y) , where x describes a particular experiment, e.g., a high-energy physics experiment, and y describes the results obtained, e.g., the particles produced and their respective properties. The law he seeks is essentially an algorithm for computing the function $f(x) = y$.

Ex-learning models this learning scenario.

Next we define Nv-learning [Bār71, BB75] which exemplifies the category of *extrapolation*.⁴

Definition 1.1.2. For a partial computable function h and a total computable function g , we say h *Nv-learns* target g iff, h is total and, in the i -th iteration, h outputs a conjecture on input $g(0) \dots g(i-1)$ and there is a j_0 such that $\forall j \geq j_0 : h(g(0) \dots g(j-1)) = g(j)$.

The successful extrapolants $h(g(j) \dots g(j-1))$, $j \geq j_0$, can be used *online*.

The following gives an informal example of (the beginning of) an Nv-learning process.

In Figure 1.2 just above, the first, second and fourth prediction learner h makes are incorrect, all other predictions shown are correct. Let g be such that, for all n , $g(n) = (n \bmod 3) + 1$. Let us assume that the input to h actually comes from taking initial segments of g (it is easy to verify that the first 7 values of g are indeed

⁴ The term ‘Nv’ stands for *next value* [Bār71].

input to h	prediction output of h	next output of g
	0	1
1	1	2
1,2	3	3
1,2,3	4	1
1,2,3,1	2	2
1,2,3,1,2	3	3
1,2,3,1,2,3	1	1
1,2,3,1,2,3,1	2	2
\vdots	\vdots	\vdots

Figure 1.2: Nv-learning process

1, 2, 3, 1, 2, 3, 1). h successfully Nv-learns g iff it fails to predict the next datum only finitely often.

Again, science gives a possible motivation for Nv-learning: scientists hope to be able to make correct predictions about future events.

Next we define Coord-learning [MO99] which exemplifies the category of *co-ordination*.

Definition 1.1.3. For a partial computable function h and a total computable function g , we say h *Coord-learns* the 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*. Note that in Coord-learning, the learner is *reactive* in the sense that it can react to the outputs of the learner, contrasting with the *passive* learners in Ex- and Nv-learning. Further note that the learner cannot tell whether the learner is sensitive to the learner's outputs.

input to h	prediction output of h	input to g	output of g
	0		1
1	1	0	2
1,2	3	0,1	3
1,2,3	4	0,1,3	1
1,2,3,1	2	0,1,3,4	2
1,2,3,1,2	3	0,1,3,4,2	3
1,2,3,1,2,3	1	0,1,3,4,2,3	1
1,2,3,1,2,3,1	2	0,1,3,4,2,3,1	2
\vdots	\vdots	\vdots	\vdots

Figure 1.3: Coord-learning process

In Figure 1.3, the first, second and fourth prediction learner h makes are incorrect, all other predictions shown are correct. Let g be such that, for all finite sequences σ , $g(\sigma) = (\text{number of elements in } \sigma \bmod 3) + 1$. Let us assume that the input to h actually comes from this g (it is easy to verify that this g would have an input/output behavior as depicted in Figure 1.3). h successfully Coord-learns g iff it fails to predict the next datum only finitely often. Notice that g is an example of a learner in Coord-learning insensitive to the outputs made by the learner.

Coord-learning can be used for modeling predictions in science where the predictions made by the learner (may) affect the learner, as in quantum physics or social science.

Common among all the criteria given above is that successful learning only has to be established *in the limit*, that is, for any finite number of iterations the learner may give “bad” outputs (wrong hypotheses or incorrect predictions), while the remaining outputs have to be “good”. Hence, this kind of learning is referred to as *Learning in the Limit*.

Learning of Languages from Positive Examples

The present thesis, mainly in Chapter 5, will also analyze learning of languages

from positive examples. TxtEx-learning as defined just below exemplifies this kind of learning.

For a class of (at least computably enumerable) languages \mathcal{L} [HU79] 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 total function T enumerating (or presenting) all and only the elements of L (with or without pauses), as h is fed the succession of values $T(0), T(1), \dots$, it outputs a corresponding succession of programs $p(0), p(1), \dots$ from some hypothesis space, and, for some j_0 , for all $j \geq j_0$, $p(j)$ is a correct program for L , and $p(j+1) = p(j)$. The function T as just above is called a *text* or *presentation* for L .

The remainder of this chapter will first discuss the two main contents of and their interconnection in the present thesis (Sections 1.2 and 1.3). After that, an outline of the thesis will be given (Section 1.4).

1.2 Abstraction

The prior literature (as exemplified in [JORS99]) usually defines each learning criterion independently of others, even though many criteria share some concepts. The proposed thesis will give a unified notation for many criteria defined in the prior literature, covering all criteria discussed within the proposed thesis.

The benefits from this level of abstraction are manifold:

- (1) Achieving better understanding of particular learning criteria and their *relations with one another*.
- (2) Development of *unified problem solving techniques*.
- (3) Discovery of *new but natural learning criteria*.
- (4) Improvements in mathematical clarity, precision, *economy* and rigor.

For example, our abstraction of learning criteria lead to the above clear categorization of learning criteria. This categorization is summarized in Figure 1.4 below.

	Passive Learnee	Reactive Learnee
Off-Line	Identification	??
On-Line	Extrapolation	Coordination

Figure 1.4: Categories of Limit Learning Criteria

The categorization of Figure 1.4 shows how we found new natural criteria: there is a missing category entry for *off-line* with *reactive learner*. We refer to this category as *dynamic modeling*. An example of a dynamic modeling criterion is InteractiveBc-learning, given in Definition 1.2.1 just below.⁵

Definition 1.2.1. For a partial computable function h and a total computable function g , we say h *InteractiveBc-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, h always outputs a *program* for the infinite sequence of g 's resultant outputs.

Note that the sequence of h 's outputs is not required to converge to a single program.

Dynamic modeling is explored in Chapter 6. Note that thanks to our abstraction, some of the results about dynamic modeling also inform about coordination, and the relationship between coordination and dynamic modeling. For example,

⁵ **Bc** stands for *behaviorally correct*.

one theorem already published in [CK08a] has as immediate corollaries that learnability in linear time is strictly more restrictive than in quadratic time both for InteractiveBc-learning and Coord-learning.

In our *modular* approach, we define names for “pieces” of our criteria (Chapter 3). 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 named **TxtGEx** and InteractiveBc-learning will be called **XBc**.⁶ A similar modular approach appears already in [CK08a, CK08b].

1.3 Complexity

In the interest of introducing efficiency into a limit learning process, one can consider requiring the learner to be *computable in polynomial time* (see [Cob64, Edm65, Coo71, Kar72] regarding the connection between efficiency and polynomial time computability). Formally, we have the following definitions.

We say that h *Ex-learns* a targets g *in polynomial time* iff h Ex-learns g and there is a polynomial Q such that, for each i , h on g computes its i th hypothesis $p(i)$ within time $Q(|g(0), g(1), \dots, g(i-1)|)$.⁷ TxtEx-learning in polynomial time is defined analogously.

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 Ex- or TxtEx-learner. Then, for suitable polynomial Q , a variant of learner h can *postpone* outputting significant conjectures based on data σ until it has seen a much larger

⁶ In general, *standard* inductive inference criteria names will be changed to slightly different names in our modular approach.

⁷ $|g(0), g(1), \dots, g(i-1)|$, the total size of the sequence $g(0), g(1), \dots, g(i-1)$ is rigorously defined in Section 2.1.

sequence of data τ so that $Q(|\tau|)$ is enough time for h to think about σ as long as it needs.⁸

Hence, a requirement for polynomial time computability of the learner will not lose any learning power. These postponement tricks are “unfair”, as they show that no *efficiency* was actually introduced into the learning process. In this way the polynomial time restriction on each output does not, by itself, have the desirable effect of constraining the total learning time.

Pitt [Pit89] discusses some possible ways to forbid such *unfair postponement 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 postponement 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.⁹ In the present thesis, we focus, for polynomial time learners, on three of Yoshinaka’s properties: Postdictive completeness¹⁰, conservativeness, and prudence. *Postdictive completeness* [Bär74a, 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 *predict* a new datum. *Prudence* [Wei82, OSW86] requires each output hypothesis has to be

⁸ Pitt talks in this context of *delaying tricks*. We changed this terminology due to the clash with Akama and Zeugmann’s terminology for *delayed* postdictive completeness which we study in Chapter 4.

⁹ These spaces, by definition, 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 .

¹⁰ In the prior literature, except for [Ful88] and [CK08a, CK08b], what we call postdictive completeness is called *consistency*.

for a target that the learner actually learns.

In the present thesis we seek better understanding of the subtleties regarding fairness (and its degrees) for runtime restricted learning in the limit. Which requirements forbid some or all postponement tricks and what can be resultantly learned in polynomial time? What, if any, conditions which produce completely or reasonably fair polynomial time learning are mathematically necessary for fairness?

1.4 Overview

In Chapter 2 below we present mathematical preliminaries, including helpful theorems not specific to this thesis.

Chapter 3 presents our *unified* approach to limiting learning criteria. We state what kinds of modules we use for our learning criteria, give example modules and show how learning criteria found in the literature can be modeled using these modules. Finally, we give several helpful propositions.

The main results of this thesis can be found in Chapters 4 through 6.

Chapter 4 analyzes the impact of postdictive completeness (and many variants thereof) on fairness in polynomial time restricted learning. By subtly loosening the requirement of postdictive completeness, we show how there are *degrees* of fairness, i.e., degrees of how many postponement tricks are forbidden by a given loosening. Many results from this chapter have been published as [CPK07, CK08b].

Chapter 5 points out many difficulties in forcing fairness with different restrictions. Considered are postdictive completeness, conservativeness and prudence, as well as combinations and variants thereof.

Chapter 6 discusses dynamic modeling as introduced in Section 1.2 above. In particular, we show in what sense dynamic modeling criteria are naturally fair, and in what sense they still allow for postponement tricks. Many results from this chapter have been published as [CK08a].

Chapter 2

MATHEMATICAL PRELIMINARIES

This chapter introduces the basic mathematical notations and gives useful lemmas and theorems, most of which are not specific to this thesis and can be found elsewhere in the literature.

2.1 General Definitions

Complexity-theoretic notions follow [RC94]. General computability-theoretic notions follow [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, $\{0, 1, 2, \dots\}$. We do not distinguish between natural numbers and their *dyadic* representations as strings.¹

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 $\text{size}(w)$ to be the length of w . Since we identify each natural number x with its dyadic representation, for all $n \in \mathbb{N}$, $\text{size}(n)$ denotes the length of the dyadic representation of n . For all strings w , we define $|w|$ to be $\max\{1, \text{size}(w)\}$.²

¹ 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.

² This convention about $|\varepsilon| = 1$ helps with runtime considerations.

For two natural numbers m, n , we let

$$m \dot{-} n = \begin{cases} m - n, & \text{if } m \geq n; \\ 0, & \text{otherwise.} \end{cases}$$

“ $\dot{-}$ ” is pronounced “monus”.

We fix the 1-1 and onto pairing function $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ from [RC94, Section 2.3]. In particular, for all x, y ,

$$\langle x, y \rangle = \sum_{k=0}^m x_k 2^{2k+1} + \sum_{k=0}^n y_k 2^{2k}, \quad (2.1)$$

where $x = \sum_{k=0}^m x_k 2^k$, $y = \sum_{k=0}^n y_k 2^k$ and $x_0, \dots, x_m, y_0, \dots, y_n \in \{0, 1\}$. The *binary* representation of $\langle x, y \rangle$ is an interleaving of the *binary* representations of x and y , where we alternate x 's and y 's digits and start on the right with the least most significant y digit. For example, $\langle 15, 2 \rangle = 94$, since $15 = 1111$ (binary), $2 = 0010$ (binary), and $94 = 10101110$ (binary). Define π_1 and π_2 to be the functions such that, for all x and y ,

$$\pi_1(\langle x, y \rangle) = x; \quad (2.2)$$

$$\pi_2(\langle x, y \rangle) = y. \quad (2.3)$$

π_1 and π_2 are, respectively, called the first and second projection functions.

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 $A \setminus B = \{a \in A \mid a \notin B\}$, $\bar{A} = \mathbb{N} \setminus A$ and $\text{Pow}(A)$ 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 , $\text{card}(A)$ denotes the cardinality of A .

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

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

For two partial functions $f, g \in \mathfrak{P}$ and $a \in \mathbb{N}$, we write $f =^a g$ iff $\text{card}(\{x \in \mathbb{N} \mid f(x) \neq g(x)\}) \leq a$. For two partial functions $f, g \in \mathfrak{P}$, we write $f =^* g$ iff $\text{card}(\{x \in \mathbb{N} \mid f(x) \neq g(x)\})$ is finite.

For two partial functions $f, g \in \mathfrak{P}$, we let $f \circ g = \lambda x. f(g(x))$.

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

Whenever we consider tuples of natural numbers as input to $f \in \mathfrak{P}$, it is understood that the general coding function $\langle \cdot, \cdot \rangle$ is used to (left-associatively) code the tuples into a single natural number.

If $f \in \mathfrak{P}$ is not defined for some argument x , then we denote this fact by $f(x)\uparrow$, and we say that f on x *diverges*; the opposite is denoted by $f(x)\downarrow$, and we say that f on x *converges*. If f on x converges to p , then we denote this fact by $f(x)\downarrow = p$.

We say that $f \in \mathfrak{P}$ *converges to* p iff $\forall^\infty x : f(x)\downarrow = p$; we write $f \rightarrow p$ to denote this.³

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

For classes of function $\mathcal{C}, \mathcal{C}'$, we say that \mathcal{C} *is closed under generalized \mathcal{C}' composition* iff $\forall g \in \mathcal{C}, \forall f_0, \dots, f_n \in \mathcal{C}'$,

$$[\lambda x. f_0(g(f_1(x), \dots, f_n(x)))] \in \mathcal{C}. \quad (2.4)$$

Computability Notions

A partial function $\psi \in \mathfrak{P}$ is *partial computable* iff there is a deterministic, multi-tape Turing machine which, on input x , returns $\psi(x)$ if $\psi(x)\downarrow$, and loops

³ $f(x)$ converges should not be confused with f converges *to*.

infinitely if $\psi(x)\uparrow$. \mathcal{P} and \mathcal{R} denote, respectively, the set of all partial computable and the set of all computable functions $\mathbb{N} \rightarrow \mathbb{N}$. The functions in \mathcal{R} are called *computable functions*. Furthermore, $\mathcal{P}_{0,1}$ and $\mathcal{R}_{0,1}$ denote, respectively, the set of all partial computable and the set of all total (partial) computable functions $\mathbb{N} \rightarrow \{0, 1\}$.

φ^{TM} is the fixed programming system from [RC94, Chapter 3] for the partial computable functions $\mathbb{N} \rightarrow \mathbb{N}$. This system is based on deterministic, multi-tape Turing machines (TMs). In this system the φ -TM-programs are *efficiently* given numerical names or codes.⁴ Φ^{TM} denotes the TM step counting complexity measure also from [RC94, Chapter 3] and associated with φ^{TM} . In the present thesis, 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.

Let \mathcal{E} denote the set of all **ce** sets. We let W be the mapping such that $\forall e : W(e) = \text{dom}(\varphi_e)$. For each e , we write W_e instead of $W(e)$. W is, then, a mapping from \mathbb{N} *onto* \mathcal{E} . We say that e is an index (in W) for W_e .

Let $\text{UComp} = \{\{\varphi_{e(i)} \mid i \in \mathbb{N}\} \mid e \in \mathcal{R} : \forall i : \varphi_{e(i)} \in \mathcal{R}\}$ be the set of all sets of uniformly computable functions. Let $\text{UDec} = \{\{L_i \mid i \in \mathbb{N}\} \mid L : \mathbb{N} \rightarrow \mathcal{E} \wedge (\lambda x, i. x \in L_i) \in \mathcal{R}\}$ be the set of all uniformly decidable sets.⁵

⁴ 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.

⁵ We consider any predicate on natural numbers to be a $\{0, 1\}$ -valued function, where 0 represents false and 1 represents true. For example $\lambda x, i. x \in L_i$ corresponds to $\lambda x, i. \begin{cases} 1, & \text{if } x \in L_i; \\ 0, & \text{otherwise.} \end{cases}$

In this thesis, a *computable operator* is a mapping from one (respectively two) partial function(s) $\mathbb{N} \rightarrow \mathbb{N}$ into one (respectively two) such partial function(s) such that there exists an algorithm which, when fed *any* enumeration(s) of the graph(s) of the input(s), it outputs *some* enumeration(s) of the graph(s) of the output(s). For example, $\lambda f \in \mathfrak{P}.(f \circ f)$ is clearly a computable operator mapping any partial function f into its self-composition. Rogers [Rog67] extensively treats the one-ary case of these operators but calls them *recursive operators*.

Finite Sequences

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

Following [LV08], we define for all $x \in \mathbb{N}$: $\bar{x} = 1^{\text{size}(x)}0x$. Using this notation we can define a function $\langle \cdot \rangle_{\text{Seq}}$ coding arbitrarily long finite sequences of natural numbers into \mathbb{N} (represented dyadically) such that

$$\langle \sigma \rangle_{\text{Seq}} = \overline{\sigma(0)} \dots \overline{\sigma(\#\text{elems}(\sigma) - 1)}. \quad (2.5)$$

In particular, $\langle \emptyset \rangle_{\text{Seq}} = \varepsilon$.

For example the finite sequence $(4, 7, 10)_{\text{decimal}} = (01, 000, 011)_{\text{dyadic}}$ is coded as 11001 1110000 1110011 (but without the spaces, which were added for ease of reading).⁶

We use \diamond (with infix notation) to denote concatenation on sequences.

⁶ 1100111100001110011 is of course the dyadic representation of some number $\in \mathbb{N}$.

Note that, for all $\sigma, \tau : \langle \sigma \diamond \tau \rangle_{\text{seq}} = \langle \sigma \rangle_{\text{seq}} \langle \tau \rangle_{\text{seq}}$. Also note that, for all $x \in \mathbb{N}$, \bar{x} is equal to the code of the sequence of length 1 containing only x , and, for all $n \in \mathbb{N}$, \bar{x}^n is equal to the code of the sequence of length n , each element being x .

For any finite sequence σ such that $\#\text{elets}(\sigma) > 0$, we let $\text{last}(\sigma)$ be the last element of σ and σ^- be σ with its last element deleted. By convention, we set $\emptyset^- = \emptyset$.

Obviously, $\langle \cdot \rangle_{\text{seq}}$ is 1-1 [LV08]. The set of all sequences is decidable in linear time. The time to encode a sequence, that is, to compute $\lambda k, v_1, \dots, v_k. \langle v_1, \dots, v_k \rangle_{\text{seq}}$ is $\mathcal{O}(\lambda k, v_1, \dots, v_k. \sum_{i=1}^k |v_i|)$. Therefore, the size of the codeword is also linear in the size of the elements: $\lambda k, v_1, \dots, v_k. |\langle v_1, \dots, v_k \rangle_{\text{seq}}|$ is $\mathcal{O}(\lambda k, v_1, \dots, v_k. \sum_{i=1}^k |v_i|)$.⁷

Furthermore, we have

$$\forall x : 1 \leq \text{size}(\bar{x}); \quad (2.6)$$

$$\forall \sigma : \#\text{elets}(\sigma) \leq |\langle \sigma \rangle_{\text{seq}}|. \quad (2.7)$$

Henceforth, we will many times identify a finite sequence σ with its code number $\langle \sigma \rangle_{\text{seq}}$. 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 $f \in \mathfrak{P}$ and $i \in \mathbb{N}$, if $\forall j < i : f(j) \downarrow$, then $f[i]$ is defined to be the finite sequence $f(0), \dots, f(i-1)$. For every set of functions $\mathcal{S} \subseteq \mathcal{R}$, we let $[\mathcal{S}] = \{f[i] \mid f \in \mathcal{S}, i \in \mathbb{N}\}$.

Finite Sets

We fix the following 1-1 coding for all finite subsets of \mathbb{N} . For each non-empty finite set $D = \{x_0 < \dots < x_n\}$, $\langle x_0, \dots, x_n \rangle_{\text{seq}}$ is the code for D and $\langle \rangle_{\text{seq}}$ 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 $x \in D$, $\text{card}(D)$, $\max(D)$ and $D \subset D'$, we consider D and D' as *sets*, not as numbers.

⁷ For these \mathcal{O} -formulas, $|\varepsilon| = 1$ helps.

The symbol $\#$ is pronounced *pause* and is used to symbolize “no new input data” in a text. For each (possibly infinite) sequence q , let $\text{content}(q) = (\text{range}(q) \setminus \{\#\})$.

Later, in Chapter 3, we’ll type infinite sequences as being in \mathfrak{R} , *but*, technically, *texts* (for languages $\subseteq \mathbb{N}$) are *infinite sequences*, but they may contain pauses ($\#$ s) which are not natural numbers. Also, finite initial segments of texts are example finite sequences which can contain pauses. In our coding above of finite sequences, we code only sequences of natural numbers (and not pauses). To get around this, we will assume from now on that $\mathbb{N} \cup \{\#\}$ is efficiently coded 1-1 onto \mathbb{N} , say, by coding $\#$ as 0 and $n \in \mathbb{N}$ as $(n+1)$. In this way texts can be thought of as $\in \mathfrak{R}$ and finite initial segments of texts can, then, be coded as sequences of natural numbers. However, for texts T and for finite initial segments of such T , we will, whenever we need to talk about the value of $T(m)$, ignore the coding and work, for example, with whether the actual (not coded) values of $T(m) = \#$ or $= n \in \mathbb{N}$. This ignoring of the coding will be useful from time to time in Chapter 5.

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

2.2 Runtime Complexity

Let exp denote the function $\lambda x.2^x$. Furthermore, for all n , we write exp^n for the n -times self-composition of exp . In particular, exp^0 denotes the identity.

Let log denote the floor of the base-2 logarithm, with the exception of $\text{log}(0) = 0$.

Definition 2.2.1. We define

$$\text{LinPrograms} = \{e \mid \exists a, b, \forall x : \Phi_e(x) \leq a|x| + b\}; \quad (2.8)$$

$$\forall k : \text{Exp}_k\text{Programs} = \{e \mid e \in \mathbb{N} \wedge \exists p \text{ polynomial } \forall n \in \mathbb{N} : \Phi_e(n) \leq \text{exp}^k(p(|n|))\};^8 \quad (2.9)$$

$$\text{ExpPrograms} = \text{Exp}_1\text{Programs}; \quad (2.10)$$

$$\text{PolyPrograms} = \text{Exp}_0\text{Programs}. \quad (2.11)$$

Furthermore, we let

$$\mathbf{LinF} = \{\varphi_e \mid e \in \text{LinPrograms}\}; \quad (2.12)$$

$$\mathbf{PF} = \{\varphi_e \mid e \in \text{PolyPrograms}\}; \quad (2.13)$$

$$\mathbf{EXPF} = \{\varphi_e \mid e \in \text{ExpPrograms}\}; \quad (2.14)$$

$$\forall k : \mathbf{EXP}_k\mathbf{F} = \{\varphi_e \mid e \in \text{Exp}_k\text{Programs}\}. \quad (2.15)$$

For $g \in \mathbf{LinF}$ we say that g is *computable in linear time*.

For $g \in \mathbf{PF}$ we say that g is *computable in polynomial time*, or also, *feasibly computable*.

For $g \in \mathbf{EXPF}$ we say that g is *computable in exponential time*.

For $g \in \mathbf{EXP}_2\mathbf{F}$ we say that g is *computable in doubly exponential time*.

Exemplifying the above definitions, we have that the set of all functions computable in doubly exponential time is

$$\mathbf{EXP}_2\mathbf{F} = \{\varphi_e \mid \text{there is a polynomial } p \text{ such that } \forall n : \Phi_e(n) \leq 2^{2^{p(|n|)}}\}. \quad (2.16)$$

Definition 2.2.2. For all e, x, t , we write $\varphi_e(x) \downarrow_t$ iff $\Phi_e(x) \leq t$. Furthermore, we write

$$\forall e, x, t : \varphi_e(x) \downarrow_t = \begin{cases} \varphi_e(x), & \text{if } \Phi_e(x) \leq t; \\ 0, & \text{otherwise.} \end{cases} \quad (2.17)$$

⁸ As we are only interested in upper bounds, we only consider polynomials with natural numbers as coefficients.

Definition 2.2.3. From linear time s-m-n given in Theorem 2.3.1, there is a function $\text{patch} \in \mathbf{LinF}$ such that

$$\forall \sigma, e, x : \varphi_{\text{patch}(\sigma, e)}(x) = \begin{cases} \sigma(x), & \text{if } x \in \text{dom}(\sigma); \\ \varphi_e(x), & \text{otherwise.} \end{cases} \quad (2.18)$$

Also from linear time s-m-n given in Theorem 2.3.1, there is a function $\text{patch}_0 \in \mathbf{LinF}$ such that,

$$\forall \sigma, x : \varphi_{\text{patch}_0(\sigma)}(x) = \begin{cases} \sigma(x), & \text{if } x < \#\text{elets}(\sigma); \\ 0, & \text{otherwise.} \end{cases} \quad (2.19)$$

Definition 2.2.4. For $P \in \mathcal{P}_{0,1}$ and $z \in \mathbb{N}$, we let

$$\mu x.P(x, z) = \begin{cases} x, & \text{if } \forall y < x : P(y, z) \downarrow \neq 1 \wedge P(x, z) \downarrow = 1; \\ \uparrow, & \text{otherwise.} \end{cases} \quad (2.20)$$

Obviously, for all computable predicates P ,

$$[\lambda z. \mu x.P(x, z)] \in \mathcal{P}. \quad (2.21)$$

The following lemma asserts the algorithmicity of bounded minimization and bounded quantification.

Lemma 2.2.5. There exists a linear time computable function min such that

$$\forall p, x, m : \varphi_{\text{min}(p)}(x, m) = \begin{cases} y, & \text{if } y \text{ is the least number } \leq m \text{ such} \\ & \text{that: } \varphi_p(x, y) \downarrow \neq 0 \wedge \forall z < y : \\ & \varphi_p(x, z) = 0; \\ m + 1, & \text{if } \forall z \leq m : \varphi_p(x, z) \downarrow = 0; \\ \uparrow, & \text{otherwise;} \end{cases} \quad (2.22)$$

Furthermore, for $Q = \forall$ or $Q = \exists$, there is a linear time computable bndQ such that

$$\forall p, x, m : \varphi_{\text{bndQ}(p)}(x, m) = \begin{cases} \uparrow, & \text{if } \exists z \leq m : \varphi_p(x, z) \uparrow \\ 1, & \text{else if } Qz \leq m : \varphi_p(x, z) \downarrow \neq 0; \\ 0, & \text{otherwise.} \end{cases} \quad (2.23)$$

Proof: This follows from [RC94, Lemmas 3.15 & 3.16]. □

The following lemma is used in many of our detailed proofs for runtime analysis.

Lemma 2.2.6. Regarding time-bounded computability, we have the following.

- (i) The function $\lambda x. |x|$ is computable in linear time [RC94, Lemma 3.2].
- (ii) The functions $\lambda x, y. \langle x, y \rangle$, π_1 and $\pi_2 \in \mathbf{LinF}$ [RC94, Section 2.3].
- (iii) Equality checks are computable in linear time [RC94, Lemma 3.2].
- (iv) The log function is computable in linear time [RC94, Lemma 3.2].
- (v) function $\lambda k. 2^k$ is computable in time $\mathcal{O}(k)$ [RC94, Lemma 3.2].
- (vi) Conditional definition is computable in a time linear in the runtimes of its defining programs [RC94, Lemma 3.14].
- (vii) For all $p \in \text{PolyPrograms}$, we have that $\lambda x, m. \varphi_{\min(p)}(x, |m|)$ is computable in polynomial time [RC94, Lemma 3.15]. Hence, a similar bound holds for maximizations.
- (viii) Boolean combinations of predicates computable in polynomial time are computable in polynomial time [RC94, Lemma 3.18].
- (ix) For all $p \in \text{PolyPrograms}$, by [RC94, Lemma 3.16], we have that, for $Q = \forall$ or $Q = \exists$, $\lambda x, m. \varphi_{\text{bnd}Q(p)}(x, |m|)$ is computable in polynomial time.
- (x) 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.

(xi) Our coding above of finite sets enables content to be computable in polynomial time.⁹

(xii) From Definition 2.2.3, patch and patch_0 are computable in linear time.

(xiii) We have $\#\text{elets}$, $(\lambda\sigma, \tau. \sigma \subseteq \tau)$, $(\lambda\sigma, i. \sigma[i])$ and

$$\lambda\sigma, i. \begin{cases} \sigma(i), & \text{if } i < \#\text{elets}(\sigma); \\ 0, & \text{otherwise;} \end{cases} \quad (2.24)$$

are computable in linear time.¹⁰

(xiv) As finite sets are represented as ordered sequences without duplicates, from (xiii), $(\lambda x, D. x \in D)$, $(\lambda D. \text{card}(D))$ and $(\lambda D, D'. D \subseteq D')$ are computable in linear time.

2.3 Useful Theorems

In this section we give several general-purpose theorems which are applied throughout the thesis.

First, we state linear time s-m-n. Intuitively, linear time s-m-n provides a linear time computable function s , which, when applied to a program p and datum x , returns a program p' which, on argument y , calls program p with argument $\langle x, y \rangle$;

⁹ 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} .

¹⁰ These assertions can be seen by basic reasoning about multi-tape Turing machines in the φ -system. The most difficult part for proving the assertions is to make sure that certain counters can be maintained within suitable runtime bounds. The proof of [RC94, Lemma 3.2(1)] gives a runtime analysis for counting *down* from a given natural number. This analysis can be adopted for analyzing counting *up*.

thus, linear time s-m-n permits one to store (in linear time) arbitrary data (and hence, programs) inside any program.

Theorem 2.3.1. (Linear Time S-m-n – [RC94, Lemma 3.13]) There is $s \in \mathbf{LinF}$ such that

$$\forall p, x, y : \varphi_{s(p,x)}(y) = \varphi_p(x, y).$$

The following theorem states the existence of an infinite 1-1 onto *padding* function for the φ -system. Intuitively, padding, when applied to a program, produces a syntactically different, but semantically equivalent program. Importantly, an additional parameter allows for finding infinitely many *different* alterations to any given program.

Theorem 2.3.2. (Infinite 1-1 Onto Padding – [Moe09]) There is a 1-1 onto function $\text{pad} \in \mathcal{R}$ and $\text{unpad}_1, \text{unpad}_2 \in \mathcal{R}$ such that

$$\forall e, n \in \mathbb{N} : \varphi_{\text{pad}(e,n)} = \varphi_e; \tag{2.25}$$

$$\forall e, n \in \mathbb{N} : \text{unpad}_1(\text{pad}(e, n)) = e; \tag{2.26}$$

$$\forall e, n \in \mathbb{N} : \text{unpad}_2(\text{pad}(e, n)) = n. \tag{2.27}$$

Proof: We say that θ is an *effective programming system (eps)* for \mathcal{P} iff θ is a partial computable function written $\lambda p, x. (\theta_p(x))$ and such that $\{\theta_p \mid p \in \mathbb{N}\} = \mathcal{P}$. Intuitively, the p of θ_p should be thought of as (the code number of) a θ -program for the partial computable function θ_p . The partial computability of $\lambda p, x. (\theta_p(x))$ means that, for each p , the running of θ -program p on any input x can be algorithmically simulated.

We say θ is an *acceptable programming system (aps)* for \mathcal{P} iff θ is an effective programming system such that any eps η can be compiled into θ , i.e., such that there is a computable function t so that, for any eps η , for all η -programs p , $\eta_p = \theta_{t(p)}$.

Intuitively, $t(p)$ is the translation of η -program p into a semantically equivalent θ -program, $t(p)$. It turns out that example apses include φ , as well as any coding of programs for a modern programming language, such as Lisp, C or Java.

Define ψ such that, for each p and n ,

$$\psi_{\langle p, n \rangle} = \varphi_p. \quad (2.28)$$

Clearly, $t = \lambda p. (\langle p, 0 \rangle)$ translates/compiles the φ -system into the p -system, and, so ψ too is an acceptable programming system.

Then, since both φ and ψ are apses, by Rogers' Isomorphism Theorem [Rog58, §2], there exists a computable *bijection* $t : \mathbb{N} \rightarrow \mathbb{N}$ such that, for each p and n ,

$$\varphi_{t(\langle p, n \rangle)} = \psi_{\langle p, n \rangle}. \quad (2.29)$$

Let pad be such that, for each p and n ,

$$\text{pad}(p, n) = t(\langle p, n \rangle). \quad (2.30)$$

Clearly, pad is computable.

To show that pad is 1-1: suppose that p, q, m , and n are such that $\text{pad}(p, m) = \text{pad}(q, n)$. Then,

$$\begin{aligned} \text{pad}(p, m) = \text{pad}(q, n) &\Rightarrow t(\langle p, m \rangle) = t(\langle q, n \rangle) \quad \{\text{by (2.30)}\} \\ &\Rightarrow \langle p, m \rangle = \langle q, n \rangle \quad \{\text{because } t \text{ is 1-1}\} \\ &\Rightarrow p = q \wedge m = n \quad \{\text{because } \langle \cdot, \cdot \rangle \text{ is 1-1}\}. \end{aligned}$$

To show that pad is onto: let q be fixed. Then,

$$\begin{aligned} q &= (t \circ t^{-1})(q) && \{\text{because } (\forall p)[(t \circ t^{-1})(p) = p]\} \\ &= t(\langle (\pi_1^2 \circ t^{-1})(q), (\pi_2^2 \circ t^{-1})(q) \rangle) && \{\text{because } (\forall p)[\langle \pi_1^2(p), \pi_2^2(p) \rangle = p]\} \\ &= \text{pad}(\langle (\pi_1^2 \circ t^{-1})(q), (\pi_2^2 \circ t^{-1})(q) \rangle) && \{\text{by (2.30)}\}. \end{aligned}$$

Finally, for each p and n ,

$$\begin{aligned}\varphi_{\text{pad}(p,n)} &= \varphi_{t(\langle p,n \rangle)} \quad \{\text{by (2.30)}\} \\ &= \psi_{\langle p,n \rangle} \quad \{\text{by (2.29)}\} \\ &= \varphi_p \quad \{\text{by (2.28)}\}.\end{aligned}$$

□

Next is Kleene's Recursion Theorem (**KRT**). **KRT**, when applied to a program p , yields a “self-referential” program e that, on any input x , generates a (quiescent) copy of its own program “text” and subsequently uses that copy as an additional datum together with x on which to run program p .

Theorem 2.3.3. (Kleene Recursion Theorem – **KRT**, [Rog67, Page 214, Problem 11-4]) We have

$$\forall p \exists e \forall x : \varphi_e(x) = \varphi_p(e, x).$$

In our applications of **KRT**, the program p is implicit. For example, we can conclude from **KRT** that there is an e such that

$$\forall x : \varphi_e(x) = e + x^2. \tag{2.31}$$

Note that **KRT** follows from each of Theorems 2.3.4 and 2.3.5 below.

Next is Case's Operator Recursion Theorem (**ORT**), intuitively achieving *infinitary* self (and other) reference. That is, **ORT** provides a means of forming an infinite computable sequence of programs $h(0), h(1), \dots$ such that each program $h(i)$ knows all programs in the sequence and its own index i . A thorough explanation of **ORT** can be found in [Cas94].

Theorem 2.3.4. (Operator Recursion Theorem – **ORT**, [Cas74, Cas94]) For each computable operator Θ , there is a computable function $h \in \mathcal{R}$ such that

$$\forall n, x : \varphi_{h(n)}(x) = \Theta(h)(\langle n, x \rangle).$$

In most of our applications of **ORT**, the operator Θ is implicit. For example, we can conclude from **ORT** that there is $h \in \mathcal{R}$ such that

$$\forall n, x : \varphi_{h(n)}(x) = h(n + x) + x^2. \quad (2.32)$$

The following theorem gives a linear time version of **ORT** and implies Theorem 2.3.4.

Theorem 2.3.5. (Linear Time Operator Recursion Theorem – [Cas09]) Let Θ be a computable operator. There is $h \in \mathbf{LinF}$ such that

$$\forall n, x : \varphi_{h(n)}(x) = \Theta(h)(\langle n, x \rangle).$$

Proof: Let s be a linear time computable s-m-n function from Theorem 2.3.1 above.

Clearly there is a φ -program d such that,

$$\forall b, n, x : \varphi_d(\langle \langle b, n \rangle, x \rangle) = \Theta(\lambda m. s(b, \langle b, m \rangle))(\langle n, x \rangle). \quad (2.33)$$

Applying the s-m-n function s to the left-hand-side of (2.33), we have,

$$\forall b, n, x : \varphi_{s(d, \langle b, n \rangle)}(x) = \varphi_d(\langle \langle b, n \rangle, x \rangle). \quad (2.34)$$

Hence, by (2.33) and (2.34),

$$\forall b, n, x : \varphi_{s(d, \langle b, n \rangle)}(x) = \Theta(\lambda m. s(b, \langle b, m \rangle))(\langle n, x \rangle). \quad (2.35)$$

Then, setting $b = d$ in (2.35), we have,

$$\forall n, x : \varphi_{s(d, \langle d, n \rangle)}(x) = \Theta(\lambda m. s(d, \langle d, m \rangle))(\langle n, x \rangle). \quad (2.36)$$

Clearly, then, from (2.36),

$$h = \lambda x. s(d, \langle d, x \rangle) \quad (2.37)$$

defines a linear time function h as desired. □

The following theorem improves linear time padding and is a variant of Theorem 2.3.2.

Theorem 2.3.6. (Infinite 1-1 Linear Time Padding – [Cas09]) There is a 1-1 function $\text{pad} \in \mathbf{LinF}$ and $\text{unpad}_1, \text{unpad}_2 \in \mathcal{P}$ such that

$$\forall e, n \in \mathbb{N} : \varphi_{\text{pad}(e,n)} = \varphi_e; \quad (2.38)$$

$$\forall e, n \in \mathbb{N} : \text{unpad}_1(\text{pad}(e, n)) = e; \quad (2.39)$$

$$\forall e, n \in \mathbb{N} : \text{unpad}_2(\text{pad}(e, n)) = n. \quad (2.40)$$

Proof: By linear time **ORT** (Theorem 2.3.5), there is $p \in \mathbf{LinF}$ such that

$$\forall e, n, x : \varphi_{p(e,n)}(x) = \begin{cases} 0, & \text{if (i) } p(e, n) \in \{p(i) \mid i < \langle e, n \rangle\};^{11} \\ 1, & \text{else if (ii) } p(e, n) \in \{p(i) \mid \langle e, n \rangle < i \leq x\}; \\ \varphi_e(x) & \text{(iii) otherwise.} \end{cases} \quad (2.41)$$

Suppose for contradiction p is *not* 1-1. Let i , then, be least such that, for some strictly larger j , $p(i) = p(j)$. Then, on $x = j$, by clause (ii) of (2.41), $p(i)$ outputs 1, but, by clause (i) of (2.41), $p(j)$ outputs 0. Hence, $\varphi_{p(i)} \neq \varphi_{p(j)}$. Therefore, $p(i) \neq p(j)$, a contradiction. Hence, p is 1-1. Then, by clause (iii) of (2.41), for each e, n , $\varphi_{p(\langle e, n \rangle)} = \varphi_e$.

This proof, sans the use of **ORT** and linear time considerations, is based on a proof of infinite 1-1 padding in [MY78]. \square

The following theorem is an improvement of linear time s-m-n (Theorem 2.3.1).

Theorem 2.3.7. (1-1 Linear Time S-m-n – [Cas09]) There is a 1-1 $s \in \mathbf{LinF}$ such that

$$\forall p, x, y : \varphi_{s(p,x)}(y) = \varphi_p(x, y).$$

¹¹ Note that, as $\langle \cdot, \cdot \rangle$ is onto, each natural number i equals the code of two natural numbers e, n , $\langle e, n \rangle$.

Proof: By Theorem 2.3.1, let $s' \in \mathbf{LinF}$ be such that

$$\forall p, x, y : \varphi_{s'(p,x)}(y) = \varphi_p(x, y). \quad (2.42)$$

Using linear time padding from Theorem 2.3.6, let $s \in \mathbf{LinF}$ be such that

$$\forall p, x : s(p, x) = \text{pad}(s'(p, x), \langle p, x \rangle). \quad (2.43)$$

Clearly, s is 1-1 and we have, for all p, x, y ,

$$\varphi_{s(p,x)}(y) = \varphi_{\text{pad}(s'(p,x), \langle p, x \rangle)}(y) = \varphi_{s'(p,x)}(y) = \varphi_p(x, y). \quad (2.44)$$

□

The following theorem improves **ORT** further to a 1-1 linear time version of **ORT**.

Theorem 2.3.8. (1-1 Linear Time Operator Recursion Theorem – [Cas09]) For each computable operator Θ , there is a 1-1 function $h \in \mathbf{LinF}$ such that

$$\forall n, x : \varphi_{h(n)}(x) = \Theta(h)(\langle n, x \rangle).$$

Proof: Use the same proof as that of Theorem 2.3.5 for linear time **ORT**, but with 1-1 linear time s-m-n (Theorem 2.3.7) instead of linear time s-m-n. □

Chapter 3

ABSTRACTION OF COMPUTATIONAL LEARNING IN THE LIMIT

In this chapter, we first give our modular definition of what a learning criterion is, interleaved with some few examples, in Section 3.1. After that, Section 3.2 presents many more examples and shows how learning criteria found in the literature can be expressed within our modular approach. Section 3.3 gives abstract theorems that hold for wide classes of learning criteria, as well as the technically useful notions of locking sequence and locking set and a proposition about these notions.

3.1 Learning Criteria Modules

In this section we give our modular definition of what a learning criterion is. As noted above, all *standard* inductive inference criteria names will be changed to slightly different names in our modular approach.

In the interest of generality, we give many definitions for limit learning also involving *non*-algorithmic learning. Nonetheless, all of the *results* given in Chapters 4 through 6 concern only *algorithmic* (including complexity bounded) learning.

Definition 3.1.1. An *effective numbering of ce languages* is a function $V : \mathbb{N} \rightarrow \mathcal{E}$ such that there is a function $f \in \mathcal{P}$ with $\forall e, x : (f(e, x) \downarrow \Leftrightarrow x \in V(e))$.¹ For such numberings V , for each e , we will write V_e instead of $V(e)$, and call e an *index* or

¹ 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.

an *hypothesis* (in V) for V_e . We use effective numberings as hypothesis spaces for our learners. Examples include W and φ .²

For the definitions and examples to follow, recall our convention from Section 2.1 above that, while texts (for languages) can contain $\#$ s, we efficiently code $\mathbb{N} \cup \{\#\}$ 1-1 onto \mathbb{N} , so that texts are actually in \mathfrak{A} and so that their finite initial segments are actually finite sequences of natural numbers.

Definition 3.1.2. 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. Note that any two learner admissibility restrictions \mathcal{C} and \mathcal{C}' can be combined by intersecting them. For ease of notation we write $\mathcal{C}\mathcal{C}'$ instead of $\mathcal{C} \cap \mathcal{C}'$.

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

Definition 3.1.3. Any function from \mathcal{R} to $\text{Pow}(\mathfrak{A})$ is called a *target presenter* for the total computable functions. Any function from \mathcal{E} to $\text{Pow}(\mathfrak{A})$ is called a *target presenter* for the **ce** languages.

For each $\mathfrak{F} \subseteq \mathfrak{A}$, we define the following target presenters.

- **Txt** ^{\mathfrak{F}} : $\mathcal{E} \rightarrow \text{Pow}(\mathfrak{A}), L \mapsto \{\rho \in \mathfrak{F} \mid \text{content}(\rho) = L\}$.
- **Arb** ^{\mathfrak{F}} : $\mathcal{R} \rightarrow \text{Pow}(\mathfrak{A}), g \mapsto \text{Txt}^{\mathfrak{F}}(g)$.³
- **Canonical** : $\mathcal{R} \rightarrow \text{Pow}(\mathfrak{A}), g \mapsto \{\lambda x. \langle x, g(x) \rangle\}$.
- **Immediate** : $\mathcal{R} \rightarrow \text{Pow}(\mathfrak{A}), g \mapsto \{g\}$.

² Recall, from Section 2.1, that we identify partial functions with their graphs.

³ Recall that we identify a function in \mathfrak{P} with its (coded) graph. In particular, each total computable function is a **ce** set.

When denoting criteria with **Canonical** as the target presenter, we will omit **Canonical**. We write **Txt** instead of $\mathbf{Txt}^{\mathfrak{R}}$ and **Arb** instead of $\mathbf{Arb}^{\mathfrak{R}}$.⁴

Note the slight difference between **Canonical** and **Immediate**: intuitively, any datum presented by **Canonical** is an argument/value pair, while **Immediate** only presents a value. For passive learners we use **Canonical** (to present the graph of the learner to the learner), while for active learners we use **Immediate**.

Definition 3.1.4. Every computable operator $\mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P}^2$ is called a *sequence generating operator*.⁵ Intuitively, a sequence generating operator defines how learner and presentation interact to generate two infinite sequences, one for learner-outputs and one for learner-outputs.

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

The two most-used sequence generating operators in this thesis are **G** and **X**, defined as follows.

- Goldstyle [Gol67]: $\mathbf{G} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (\lambda i. h(g[i]), g)$.
- Crossfeeding [MO99] $\mathbf{X} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{P}, (h, g) \mapsto (p, q)$ such that $\forall n p(n) = h(q[n]) \wedge q(n) = g(p[n])$.⁶

“**G**” is a reference to Gold [Gol67]. Intuitively, **G** takes a learner h and a learner g , and feeds longer and longer initial segments of g into h , considering the successive

⁴ We will not be concerned with other choices for \mathfrak{F} than \mathfrak{R} . However, one can use the parameter \mathfrak{F} to model learning from presentations from specific kinds of text only, such as *computable text*, *primitive recursive text* or *fat text* (see [JORS99]).

⁵ As noted above in Section 2, *these* computable operators are the recursive operators of [Rog67] but with two arguments and two outputs; however, here they are also restricted to the indicated domain.

⁶ **X** we pronounce *cross*, and it is short for *crossfeed*.

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 . Regarding \mathbf{X} , learner and learner have symmetrical information in each iteration.

Definition 3.1.5. Every subset of \mathcal{P}^2 is called a *sequence acceptance criterion*. Intuitively, a sequence acceptance criterion defines which 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 V of some **ce** languages, we define the following sequence acceptance criteria.⁷ Most of these sequence acceptance criteria exist in two variants, one for passive learners, one for active learners. For passive learners, we let $\gamma = \text{content}$, and for active learners we let γ be the identity on $\mathfrak{P} \cup \text{Seq}$.

- Explanatory: $\mathbf{Ex}_V = \{(p, q) \in \mathfrak{P}^2 \mid \exists e : p \rightarrow e \text{ and } V_e = \gamma(q)\}$.
- Behaviorally correct: $\mathbf{Bc}_V = \{(p, q) \in \mathfrak{P}^2 \mid \forall^\infty n V_{p(n)} = \gamma(q)\}$.
- Always giving hypotheses: \mathfrak{R}^2 .
- Postdictive Completeness: $\mathbf{Pcp}_V = \{(p, q) \in \mathfrak{R}^2 \mid \forall i : \gamma(q[i]) \subseteq V_{p(i)}\}$.
- Conservativeness: $\mathbf{Conv}_V = \{(p, q) \in \mathfrak{R}^2 \mid \forall i : p(i) \neq p(i+1) \Rightarrow \gamma(q[i+1]) \not\subseteq V_{p(i)}\}$.
- Matching [Bär71, BB75, MO99]: $\mathbf{M} = \{(p, q) \in \mathfrak{P} \times \mathfrak{R} \mid p =^* q\}$.

Definition 3.1.6. 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*:

$$\mathcal{T}\delta = \{h \in \mathcal{P} \mid \forall T \in \text{range}(\alpha) : \beta(h, T) \in \delta\}.$$

⁷ Recall, from Definition 3.1.1, that function learning is covered, as the range of V might be, for example, the set of all graphs of partial computable functions.

We then speak of “total” For example, *total* postdictive completeness in W , i.e., \mathcal{TPcp}_W , requires postdictive completeness on *any* input data, including input data not necessarily taken from a target to be learned.⁸

Definition 3.1.7. For now, 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.⁹ Let $\mathcal{C}, \alpha, \beta, \delta$ be, respectively, a learner admissibility restriction, a target presenter, a sequence generating operator and a sequence acceptance criterion. It is understood that in learning criteria involving passive learners, we use $\gamma = \text{content}$, otherwise we use γ equal to the identity on $\mathfrak{P} \cup \text{Seq}$.

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

$$\mathcal{C}\alpha\beta\delta = \{\mathcal{L} \subseteq \mathcal{E} \mid \exists h \in \mathfrak{P} : h \text{ } (\mathcal{C}, \alpha, \beta, \delta)\text{-learns } \mathcal{L}\}. \quad (3.1)$$

We refer to the sets $\mathcal{C}\alpha\beta\delta$ as in (3.1) 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 learners $(\mathcal{C}, \alpha, \beta, \delta)$ -learned by h is denoted by

$$\mathcal{C}\alpha\beta\delta(h) = \{L \in \mathcal{E} \mid h \text{ } (\mathcal{C}, \alpha, \beta, \delta)\text{-learns } L\}.$$

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

⁸ This would force any learner to be total computable.

⁹ We extend this notion of a learning criterion slightly in Definition 3.2.5 below.

3.2 Examples

In this section we give many more 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. Note that we repeat many of the definitions given in Section 3.1 for convenience.

Example 3.2.1. Effective numberings include the following important examples.

- W .
- φ .
- A canonical numbering of all regular languages, represented by efficiently coded DFAs. (where membership is trivially uniformly polynomial time decidable and (5.1) below 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, membership is uniformly polynomial time decidable [HU79, Sch91], and (5.1) below holds. These grammars are called CFGs with Prohibition in [Bur05].¹⁰

Example 3.2.2. 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 linear time computable functions) may be used as a learner admissibility restriction.

Example 3.2.3. We define the following example sequence generating operators. All learners h typically give an initial conjecture based on no data.

¹⁰ 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.

- Goldstyle [Gol67]: $\mathbf{G} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (\lambda i. h(g[i]), g)$.
- Set-driven [WC80, JORS99]: $\mathbf{Sd} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (\lambda i. h(\text{content}(g[i])), g)$.
- Partly set-driven [SR84, JORS99]: $\mathbf{Psd} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (\lambda i. h(\text{content}(g[i]), i), g)$.
- Iterative [Ful85, Wie76]: $\mathbf{It} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (p, q)$ such that $p(0) = 0$, $\forall n : p(n+1) = h(q(n), p(n))$ and $q = g$.
- Transductive¹¹: $\mathbf{Td} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (p, q)$ such that $p(0) = 0$, $\forall n : p(n+1) = h(q(n))$ and $q = g$.
- Crossfeeding [MO99] $\mathbf{X} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{P}, (h, g) \mapsto (p, q)$ such that $\forall n p(n) = h(q[n]) \wedge q(n) = g(p[n])$.
- Learnee Iterative¹²: $\mathbf{Li} : \mathfrak{P} \times \mathfrak{R} \rightarrow \mathfrak{P} \times \mathfrak{P}, (h, g) \mapsto (p, q)$ such that $\forall n p(n) = h(q[n]) \wedge q(0) = 0 \wedge \forall n : q(n+1) = g(p(n), q(n))$.

This paragraph is a reminder of a paragraph in Definition 3.1.4. “ \mathbf{G} ” is a reference to Gold [Gol67]. Intuitively, \mathbf{G} takes a learner h and a learner 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 learner, while the learner does not react at all to the learning process.

¹¹ \mathbf{Td} is implicit in [CCJS07] as the $m = 1$, $n = 0$ cases of the two families of learning criteria $(\mathbf{BMS}_m)_{m>0}$ and $(\mathbf{MLF}_n)_{n \geq 0}$.

¹² \mathbf{Li} is not from the prior literature and is not used in later chapters; it is given here for illustrative purposes.

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 with respect to **G**.

The first three bullets given in Example 3.2.3 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 importance to Chapter 6 of this thesis. Let $f, g, h, p, q \in \mathcal{P}$.

$$\mathbf{X}(h, g) = (p, q) \Rightarrow \mathbf{G}(h, q) = (p, q). \quad (3.2)$$

$$\mathbf{X}(h, g) = (p, q) \Rightarrow \mathbf{G}(g, p) = (q, p). \quad (3.3)$$

$$\mathbf{X}(h, \lambda\sigma.f(\#\text{elets}(\sigma))) = \mathbf{G}(h, f). \quad (3.4)$$

Example 3.2.4. We define the following sequence acceptance criteria. Let V be an effective numbering of some **ce** sets. Note that **M** is independent of any hypothesis space, and that **Pcs** is only defined for φ . Again, most of these sequence acceptance criteria exists in two variants, one for passive learnee, one for active learnee. Recall that, for passive learnee, we let $\gamma = \text{content}$, and for active learnee we let γ be the identity on \mathfrak{P} .

- Explanatory: $\mathbf{Ex}_V = \{(p, q) \in \mathfrak{P}^2 \mid \exists e : p \rightarrow e \text{ and } V_e = \gamma(q)\}$.

- Explanatory with up to $a \in \mathbb{N} \cup \{*\}$ errors [CS83, BB75]:

$$\mathbf{Ex}_V^a = \{(p, q) \in \mathfrak{P}^2 \mid \exists e : p \rightarrow e \text{ and } V_e =^a \gamma(q)\}.$$

- Behaviorally correct [CS83, Bär74b]: $\mathbf{Bc}_V = \{(p, q) \in \mathfrak{P}^2 \mid \forall^\infty n V_{p(n)} = \gamma(q)\}$.

- Behaviorally correct with up to $a \in \mathbb{N} \cup \{*\}$ errors [CS83]:

$$\mathbf{Bc}_V^a = \{(p, q) \in \mathfrak{P}^2 \mid \forall^\infty n V_{p(n)} =^a \gamma(q)\}.$$

- Matching [Bār71, BB75, MO99]: $\mathbf{M} = \{(p, q) \in \mathfrak{P} \times \mathfrak{R} \mid p =^* q\}$.
- Postdictively complete [Bār74a, BB75, Wie76]: $\mathbf{Pcp}_V = \{(p, q) \in \mathfrak{R}^2 \mid \forall n : \gamma(q[n]) \subseteq \varphi_{p(n)}\}$.
- Postdictively consistent [Wie78]: $\mathbf{Pcs}_\varphi = \{(p, q) \in \mathfrak{R}^2 \mid \forall n \forall i < n : \varphi_{p(n)}(i) \downarrow \Rightarrow q(i) \in \varphi_{p(n)}\}$.
- Hypotheses are programs for total functions [CS83, CJNM94]: $\mathbf{T}_V = \{(p, q) \in \mathfrak{R}^2 \mid \forall n : V_{p(n)} \in \mathcal{R}\}$.
- Always giving hypotheses: \mathfrak{R}^2 .

In Chapter 4, we will define variants of \mathbf{Pcp}_V and \mathbf{Pcs}_V .

The following definition extends the notion of a learning criterion as given in Definition 3.1.7 above.

Definition 3.2.5. Let \mathcal{C} , α , β , δ , V , respectively, be a learner admissibility restriction, a target presenter, a sequence generating operator, a sequence acceptance criterion and an effective numbering of **ce** languages.

We let Id be the function mapping the learning criterion $(\mathcal{C}, \alpha, \beta, \delta)$ to the set $\mathcal{C}\alpha\beta\delta$, as defined in (3.1).

We define two versions of prudent learning [Wei82, OSW86] as follows.

$$\mathbf{Prud}_V(\mathcal{C}, \alpha, \beta, \delta) = \{\mathcal{L} \subseteq \text{dom}(\alpha) \mid \begin{array}{l} \exists h \in \mathcal{C}, \exists \mathcal{L}' : \mathcal{L} \subseteq \mathcal{L}' \subseteq \alpha\beta\delta(h) \wedge \\ \forall t \in \mathcal{L}', \forall T \in \alpha(t), \forall i : V_{\beta_1(h, T)(i)} \in \mathcal{L}' \end{array}\},$$

and

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

Intuitively, a prudent learner may only output conjectures for targets it successfully learns (in the case of \mathbf{Prud}_V the only restricted outputs are those on the learner's way to successful learning).

Furthermore, for each set $\mathfrak{F} \subseteq \mathfrak{R}$, we make the following definition pertaining to *reliable identification* as in [Min76, BB75].

$$\text{Rel}_{\mathfrak{F}}(\mathcal{C}, \alpha, \beta, \delta) = \{\mathcal{L} \subseteq \text{dom}(\alpha) \mid \begin{array}{l} \exists h \in \mathcal{C} : \mathcal{L} \subseteq \alpha\beta\delta(h) \wedge \forall g \in \mathfrak{F} \forall p, q \in \mathcal{P} : \\ (\beta(h, g) = (p, q) \wedge \exists e : p \rightarrow e) \Rightarrow (p, q) \in \delta \}. \end{array}$$

Intuitively, in reliable learning, a learner may only converge on targets (in \mathfrak{F}) it successfully learns. Considered in this thesis are $\text{Rel}_{\mathfrak{R}}$ and $\text{Rel}_{\mathcal{R}}$.

For $\mathcal{D} \in \{\text{Id}, \mathbf{Prud}, \mathcal{T}\mathbf{Prud}\} \cup \{\text{Rel}_{\mathfrak{F}} \mid \mathfrak{F} \subseteq \mathfrak{R}\}$, a learning criterion¹³ C and a learner h , we write $\mathcal{D}C$ instead of $\mathcal{D}(C)$ and *now* we consider $\mathcal{D}C$ to be a *learning criterion*; further, we let $\mathcal{D}C(h)$ denote the set of all targets learnable by h for learning criterion $\mathcal{D}C$.

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 in the following two tables are for *algorithmic* learners. Furthermore, the default target presenter is **Canonical**; therefore, all learning criteria in the just below table are for function learning.

Note that many criteria have more than one learner admissibility restriction or more than one sequence acceptance criterion, combined, in each case, as noted above, using set-intersection.

¹³ Recall that each learning criterion is a 4-tuple in the domain of \mathcal{D} .

Ex ([Gol67, BB75, CS83])	\leftrightarrow	GEx_φ
Bc ([Bār74b, CS83])	\leftrightarrow	GBc_φ
Nv ([Bār71, BB75, CS83, CJNM94])	\leftrightarrow	RG_M
Nv' ([BF72, CS83, CJNM94])	\leftrightarrow	GR²M
Nv'' ([Pod74, CS83, CJNM94])	\leftrightarrow	GM
Cons ([Bār74a, CJSW04])	\leftrightarrow	GPcpEx_φ
R-Cons ([JB82, CJSW04])	\leftrightarrow	RG_PGPcpEx_φ
T-Cons ([WL76, CJSW04])	\leftrightarrow	TPcpGEx_φ
Reliable on \mathcal{R} ([Min76, BB75])	\leftrightarrow	Rel_ℛGEx_φ
It ([Wie76])	\leftrightarrow	ItEx_φ
learnable by a player ([MO99])	\leftrightarrow	ImmediateXM
learnable by a total player ([MO99])	\leftrightarrow	ℛImmediateXM

Regarding language learning from positive data, we write the criterion of TxtEx-learning, with V -indices for the hypothesis space, as **TxtGEx_V**. However, **TxtGEx_V** *with* the three restrictions of total postdictive completeness, total conservativeness *and* prudence (not total prudence) in our modular notation is written **PrudTPcpTConvTxtGEx_V**, which we abbreviate as **PrudT(PcpConv)TxtGEx_V**. If, instead, we wanted this criterion *but* with *total* prudence in the place of prudence, it could be written **TPrudT(PcpConv)TxtGEx_V**.

The next table displays further examples of language learning criteria.

TxtEx ([Gol67])	\leftrightarrow	TxtGEx_W
TxtCons ([Ang80])	\leftrightarrow	TxtGPcpEx_W
polynomial time TxtEx-learning	\leftrightarrow	PFTxtGEx_W
consistent, conservative and prudent TxtEx-learning	\leftrightarrow	TPrudTxtGPcpConvEx_W
totally consistent, totally conservative and prudent TxtEx-learning	\leftrightarrow	TPrudT(PcpConv)TxtGEx_W

As a possible aid to the reader, we explain next, in some detail, how each of the last three example learning criteria just above fit the abstract and (extended) notion of a learning criterion

$$\mathcal{D}(\mathcal{C}, \alpha, \beta, \delta)$$

from Definition 3.2.5 above.

In the third from last example just above, $\mathcal{D} = \text{Id}$, $\mathcal{C} = \mathbf{PF}$, $\alpha = \mathbf{Txt}$, $\beta = \mathbf{G}$, and $\delta = \mathbf{Ex}_W$.

In the second last example just above, $\mathcal{D} = \mathbf{TPrud}$, $\mathcal{C} = \mathcal{P}$, $\alpha = \mathbf{Txt}$, $\beta = \mathbf{G}$, and $\delta = \mathbf{PcpConvEx}_W$.

In the last example just above, $\mathcal{D} = \mathbf{TPrud}$, $\mathcal{C} = \mathcal{T}(\mathbf{PcpConv})$, $\alpha = \mathbf{Txt}$, $\beta = \mathbf{G}$, and $\delta = \mathbf{Ex}_W$.

The idea of somehow dividing a learning criterion is not entirely new. For example, Freivalds et. al. [FKS95] defined *admissible sequences for a given function*, which basically defines a binary predicate on a pair of infinite sequences.

Furthermore, similarities between extrapolation (like **GM**) and coordination (like **XM**) have been pointed out in [CJM⁺05]. In particular, *blind* learners are defined as functions where each output depends only on the *length* of it's input, and with each function $g \in \mathcal{R}$, a blind learner $g' = \lambda\sigma.g(\#\text{elets}(\sigma))$ is associated.

The mapping $\Theta = \lambda g.g'$ is, then, a natural embedding of learners in the \mathbf{G} -sense to learners in the \mathbf{X} -sense. More formally, for all $\delta \subseteq \mathfrak{P} \times \mathfrak{R}$ and $\mathcal{S} \subseteq \mathcal{R}$,

$$\mathcal{S} \in \mathbf{G}\delta \Leftrightarrow \Theta(\mathcal{S}) \in \mathbf{ImmediateX}\delta. \quad (3.5)$$

The special case of (3.5) with $\delta = \mathbf{M}$ is used in [CJM⁺05].

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 we don't know of any degenerate sequence acceptance criteria implicit in the prior literature. We conjecture that any degenerate sequence acceptance criteria would be useless to model learning. Hence, the present paper will solely focus on non-degenerate such criteria.

It is easy to see that, for non-degenerate δ , we have, for all $\mathcal{C} \subseteq \mathcal{P}$,

$$\mathcal{CX}\delta = \mathcal{CX}\mathcal{R}^2\delta. \quad (3.6)$$

3.3 Abstract Theorems

Propositions 3.3.2-3.3.4 give, respectively, 1. a way of translating learnability from one hypothesis space to another, 2. the relation between function learning from arbitrary text and text learning of sets of functions and 3. the relation between function learning from arbitrary vs. canonical text.

The following definition formalizes properties of sequence acceptance criteria that will allow for such criteria to be translated from using one hypothesis space to another. The succeeding proposition makes use of these properties.

Definition 3.3.1. For all $p \in \mathfrak{P}$ and V, V' numberings for **ce** sets, define

$$\text{Sem}_{V, V'}(p) = \{p' \in \mathfrak{P} \mid \text{dom}(p) \subseteq \text{dom}(p') \wedge \forall i \in \text{dom}(p) : V_{p(i)} = V'_{p'(i)}\}; \quad (3.7)$$

$$\text{Mc}(p) = \{p' \in \mathfrak{P} \mid \begin{array}{l} \text{dom}(p) \subseteq \text{dom}(p') \wedge \forall i \in \text{dom}(p) : [(i+1 \in \text{dom}(p)) \\ \wedge p'(i) \neq p'(i+1)) \Rightarrow p(i) \neq p(i+1)] \end{array}\}. \quad (3.8)$$

Intuitively, $\text{Sem}_{V,V'}(p)$ is the set of all those sequences p' that, for each argument on which p converges, converge to a semantically identical output, just in a possibly different numbering. Furthermore, $\text{Mc}(p)$ is the set of all those sequences p' that make a mind-change only where p makes one (or if p diverges on one of the arguments in question).

A sequence acceptance criterion δ is said to be a *semantic restriction in V* iff, for all $(p, q) \in \delta$ and $p' \in \text{Sem}_{V,V}(p)$, $(p', q) \in \delta$.

A sequence acceptance criterion δ is said to be a *mind-change semantic restriction in V* iff, for all $(p, q) \in \delta$ and $p' \in \text{Sem}_{V,V}(p) \cap \text{Mc}(p)$, $(p', q) \in \delta$.

Note that each semantic restriction is a mind-change semantic restriction.

For each δ , a mind-change semantic restriction in V , we let, for each numbering V' , $\delta_{V'} = \{(p', q) \mid \exists p \in \mathfrak{P} : (p, q) \in \delta \wedge p' \in \text{Sem}_{V,V'}(p) \cap \text{Mc}(p)\}$.

Note that **Pcp** is a semantic restriction in W , and that **Ex** and **Conv** are mind-change semantic restrictions in W .

Proposition 3.3.2. Let α be a target presenter and δ, δ' mind-change semantic restrictions. Further, let V, V' be effective numberings of some **ce** sets.

Suppose $h \in \mathfrak{P}$ be such that there is $f \in \mathfrak{P}$ with $\forall e \in \text{range}(h) : [f(e) \downarrow \wedge V_e = V'_{f(e)}]$. Let $\mathcal{D} \in \{\text{Id}, \text{Prud}, \mathcal{T}\text{Prud}\}$. Then we have

$$(\mathcal{D}\mathcal{T}\delta\alpha\mathbf{G}\delta'_V)(h) \subseteq (\mathcal{D}\mathcal{T}\delta\alpha\mathbf{G}\delta'_{V'})(f \circ h).$$

Proof: Let $L \in (\mathcal{D}\mathcal{T}\delta\alpha\mathbf{G}\delta'_V)(h)$. Then h necessarily fulfills the admissibility restriction $\mathcal{T}\delta_V$, i.e., $h \in \mathcal{T}\delta_V$. First, we show that $f \circ h \in \mathcal{T}\delta_{V'}$.

Let $T \in \text{range}(\alpha)$. Let $(p, q) = \mathbf{G}(h, T)$. Then $(p, q) \in \delta_V$ as $h \in \mathcal{T}\delta_V$. Hence, $\mathbf{G}(f \circ h, T) = (f \circ p, q)$. Therefore, by the supposition about f and since δ a mind-change semantic restriction, $(f \circ p, q) \in \delta_{V'}$. Hence, $f \circ h \in \mathcal{T}\delta_{V'}$.

Next we show that, for all texts T for L , $\mathbf{G}(f \circ h, T) \in \delta'_{V'}$. Let T be a text for L . Let $(p, q) = \mathbf{G}(h, T)$. Thus, $(p, q) \in \delta'_V$. Then $\mathbf{G}(f \circ h, T) = (f \circ p, q)$. Therefore, by

the supposition about f and since δ' a mind-change semantic restriction, $(f \circ p, q) \in \delta'_{V'}$.

Now we can conclude, in the case of $\mathcal{D} = \text{Id}$, $L \in (\mathcal{DT}\delta\alpha\mathbf{G}\delta'_{V'}) (f \circ h)$.

What remains to be shown is, in the cases where $\mathcal{D} \in \{\mathbf{Prud}, \mathcal{TPrud}\}$, (total) prudence of h implies (total) prudence of $f \circ h$, which is similarly easy. \square

The following is a trivial remark about the relationship between **Arb** and **Txt**.

Proposition 3.3.3. Let \mathcal{C} be a learner admissibility restriction, β a sequence generating operator and δ a sequence acceptance criterion.

Let $\mathcal{D} \in \{\text{Id}, \mathbf{Prud}, \mathcal{TPrud}\}$. Then we have

$$\mathcal{DCArb}\beta\delta = \text{Pow}(\mathcal{R}) \cap \mathcal{DCTxt}\beta\delta.$$

Proof: The proof of this proposition is similarly straightforward as that of Proposition 3.3.2. \square

The following is a trivial remark about the relationship between **Canonical** and **Arb**.

Proposition 3.3.4. Let \mathcal{C} be a learner admissibility restriction, β a sequence generating operator and δ a sequence acceptance criterion.

Let $\mathcal{D} \in \{\text{Id}, \mathbf{Prud}, \mathcal{TPrud}\}$. Then we have

$$\mathcal{DCArb}\beta\delta \subseteq \mathcal{DCCanonical}\beta\delta.$$

Proof: The proof of this proposition is similarly straightforward as that of Proposition 3.3.2. \square

Next are the conceptually useful definitions of locking sequence and locking set. The succeeding proposition shows the importance of such sequences/sets.

Definition 3.3.5 ([BB75, JORS99]). A *locking sequence* for a learner h on a language L is a sequence σ such that $\text{content}(\sigma) \subseteq L$ and, for all σ' with $\sigma \subseteq \sigma'$ and $\text{content}(\sigma') \subseteq L$, $h(\sigma') = h(\sigma)$. In the literature, locking sequences have been seen to be very useful.

A *locking set* for a *set-driven* learner h on a language L is a set D such that $D \subseteq L$ and, for all D' with $D \subseteq D' \subseteq L$, $h(D') = h(D)$.

Proposition 3.3.6 (Locking Lemma, [BB75]). Let $h \in \mathcal{P}$. Then, for each $L \in \mathbf{TxtGEx}$ there is a locking sequence for h on L .

Furthermore, for each $L \in \mathbf{TxtSdEx}$, there is a locking set for h on L .

Chapter 4

DELAYED POSTDICTIVE COMPLETENESS AND CONSISTENCY

Akama and Zeugmann [AZ08] presented success criteria that are a little less restrictive than postdictively complete Ex-learning. Their criteria delay the requirement to postdict a given datum by a fixed natural number δ of (not necessarily distinct) hypotheses output. For ordinary postdictive completeness, if a learner h has seen so far, on a computable g , input, $g(0), \dots, g(n-1)$, then h 's corresponding hypothesis, p_n , must correctly compute $g(0), \dots, g(n-1)$.¹ For delay δ , Akama and Zeugmann, require only that, on $g(0), \dots, g(n-1)$, h 's *later* hypothesis, $p_{n+\delta}$, must correctly compute $g(0), \dots, g(n-1)$. A delay δ learner could, for example, after seeing $g(0), \dots, g(n-1)$, run a counter down from δ to 0 to see which future hypothesis must correctly compute $g(0), \dots, g(n-1)$.

In the present chapter, we extend this notion of delayed postdictive completeness from *constant* delays δ to *dynamically computed* delays. *One* of the ways we consider herein to do this involves counting down from notations for constructive ordinals. We explain. Everyone knows how to use the natural numbers for counting, including for counting *down*. Freivalds and Smith [FS93], as well as [ACJS04], employed in learning theory *notations for constructive ordinals* [Rog67, § 11.7] as devices for algorithmic counting down.

¹ Note that, for $n = 0$, the data seen is empty and the output hypothesis, p_0 , is unconstrained.

Intuitively *ordinals* are representations of well-orderings. 0 represents the empty ordering, 1 represents the ordering of 0 by itself, 2 the ordering $0 < 1$, 3 the ordering $0 < 1 < 2$, \dots . The ordinal ω represents the standard ordering of all of \mathbb{N} . $\omega + 1$ represents, for example, the ordering of \mathbb{N} consisting of the positive integers in standard order *followed by* 0. The *successor ordinals* are those of the form $\alpha + 1$ which have a single element laid out after a copy of another ordinal α . $\omega + \omega$ can be thought of as two copies of ω laid end to end — much bigger than ω . $\omega \cdot 3$ represents three copies of ω laid end to end. By contrast, $3 \cdot \omega$ represents ω copies of 3 — which is just ω . We see, then, for ordinals, $+$, \cdot are not commutative. $\omega \cdot \omega$ is ω copies of ω laid out end to end. We can iterate this and define exponentiation for ordinals. *Limit ordinals* are those, like ω , $\omega + \omega$, $\omega \cdot \omega$, and ω^ω , which are not 0 and are not successor ordinals. All of them are infinite. Importantly, the *constructive ordinals* are just those that have a program (called a *notation*) in some system which specifies how to build them (lay them out end to end, so to speak).² Informally, here, for example, is how to think of counting down from such a notation for $\omega + \omega$. One first computes some estimate for a natural number to count down from and begins counting down from it; then, later, one can revise *once* this estimate and subsequently count down some more from that. An example of a *countdown* from $\omega + \omega$ looks like this.

$$\omega + 42, \omega + 41, \dots, \omega + 1, \omega, 23, 22, \dots, 1, 0.$$

In this example, the first estimate for a natural number to count down from is 42, the revision occurs when counting down from ω to 23. For counting down from a notation for $\omega + \omega + \omega = \omega \cdot 3$, one can revise the initial estimate *twice*. Since ordinal notations represent well-orders, they do *not* permit infinitely long

² Technically, we count down from *notations* for constructive ordinals (instead of from the ordinals themselves) simply because notations, being finite (programs), in principle, fit inside computers; whereas, at least infinite ordinals do not.

countdowns, neither algorithmic (we do finite, algorithmic countdowns) nor non-algorithmic. Note that, for counting down from ω or higher ordinals, there exist arbitrarily long (but, of course, finite) countdowns.

[SSV04] gives a further generalized notion of counting down. They consider certain partial orders with no *computable* infinitely descending chains. In the present chapter we consider arbitrary and also computable (directed) graphs with no infinite, computable (directed) paths, and we *algorithmically* count “down” along their paths. Theorem 4.5.4 gives a nice example of linearly ordered, computable such graph which nonetheless has infinite paths (just not computable ones). We call our graphs in the present chapter, *countdown graphs*.

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

In the formalizations of each single countdown below, for a given graph, our counters start *inside* the graph, e.g., for an ordinal α , it starts at a notation for an ordinal $< \alpha$. N.B. This is compensated for by how we stop, i.e., how we bottom out. We do not bottom out when, e.g., we run out of some place to count down to; instead, we bottom out by going off the track, i.e., by counting up, sideways, in place or off the whole graph.

Section 4.1 works out the details about a particular kind of countdown graphs, namely those based on systems of ordinal notations. Further, it introduces *feasible* systems of ordinal notations, graphs that have an order isomorphic to a constructive ordinal and allow for many useful operations on their elements to be carried out in polynomial time. This allows for our strong complexity theoretic results in

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

Section 4.3.

Section 4.2 formalizes the use of countdown graphs for delaying postdictive completeness and consistency.

Sections 4.3 through 4.6 present our results on delayed postdictive completeness and consistency.

All of our results in Section 4.3 provide information about polynomial time learners. From Theorem 4.3.1 and its proof we see that, for polynomial time learning, postdictive completeness (and delayed variants) allows *some* but *not* all postponement tricks, while from Theorem 4.3.2 we have that there is a surprisingly tight boundary, for polynomial time learning, between what postponement is allowed and what is not. For *example*: 1. the set of polynomial time computable functions *is* polynomial time postdictively completely Ex-learnable (by a complexity-bounded enumeration technique) employing some postponement, *but* 2. the set of exponential time computable functions, while polynomial time Ex-learnable with a little more postponement, is *not* polynomial time postdictively completely Ex-learnable! From Theorem 4.3.1, we see that, for w a notation for ω , the set of exponential time functions *is* polynomial time Ex-learnable with w -*delayed* postdictive completeness. Theorems 4.3.1 and 4.3.2 also provide generalizations to further, small constructive limit ordinals.

Therefore, Theorems 4.3.1 and 4.3.2 provide strong justification for studying the herein ordinal countdown variants of Postdictive Completeness.

In the prior literature we also see further variants of postdictive completeness and consistency not based on delay. For example, [CJSW04] surveys with references variants where the learner is further restricted to be total computable, or even additionally has to observe the restriction of postdictive completeness or consistency on all input (not only on input taken from a target).

Another variant, which we call *local postdictive completeness*, is given in

[JLZ06] and [AZ08], where it is called *local consistency* and *coherence*, respectively. In this variant, the learner is only required to postdict the most recent datum, and need not postdict the previous data items. Interestingly, [JLZ06] shows that this local version of postdictive completeness is *not* equivalent to unmodified postdictive completeness in a setting of iterative learning, while [AZ08] shows that these two notions *are* equivalent in a non-iterative setting, even for finitely delayed versions thereof. We extend this latter result to all our delayed postdictive completeness criteria, as well as to that of our delayed versions of postdictive consistency (Theorem 4.4.1).

Section 4.4 shows how the different variants of our criteria relate in learning power (we already mentioned Theorem 4.4.1 about local vs. global restrictions just above). Our main theorem in this section is Theorem 4.4.2. For *example*, it entails that there is a set of computable functions which is postdictively *consistently* learnable (with no delays) by a transductive, linear time learner *but is not* postdictively *completely* learnable with delays employing *any* countdown graph.

Furthermore, some of our results in Section 4.4 entail learnability with linear time learners. For these latter results we get by with an extremely *fair*⁴, restricted kind of *linear-time* learner, we call *transductive*. A transductive learner has access only to its current datum.

In Section 4.5, our main result, Theorem 4.5.6 and its Corollaries 4.5.7 and 4.5.8 *completely characterize* learning power *in dependence on* associated (computable) *countdown graphs*. Another corollary to Theorem 4.5.6, Corollary 4.5.9, extends the finite hierarchy given in [AZ08] into the constructive transfinite.

Finally, Section 4.6 gives our interesting characterization results in Theorem 4.6.1. Each of these characterizations are of a form of reliable identification⁵

⁴ See Section 1.

⁵ See Definition 3.2.5.

and they are in terms of postdictively complete learning with very general (even non-computable) countdown graphs. These answers a question of Thomas Zeugmann [Zeu08] and give further justification for the use of general countdown graphs.

For the remainder of this chapter, we will exclusively use φ as our hypothesis space; thus, we will omit any further mention thereof.

4.1 Feasible Systems of Ordinal Notation

In this section we first introduce *systems of ordinal notations* (Definition 4.1.1). Then we specialize this notion by adding feasibility constraints (Definition 4.1.3). The work in this section on feasible systems of ordinal notations was previously published as part of [CKP07]. Most of this section is based directly or indirectly on [Rog67, §11.7/8].

Note that [MV05] independently gives an *example* feasible system of ordinal notations for the ordinals up to ε_0 ,⁶ developed for practical purposes in the area of automated termination proofs for the ACL2 theorem-proving system.

Definition 4.1.1. (System of Ordinal Notations)

For an ordinal α , a *system of ordinal notations* for all and only the ordinals $< \alpha$ is a pair (S, ν_S) , where $S \subseteq \mathbb{N}$ and ν_S maps S onto the set of all ordinals $< \alpha$.⁷ For each $u \in S$, u is called a *notation for* $\nu_S(u)$. Additionally we require:

There is a partial computable function f such that, for all $u \in S$,

- (a) u is a notation for 0 $\Rightarrow f(u) = 0$,
- (b) u is a notation for a successor ordinal $\Rightarrow f(u) = 1$ and

⁶ ε_0 is the constructive ordinal which is the least upper bound of the sequence ω , ω^ω , ω^{ω^ω} , \dots

⁷ We will sometime ambiguously refer to (S, ν_S) as S .

(c) u is a notation for a limit ordinal $\Rightarrow f(u) = 2$.

And:

(d) There is a partial computable function pred_S such that for each $u \in S$ which is a notation for a successor ordinal $\beta + 1$, $\text{pred}_S(u)$ is a notation for β in S .

(e) There is a partial computable function

$$\text{lim}_S : \mathbb{N} \times \{0\}^* \rightarrow \mathbb{N}$$

such that, for all limit-ordinals $\lambda < \alpha$ and notations l in S for λ , we have that $(\nu_S(\text{lim}_S(l, 0^i)))_{i < \omega}$ is a strictly increasing sequence of ordinals with limit (or least upper bound) λ .

The following is used for our definition of a feasible system of ordinal notations.

Definition 4.1.2. For a system of ordinal notations S , a pair (l_S, n_S) of partial computable functions $\mathbb{N} \rightarrow \mathbb{N}$ is a *decompose pair for S* iff, for all notations $u \in S$ for an ordinal α , $l_S(u)$ denotes a notation for the biggest (limit ordinal or 0) $\lambda \leq \alpha$, and $n_S(u)$ is such that $\alpha = \lambda + n_S(u)$.

Definition 4.1.3 below gives a list of requirements that we consider useful in dealing with systems of ordinal notations in a setting of feasibility. This list starts with the requirements for a system of ordinal notations as given in Definition 4.1.1 above, and, then, extending these requirements. Some of these extensions (such as feasibility of addition and multiplication on notations) are inspired by what is feasibly possible for natural numbers. Other extensions are requirements that hold easily in example systems and are included for technical usefulness.

Definition 4.1.3. (Feasible System of Ordinal Notations)

A system of ordinal notation (S, ν_S) for the ordinals $< \alpha$, where α is infinite and the

set of all ordinals $< \alpha$ is additively and multiplicatively closed, is called *feasible* iff: There is a polynomial time computable function f such that, for all $u \in S$,

- (a) u is a notation for 0 $\Rightarrow f(u) = 0$,
- (b) u is a notation for a successor ordinal $\Rightarrow f(u) = 1$ and
- (c) u is a notation for a limit ordinal $\Rightarrow f(u) = 2$.

And:

- (d) There is a polynomial time computable function pred_S such that for all u notations for a successor ordinal $\beta + 1$, $\text{pred}_S(u)$ is a notation for β in S .
- (e) There is a polynomial time computable $\text{lim}_S : S \times \{0\}^* \rightarrow S$ such that, for all limit-ordinals $\lambda < \alpha$ and notations l in S for λ , we have that $(\nu_S(\text{lim}_S(l, 0^i)))_{i < \omega}$ is a strictly increasing sequence of ordinals with limit λ .

Additionally we require

- (f) there is a polynomial time computable function $+_S$ such that, $\forall u, v \in S : \nu_S(u +_S v) = \nu_S(u) + \nu_S(v)$,
- (g) there is a polynomial time computable function \cdot_S such that, $\forall u, v \in S : \nu_S(u \cdot_S v) = \nu_S(u) \cdot \nu_S(v)$,
- (h) there is a polynomial time computable function \ulcorner_S , such that, $\forall n \in \mathbb{N}, \nu_S(\ulcorner_S n) = n$ and
- (i) there is a decompose pair (l_S, n_S) for S such that l_S, n_S are computable in polynomial time.

Definition 4.1.4. Following Rogers [Rog67], we say that a system of ordinal notations S is *univalent* iff ν_S is 1-1. We define the relation \leq_S on natural numbers such

that: $u \leq_S v \Leftrightarrow [u, v \in S \wedge \nu_S(u) \leq \nu_S(v)]$. Also following Rogers, we say a system of ordinal notations S is *computably related* iff \leq_S is computably decidable, and *computably decidable* iff the set of notations S is computably decidable. Analogously, we define a system S to be *polynomial time related* iff \leq_S is polynomial time decidable, and *polynomial time decidable* iff the set S is polynomial time decidable.

The following proposition simplifies dealing with feasible systems of ordinal notations.

Proposition 4.1.5. The following observations

- In Definition 4.1.3 above we have that feasible relatedness, together with (f), (h) and (i) implies (a)-(d).
- Every polynomial time related feasible system of ordinal notations S is polynomial time decidable, as we have: $u \in S \Leftrightarrow u \leq_S u$.
- Every polynomial time related feasible system of ordinal notations is a computably related system of ordinal notations.⁸
- For a univalent *or* polynomial time related feasible system of ordinal notations S , it is polynomial time decidable whether two notations in S are notations for the same ordinal.⁹

Next we state with three lemmas and a theorem that any computably related system of ordinal notations for all ordinals $< \alpha$ can be turned into a feasibly related feasible system of ordinal notations, giving a notation to at least all ordinals $< \alpha$. The proofs of the two first lemmas rely heavily on postponement tricks.

⁸ Therefore, all theorems for computably related systems of ordinal notations hold. For example, there cannot be a polynomial time related feasible system of ordinal notations for *all* constructive ordinals (see [Rog67, Corollary XV]).

⁹ For univalent systems there are of course no two different notations in S for the same ordinal. For a feasibly related systems of ordinal notations, $u, v \in \mathbb{N}$ are notations in S for the same ordinal iff $[u \leq_S v \text{ and } v \leq_S u]$.

Lemma 4.1.6. Suppose S is a system of ordinal notations such that there is a polynomial time computable function s_S such that, for each $u \in S$, $s_S(u)$ is a notation in S for the successor ordinal of $\nu_S(u)$. Let $\lim_S : \mathbb{N} \times \{0\}^* \rightarrow \mathbb{N}$ be a computable function satisfying (e) from Definition 4.1.1. Then there is a *polynomial time* computable function $\lim'_S : \mathbb{N} \times \{0\}^* \rightarrow \mathbb{N}$ satisfying (e) from Definition 4.1.3.

Proof: Define \lim'_S thus. On input $(u, 0^i)$, run \lim_S on inputs $(u, 0^j)$ for all $j \leq i$, each for up to i steps. If none converges, output \underline{i} — a notation in S for i . Otherwise, for some $j \leq i$, $\lim_S(u, 0^j)$ converges. In this case, for the maximal such j , compute the i -times successor of $\lim_S(u, 0^j)$ and output the result — a notation for $\nu_S(\lim_S(u, 0^j)) + i$. Importantly, thanks to [RC94, Corollary 3.7], the algorithm just provided for \lim'_S is feasible.

If u is a notation in S for a limit ordinal, for each i , the corresponding output as found by the algorithm on $(u, 0^i)$ is a notation in S for an ordinal bigger or equal to the ordinal of the output found for smaller i ; therefore, the sequence of ordinals corresponding to the outputs for successively larger i is strictly monotonic increasing. Also, when u is a notation in S for a limit ordinal, the limit of $(\nu_S(\lim'_S(u, 0^i)))_{i \in \mathbb{N}}$ is the same as the limit of $(\nu_S(\lim_S(u, 0^i)))_{i \in \mathbb{N}}$.

□

Lemma 4.1.7. Suppose S is a computably related system of ordinal notations for all and only the ordinals $< \alpha$ for some ordinal α . Then there is a *polynomial time* related system S' of ordinal notations for all and only the ordinals $< \alpha$.

Proof: Define S' thus. Let e be the numerical name for a program deciding \leq_S . Define $t : \mathbb{N} \rightarrow \mathbb{N}, u \mapsto \max(\{\Phi_e(i, j) \mid i, j \leq u\})$. Let S' be the system of notations where for all β given a notation u in S , we have

that $\langle 0^{t(u)}, 0^u \rangle$ is a notation for β .¹⁰ Obviously, $\forall m, n, u, v \in \mathbb{N} : \langle 0^m, 0^u \rangle \leq_{S'} \langle 0^n, 0^v \rangle \Leftrightarrow [\varphi_e(u, v) = 1 \text{ in } \leq \max\{n, m\} \text{ steps and } m = t(u) \text{ and } n = t(v)]$. It follows from [RC94, Lemma 3.2(f) and Corollary 3.7] that $\lambda 0^m, 0^u . t(u) = m$ is polynomial time decidable. Therefore, on the resulting notations we have that order is feasibly decidable. \square

Lemma 4.1.8. Suppose S is a feasibly related system of ordinal notations giving a notation to all and only the ordinals $< \alpha$ for some ordinal α . Then there is a feasibly related system of ordinal notations S' fulfilling (a)-(f) and (h)-(i) as in Definition 4.1.3, giving a notation at least to all ordinals $< \alpha$. In fact, S' gives a notation to all and only the ordinals $< \omega^\alpha$. If S is univalent, so is S' .

Proof: Assume without loss of generality that 0 is the only notation for 0 in S . Let $\langle \rangle$ be a notation in S' for 0. By the Cantor Normal Form theorem, each ordinal γ , $0 < \gamma < \omega^\alpha$ has exactly one representation such that $\gamma = \sum_{i=k}^0 \omega^{\delta_i} \times n_i$, where $\alpha > \delta_k > \dots > \delta_0 \geq 0$ and $n_k, \dots, n_0 \in \mathbb{N} \setminus \{0\}$ (see [Sie65, Theorem 2, Chapter XIV.19, page 323]). Define a system S' by all and only the following assignments of notations. For each γ with $0 < \gamma < \omega^\alpha$, the representation as above and d_k, \dots, d_0 notations in S for $\delta_k, \dots, \delta_0$, respectively, let

$$\langle d_k, n_k, \dots, d_0, n_0 \rangle \text{ be a notation in } S' \text{ for } \gamma.$$

For S' , the requirements (a)-(d) of Definition 4.1.3 are clear. It is similarly clear that in S' , the successor of each notation is polynomial time computable from that notation.

To show (e) for S' : Consider, for S' , a modification of (e) as in Definition 4.1.3 in which “partial computable” replaces “polynomial time computable”. Below we

¹⁰ In the notations $\lambda 0^m, 0^n . f(m, n)$ or $\lambda 0^m, 0^n . f(0^m, 0^n)$, the arguments are of the form a *pair* of strings of zeros, the first string of length m , the second of length n .

show that, for S' , this *modified* version of (e) holds. We already noted above that, for S' , one can polynomial time compute successors of notations. Hence, we can apply Lemma 4.1.6 to S' to obtain, for S' , a polynomial time computable $\lim_{S'}$ satisfying the *unmodified* version of (e). Here, then, is how to compute the partial computable function for the *modified* version of (e). Suppose the input is $(u, 0^i)$, where $u = \langle d_k, n_k, \dots, d_0, n_0 \rangle$ is a notation in S' for a limit ordinal; therefore, d_0 is not a notation for 0 (we don't care what happens if u is otherwise). We output as follows.

If d_0 is a notation for a limit ordinal, $n_0 \neq 1$, output $\langle d_k, n_k, \dots, d_1, n_1, d_0, n_0 - 1, \lim_S(d_0, 0^i), 1 \rangle$.

If d_0 is a notation for a limit ordinal, $n_0 = 1$, output $\langle d_k, n_k, \dots, d_1, n_1, \lim_S(d_0, 0^i), 1 \rangle$.

Otherwise let b be a notation for the predecessor of what d_0 is a notation for.

If $n_0 \neq 1$, output $\langle d_k, n_k, \dots, d_1, n_1, d_0, n_0 - 1, b, i \rangle$.

If $n_0 = 1$, output $\langle d_k, n_k, \dots, d_1, n_1, b, i \rangle$.

To show (f): We define $\langle d_k, n_k, \dots, d_0, n_0 \rangle +_{S'} \langle d'_{k'}, n'_{k'}, \dots, d'_0, n'_0 \rangle$ thus.

If there is i such that d_i and $d'_{k'}$ denote the same ordinal (this is polynomial time decidable by Remark 4.1.5), output $\langle d_k, n_k, \dots, d_{i+1}, n_{i+1}, d_i, n_i + n'_{k'}, d'_{k'-1} n_{k'-1}, \dots, d'_0, n'_0 \rangle$.

Otherwise, let i be minimal such that $d_i >_S d'_{k'}$ (if such an i does not exist, output is trivial). Output $\langle d_k, n_k, \dots, d_{i+1}, n_{i+1}, d_i, n_i, d'_{k'}, n'_{k'}, \dots, d'_0, n'_0 \rangle$.

To show (h): A notation in S' for 0 is $\langle \rangle$, for $n > 0$ a notation in S' is $\langle 0, n \rangle$.

To show (i): $l_{S'}$ and $n_{S'}$ are trivial.

To show polynomial time relatedness: First check whether all coefficients are not 0 and that the sequence of exponents is strictly decreasing with respect to \leq_S . Then compare component-wise exponents and coefficients. As long as both are equal, proceed. If one ordinal notation ends, the ending notation is the notation for the smaller ordinal. If the exponents are not equal, the notation having the smaller

exponent is the notation for the smaller ordinal. If the coefficients are not equal, the notation with the smaller coefficient is the notation for the smaller ordinal.

Runtime: k times the runtime of the runtime for comparison in S plus runtime linear in the length of the input to compare the coefficients.

If S is univalent, the uniqueness of the Cantor Normal Form guarantees univalence of S' .

□

Theorem 4.1.9. Suppose S is a feasibly related system of ordinal notations giving a notation to all and only the ordinals $< \alpha$. Then there is a feasibly related *feasible* system of ordinal notations S' giving a notation at least to all ordinals $< \alpha$. In fact, S' gives a notation to all and only the ordinals $< \omega^{\omega^\alpha}$. If S is univalent, so is S' .

Proofsketch. Apply the construction of the proof of Lemma 4.1.8 *twice* to S . The resulting system will also allow for feasible multiplication (details for this can easily be generalized from [MV05]).

□

Corollary 4.1.10. Let α be a constructive ordinal. Then there is a univalent, feasibly related feasible system of ordinal notations giving a notation to α .

Proof: By [Rog67, Theorem 11.XIX], there is a univalent, computably related system of ordinal notations giving a notation to α . The result follows now from first applying Lemma 4.1.7 and then Theorem 4.1.9.

□

4.2 Countdown Graphs and Countdown Functions

This section formalizes the intuitive discussions from the introduction to this chapter with respect to countdown graphs and our use of them.

Definition 4.2.1. A *graph* is a pair (G, \rightarrow) , where $G \subseteq \mathbb{N}$ and \rightarrow is a binary relation on G . We will use infix notation for \rightarrow . For each graph (G, \rightarrow) , we say that τ is a *G-path* iff: $\#\text{elets}(\tau) > 0$, $\forall i < \#\text{elets}(\tau) : (\tau(i) \in G)$ and $\forall i < \#\text{elets}(\tau) - 1 : (\tau(i) \rightarrow \tau(i+1))$. For each graph G , let \vec{G} denote the set of all G -paths.

For all $m, n \in \mathbb{N}$, we write $m \rightarrow^* n$ (respectively, $m \rightarrow^+ n$) iff there is a G -path τ such that $\tau(0) = m$, $\text{last}(\tau) = n$ (respectively, additionally $\#\text{elets}(\tau) > 1$). We sometimes write G for (G, \rightarrow) . A graph (G, \rightarrow) is said to be *computable* iff both G and \rightarrow are computable. Note that a graph G is computable iff \vec{G} is computable. For a graph (G, \rightarrow) we sometimes identify $m \in G$ with $\{n \in G \mid m \rightarrow^+ n\}$. With every pre-order (A, \leq_A) ¹¹ we associate the graph $(A, >_A)$, where, for all $a, b \in A$, $a >_A b$ iff $(b \leq_A a$ and $a \not\leq_A b)$.

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

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

In Theorem 4.5.4 below we give an interesting example of a countdown graph not based on a system of ordinal notations.

Soon we define what postdictive completeness, respectively consistency, with respect to $G \in \mathcal{G}$ means. Intuitively, every learner is composed of two functions: one for giving hypotheses, and one for the countdown. For the purpose of this

¹¹ A *pre-order* is a pair (A, \leq_A) such that \leq_A is a transitive and reflexive relation on A .

entries) is the n -th countdown of σ_0 . As we will see below, for an associated learner g , the n -th row will be relevant to $g(n)$.

For each $\sigma \in \mathbb{M}$ and $n < \#\text{elets}(\sigma) - 1$ we define

$$\text{row}(n, \sigma) = \langle \sigma(n+1)(n), \dots, \sigma(\#\text{elets}(\sigma) - 1)(n) \rangle_{\text{Seq}}. \quad (4.3)$$

For σ_0 as presented above in (4.2), we have, for example, $\text{row}(4, \sigma_0) = \langle 2, 1, 0 \rangle_{\text{Seq}}$. Each $\text{row}(n, \sigma)$ is a countdown.

We will consider a given countdown sequence τ as *terminated* with respect to a given countdown graph $G \in \mathcal{G}$, iff $\tau \notin \vec{G}$. We then say that “ τ has terminated” or “ τ has bottomed out”. For a given multicountdown sequence we will define, just below, the set of all n such that the n -th countdown has (started and) bottomed out. For all σ and all $G \in \mathcal{G}$, define

$$\perp_G(\sigma) = \{n < \#\text{elets}(\sigma) - 1 \mid \sigma \notin \mathbb{M} \vee \text{row}(n, \sigma) \notin \vec{G}\}; \quad (4.4)$$

$$\perp\!\!\!\perp_G(\sigma) = \perp_G(\sigma) \setminus \perp_G(\sigma^-). \quad (4.5)$$

We omit the subscript G whenever no confusion can arise. Note that $\perp(\emptyset) = \emptyset = \perp\!\!\!\perp(\emptyset)$.

We pronounce \perp as “bottom” and $\perp\!\!\!\perp$ as “recent bottom”. For $\sigma \in \mathbb{M}$, $\perp(\sigma)$ is the set of all countdown numbers where the countdown has terminated, while $\perp\!\!\!\perp(\sigma)$ is the set of all countdown numbers that have just terminated.

Let us, for example, consider the finite countdown graph G on $\{0, 1, 2, 3\}$ with the natural $>$ -order on \mathbb{N} . For σ_0 depicted above in (4.2), we have $\perp_G(\sigma_0) = \{0, 1, 2, 3, 6\}$ and $\perp\!\!\!\perp_G(\sigma_0) = \{6\}$. The example of rows $n = 4$ and $n = 5$ shows that reaching a minimal element (in this case 0) of G does not imply immediate termination of the countdown. The example of rows $n = 2$ and $n = 3$ shows how countdowns terminate when not obeying the graph relation. Note that the countdown for row $n = 6$ has terminated immediately when it started, as it started with 5, and $\langle 5 \rangle_{\text{Seq}}$ is not a G -path.

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

Intuitively, in the definition just below, p and d , respectively, represent a sequence of hypotheses and a sequence of corresponding multicounts, and q a target function.

Definition 4.2.2. Let, for all $G \in \mathcal{G}$ and $p, d, q \in \mathcal{R}$,

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

For all $g \in \mathcal{R}$ and $h, f \in \mathcal{P}$, we say that $\langle h, f \rangle$ works *postdictively completely* (respectively, *consistently*) on g with G -delay iff $\mathbf{G}(\langle h, f \rangle, g) \in \mathbf{Pcp}_G$ (respectively, \mathbf{Pcs}_G).¹³

We omit “with G -delay”, if no confusion can arise.

Further, we define local variants of the two above families of admissibility restrictions, *local postdictive completeness* and *local postdictive consistency* thus.

- $\mathbf{Pcpl}_G(\langle p, d \rangle, q) :\Leftrightarrow \forall x \forall n \in \perp\!\!\!\perp_G(d[x]) : q(n) \in \varphi_{p(x)}$,
- $\mathbf{Pcsl}_G(\langle p, d \rangle, q) :\Leftrightarrow \forall x \forall n \in \perp\!\!\!\perp_G(d[x]) : \varphi_{p(x)}(n) \downarrow \Rightarrow q(n) \in \varphi_{p(x)}$.

To accommodate for the countdown-output of a learner, we redefine \mathbf{Ex} slightly *for the remainder of this chapter* as follows.

$$\mathbf{Ex} = \{(\langle p, d \rangle, q) \in \mathfrak{P}^2 \mid \exists e : p \rightarrow e \wedge \varphi_{p'} = \text{content}(q)\}. \quad (4.6)$$

For notational purposes, we define the following variants of row, \perp and $\perp\!\!\!\perp$.

Intuitively, in the following definition, σ plays the role of data sequence, f that of countdown function, n that of row number and $\lambda i \leq \#\text{elets}(\sigma).f(\sigma[i])$ that of associated multicountdown sequence.

¹³ G is a graph, while \mathbf{G} stands for “Gold” and is defined in Example 3.2.3.

Definition 4.2.3. Let, for all $G \in \mathcal{G}$, $\sigma \in \text{Seq}$, $f \in \mathcal{P}$ and $n \leq \#\text{elets}(\sigma)$,

$$\text{row}(n, f, \sigma) = \text{row}(n, \lambda i \leq \#\text{elets}(\sigma).f(\sigma[i])), \quad (4.7)$$

$$\perp_G(f, \sigma) = \perp_G(\lambda i \leq \#\text{elets}(\sigma).f(\sigma[i])), \quad (4.8)$$

$$\perp\!\!\!\perp_G(f, \sigma) = \perp_G(f, \sigma) \setminus \perp(f, \sigma^-). \quad (4.9)$$

We omit the subscript G whenever no confusion can arise.

Note that, for all $f \in \mathcal{P}$, all σ and all $n < \#\text{elets}(\sigma)$,

$$\text{row}(n, f, \sigma) = \lambda i \leq (\#\text{elets}(\sigma) - n - 1).f(\sigma[i + n + 1])(n). \quad (4.10)$$

Also we have, for all $G \in \mathcal{G}$ and all $f \in \mathcal{P}$ such that $\forall \tau : (\lambda i \leq \#\text{elets}(\tau).f(\tau[i])) \in \mathbb{M}$, for all $\sigma \in \text{Seq}$ and all $n < \#\text{elets}(\sigma)$,

$$n \in \perp_G(f, \sigma) \Leftrightarrow \text{row}(n, f, \sigma) \notin \vec{G}. \quad (4.11)$$

The following proposition is useful in several later proofs.

Proposition 4.2.4. Obviously, we have for all $G \in \mathcal{G}$ and $\mathcal{S} \subseteq \mathcal{R}$ such that $[\mathcal{S}] = \text{Seq}$,¹⁴

$$\mathcal{S} \in \mathcal{RGPcp}_G \mathbf{Ex} \cup \mathbf{GPcp}_G \mathbf{Ex} \Rightarrow \mathcal{S} \in \mathcal{TPcp}_G \mathbf{GEx}; \quad (4.12)$$

$$\mathcal{S} \in \mathcal{RGPcpl}_G \mathbf{Ex} \cup \mathbf{GPcpl}_G \mathbf{Ex} \Rightarrow \mathcal{S} \in \mathcal{TPcpl}_G \mathbf{GEx}; \quad (4.13)$$

$$\mathcal{S} \in \mathcal{RGPcs}_G \mathbf{Ex} \cup \mathbf{GPcs}_G \mathbf{Ex} \Rightarrow \mathcal{S} \in \mathcal{TPcs}_G \mathbf{GEx}; \quad (4.14)$$

$$\mathcal{S} \in \mathcal{RGPcsl}_G \mathbf{Ex} \cup \mathbf{GPCsl}_G \mathbf{Ex} \Rightarrow \mathcal{S} \in \mathcal{TPcsl}_G \mathbf{GEx}. \quad (4.15)$$

The just below lemma encapsulates many diagonalizations for the proofs in the present chapter.

Lemma 4.2.5. Let $\mathcal{C} \subseteq \mathcal{P}$ and $\delta \subseteq \mathbf{Ex}$. Let $\mathcal{S} \subseteq \mathcal{R}$ such that $(\forall \langle h, f \rangle \in \mathcal{C} \mid \langle h, f \rangle \mathbf{G}\delta\text{-learns } \mathcal{S}) \exists t_0, t_1 \in \mathcal{P} \forall e \in \mathbb{N} : \exists \mathcal{S}_e \subseteq \mathcal{R}$ such that (i) and (ii) just below hold.

¹⁴ Recall that, for $\mathcal{S} \subseteq \mathcal{R}$, $[\mathcal{S}] = \{f[i] \mid f \in \mathcal{S}, i \in \mathbb{N}\}$ (see Section 2.1).

(i) $([\varphi_e] \subseteq [\mathcal{S}_e] \wedge \varphi_e \in \mathcal{R}) \Rightarrow \varphi_e \in \mathcal{S}$; and

(ii) For all $\sigma \in [\mathcal{S}_e]$, (iia)-(iie) just below hold.

(a) $\varphi_e = \sigma \Rightarrow (t_0(e, \sigma) \downarrow \wedge t_1(e, \sigma) \downarrow \wedge h(\sigma \diamond t_0(e, \sigma)) \downarrow \wedge h(\sigma) \downarrow)$

(b) $(t_0(e, \sigma) \downarrow \wedge t_1(e, \sigma) \downarrow) \Rightarrow$

$(t_0(e, \sigma), t_1(e, \sigma) \in \text{Seq} \wedge \#elets(t_0(e, \sigma)), \#elets(t_1(e, \sigma)) > 0).$

(c) $\sigma \subseteq \varphi_e \wedge h(\sigma \diamond t_0(e, \sigma)) \downarrow \neq h(\sigma) \downarrow \Rightarrow \sigma \diamond t_0(e, \sigma) \in [\mathcal{S}_e].$

(d) $\sigma \subseteq \varphi_e \wedge h(\sigma \diamond t_0(e, \sigma)) \downarrow = h(\sigma) \downarrow \wedge t_1(e, \sigma) \downarrow \Rightarrow \sigma \diamond t_1(e, \sigma) \in [\mathcal{S}_e].$

(e) $(t_0(e, \sigma) \downarrow \wedge t_1(e, \sigma) \downarrow \wedge h(\sigma \diamond t_0(e, \sigma)) \downarrow = h(\sigma) \downarrow = h(\sigma \diamond t_1(e, \sigma))) \Rightarrow$

$\varphi_{h(\sigma \diamond t_1(e, \sigma))} \notin \mathcal{R}.$

Then $\mathcal{S} \notin \mathcal{CG}\delta$.

Proof: Suppose, by way of contradiction otherwise. Suppose $h \in \mathcal{C}$ witnesses $\mathcal{S} \in \mathcal{CG}\delta$. For all $j \in \{0, 1\}$, let t_j be as found by (ii).

Define, with **KRT** (Theorem 2.3.3), $g = \varphi_e$ so that g works according to the following informal definition in stages. For each s , g_s denotes the finite initial segment of g as defined just before the beginning of stage s .

```

1  $g_0 \leftarrow \emptyset;$ 
2 for  $s = 0$  to  $\infty$  do
3    $\tau_0 \leftarrow t_0(e, g_s);$ 
4    $\tau_1 \leftarrow t_1(e, g_s);$ 
5   if  $h(g_s \diamond \tau_0) \neq h(g_s)$  then
6      $g_{s+1} \leftarrow g_s \diamond \tau_0;$ 
7   else
8      $g_{s+1} \leftarrow g_s \diamond \tau_1;$ 

```

For $s \in \mathbb{N}$ and $j \in \{0, 1\}$ we define

$$\tau_j^s = t_j(e, g_s). \quad (4.16)$$

Claim 1: We have (4.17) and (4.18) just below.

$$\forall s \in \mathbb{N} : (g_s \text{ is defined} \wedge g_s \in [\mathcal{S}_e]). \quad (4.17)$$

$$\forall s \in \mathbb{N} : (\tau_0^s \downarrow \wedge \tau_1^s \downarrow). \quad (4.18)$$

We show the claim by induction on s with trivial base case. Let now $s \in \mathbb{N}$ such that the claim holds for s . By (iia) we have $\tau_0^{s+1} \downarrow$ and $\tau_1^{s+1} \downarrow$, as well as $h(g_s \diamond \tau_0^s) \neq h(g_s)$ is a computable predicate. We use (iic) and (iid) to see that g_{s+1} is defined and $g_{s+1} \in [\mathcal{S}_e]$. □ (FOR CLAIM 1)

By Claim 1, all stages will be reached. Furthermore, by (iib), for all s , $\#\text{elets}(\tau_0^s), \#\text{elets}(\tau_1^s) > 0$; hence, $g(i)$ will be defined no later than after stage i . Thus,

$$g \in \mathcal{R}. \quad (4.19)$$

By (4.19), (i) and Claim 1, we now have $g \in \mathcal{S}$.

Claim 2: h does not converge on g .

We show that, for any stage s , there exist a $y \geq \#\text{elets}(g_s)$ such that $h(g[y+1]) \neq h(g[y])$. Let s be any stage.

Case 1: $h(g_s \diamond \tau_0^s) \neq h(g_s)$. Trivial.

Case 2: $h(g_s \diamond \tau_0^s) = h(g_s)$ and $h(g_s \diamond \tau_1^s) \neq h(g_s)$. Trivial.

Case 3: $h(g_s \diamond \tau_0^s) = h(g_s) = h(g_s \diamond \tau_1^s)$.

By (iie), we have that $\varphi_{h(g_s \diamond \tau_1^s)} \neq g$. As $\delta \subseteq \mathbf{Ex}$ and h $\mathbf{G}\delta$ -learns $g \in \mathcal{S}$, there is a $y \geq \#\text{elets}(g_s)$ as required. □ (FOR CLAIM 2)

□

4.3 Complexity Results

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

$$\forall n \in \mathbb{N}, u \in \mathcal{N} : (u \text{ is notation in } \mathcal{N} \text{ for } \alpha + n) \Rightarrow n \leq u. \quad (4.20)$$

The following theorem gives the positive results as to which of several complexity classes *can* be learned with a given delay. The then succeeding theorem, Theorem 4.3.2, states that these results are very tight: none of these classes could be learned with strictly smaller delay. Recall from Definition 2.2.1 that \mathbf{EXPF}_k denotes the set of all functions computable within a timebound given by a stack of k exponentials raised to a polynomial.

Theorem 4.3.1. (a) $\mathbf{PF} \in \mathbf{PFGP}_{\mathbf{cp}_0} \mathbf{Ex}$.

(b) $\mathbf{EXPF} \in \mathbf{PFGP}_{\mathbf{cp}_w} \mathbf{Ex}$.

(c) $\forall n : \mathbf{EXP}_n \mathbf{F} \in \mathbf{PFGP}_{\mathbf{cp}_{w \cdot \underline{n}}} \mathbf{Ex}$.

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

Note that (a) and (b) are *both* special cases of (c). We will prove (a) in detail and will then give a sketch as to how this proof can be generalized to a proof of (c). *Proof of (a).* This proof employs a complexity-bounded enumeration technique [JORS99].

¹⁵ Specific systems of ordinal notations seen in the literature typically, perhaps always, satisfy (4.20).

By [RC94, Theorems 4.13(b) & 4.17] there is a linear time computable e such that $\mathbf{PF} = \{\varphi_{e(j)} \mid j \in \mathbb{N}\}$ and $\forall j \in \mathbb{N} : e(j) \in \text{PolyPrograms}$.

Then, by Lemma 2.2.6 (specifically items (i), (vi), (vii), (ix) with (2.7), (x), (xii) and (xiii)), it is easy to see that there is $h \in \mathbf{PF}$ such that¹⁶

$$\forall \sigma : h(\sigma) = \begin{cases} e(j), & \text{if there is a minimal } j \leq |\sigma| : \forall x < \#\text{elets}(\sigma) : \\ & \varphi_{e(j)}(x) \downarrow_{|\sigma|} = \sigma(x); \\ \text{patch}_0(\sigma), & \text{otherwise.} \end{cases} \quad (4.21)$$

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

- $\forall^\infty n, j_0 \leq n \leq |g[n]|$ (by (2.7)).
- $\forall^\infty n \forall j < j_0 : g[n] \not\leq \varphi_{e(j)}$ (as j_0 minimal such that $\varphi_{e(j_0)} = g$).
- We have $\forall^\infty x : \Phi_{e(j_0)}(x) \leq x$.¹⁷ Hence, $\forall^\infty n \forall x \leq n : \Phi_{e(j_0)}(x) \leq n$.¹⁸ Therefore, using (2.7), $\forall^\infty n \forall x < n : \varphi_{e(j_0)}(x) \downarrow_{|g[n]|}$; hence, also $\forall^\infty n \forall x < n : \varphi_{e(j_0)}(x) \downarrow_{|g[n]|} = \varphi_{e(j_0)}(x) = g(x)$.

By the three items above, we have $\forall^\infty n : h(g[n]) = e(j_0)$. Let $f = \lambda \sigma. \underline{0}$. Obviously, $\langle h, f \rangle$ witnesses $\mathbf{PF} \in \mathbf{GPcp}_0\mathbf{Ex}$. The furthermore clause follows from the choice of e and patch_0 . □ (FOR (a))

¹⁶ Recall that the properties of patch_0 are listed in Definition 2.2.3.

¹⁷ Recall from Section 2.1 that, for each $x \in \mathbb{N}$, $|x|$ denotes the length of x represented dyadically. By [RC94, §2.5, (9)], there are $a, b \in \mathbb{N}$ such that $\forall x : 2^{|x|} \leq a \cdot x + b$; thus, there is $d > 0$ such that $\forall^\infty x : 2^{|x|} \leq d \cdot x$. Clearly, $\forall^\infty x : p(|x|) \leq \frac{1}{d} 2^{|x|}$. Thus, $\forall^\infty x : p(|x|) \leq x$.

¹⁸ Let n_0, n_1 be such that $\forall x \geq n_0 : \Phi_{e(j_0)}(x) \leq x$ and $\forall x < n_0 : \Phi_{e(j_0)}(x) \leq n_1$. Then, for all $n \geq \max\{n_0, n_1\}$ and for all $x \leq n$, we have (if $x < n_0$) $\Phi_{e(j_0)}(x) \leq n_1 \leq n$, and (otherwise) $\Phi_{e(j_0)}(x) \leq x \leq n$.

Proofsketch of (c). Define

$$\forall \sigma, \forall k < \#\text{elets}(\sigma) : f_k(\sigma) = \begin{cases} w \cdot \underline{n-1} + \underline{\exp^1(k) \dot{-} \#\text{elets}(\sigma)}, & \text{if } \#\text{elets}(\sigma) \leq \exp^1(k); \\ \vdots & \vdots \\ w \cdot \underline{1} + \underline{\exp^{n-1}(k) \dot{-} \#\text{elets}(\sigma)}, & \text{else if } \#\text{elets}(\sigma) \leq \\ & \exp^{n-1}(k); \\ w \cdot \underline{0} + \underline{\exp^n(k) \dot{-} \#\text{elets}(\sigma)}, & \text{otherwise.} \end{cases} \quad (4.22)$$

We define $f \in \mathbf{PF}$ by

$$\forall \sigma : f(\sigma) = \langle f_0(\sigma), \dots, f_{\#\text{elets}(\sigma)-1}(\sigma) \rangle_{\text{Seq}}.$$

□ (FOR PROOF SKETCH OF (C))

Theorem 4.3.2. (a) $\forall n \in \mathbb{N} : \mathbf{EXPF} \notin \mathbf{PFGP}_{\mathbf{cp}_n} \mathbf{Ex}$.

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

Proof of (a). Suppose by way of contradiction otherwise as witnessed by n and $\langle h, f \rangle$. Note that $[\mathbf{EXPF}] = \text{Seq}$; thus, $\langle h, f \rangle \in \mathcal{TP}_{\mathbf{cp}_n}$ (see Proposition 4.2.4 above); in particular, h and f are total.

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

```

1  $g_0 \leftarrow \emptyset$ ;
2 for  $s = 0$  to  $\infty$  do
3   if  $h(g_s \diamond \bar{0} \diamond \bar{0}^n) \neq h(g_s)$  then
4      $g_{s+1} = g_s \diamond \bar{0} \diamond \bar{0}^n$ ;
5   else
6      $g_{s+1} = g_s \diamond \bar{1} \diamond \bar{0}^n$ ;

```

Claim 1: h does not converge on g .

We show the claim by showing $\forall s : h(g_{s+1}) \neq h(g_s)$. As $\langle h, f \rangle \in \mathcal{TP}\mathbf{cp}_{\underline{n}}$, we have for all $s \in \mathbb{N}$ and each $j \in \{0, 1\}$, $\lambda_i \leq n \cdot f(g_s \diamond \bar{j} \diamond \bar{0}^i)$ is not an \underline{n} -path, as there is no path of length $n + 1$ in \underline{n} ; hence, $\varphi_{h(g_s \diamond \bar{j} \diamond \bar{0}^n)}(\#\text{elets}(g_s)) = j$.

If $h(g_s \diamond \bar{0} \diamond \bar{0}^n) = h(g_s)$, then

$$\varphi_{h(g_{s+1})}(\#\text{elets}(g_s)) = \varphi_{h(g_s \diamond 1 \diamond \bar{0}^n)}(\#\text{elets}(g_s)) \quad (4.23)$$

$$= 1 \quad (4.24)$$

$$\neq 0 \quad (4.25)$$

$$= \varphi_{h(g_s \diamond \bar{0} \diamond \bar{0}^n)}(\#\text{elets}(g_s)) \quad (4.26)$$

$$= \varphi_{h(g_s)}(\#\text{elets}(g_s)); \quad (4.27)$$

thus, $h(g_{s+1}) \neq h(g_s)$.

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

□ (FOR CLAIM 1)

Claim 2: $g \in \mathbf{EXPF}$.

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

Let p be a polynomial upper-bounding the runtime of h such that $\forall x : x \leq p(x)$. For any stage s , the time to execute stage s is in $\mathcal{O}(\lambda s \cdot p(|g_s \diamond \bar{0}^{n+1}|) + p(|g_s|)) = \mathcal{O}(\lambda s \cdot p(|g_s| + n + 1)) \stackrel{19}{=} \mathcal{O}(\lambda s \cdot p(s \cdot (n + 1) + n + 1)) = \mathcal{O}(\lambda s \cdot p(s))$. Therefore, for all

¹⁹ As g is 0, 1-valued, $\mathcal{O}(|g_s|) = \mathcal{O}(\#\text{elets}(g_s))$.

s , the time to execute all stages 0 to s is bounded above by $\mathcal{O}(\lambda s \cdot (s+1) \cdot p(s)) \subseteq \mathcal{O}(\lambda s \cdot 2^{p'(|s|)})$ for some polynomial p' .²⁰ \square (FOR CLAIM 2) \square (FOR (a))

Proof of (b). Suppose by way of contradiction otherwise as witnessed by $\langle h, f \rangle \in \mathbf{PF}$. The proof requires a definition of g different from that in (a) just above as follows.

```

1  $g_0 \leftarrow \emptyset$ ;
2 for  $s = 0$  to  $\infty$  do
3    $i \leftarrow 0$ ;
4   while  $\#elets(g_s) \notin \perp(f, g_s \diamond \bar{0} \diamond \bar{0}^i)$  do
5      $i \leftarrow i + 1$ ;
6    $j \leftarrow 0$ ;
7   while  $\#elets(g_s) \notin \perp(f, g_s \diamond \bar{1} \diamond \bar{0}^j)$  do
8      $j \leftarrow j + 1$ ;
9   if  $h(g_s \diamond \bar{0} \diamond \bar{0}^i) \neq h(g_s)$  then
10     $g_{s+1} = g_s \diamond \bar{0} \diamond \bar{0}^i$ ;
11  else
12     $g_{s+1} = g_s \diamond \bar{1} \diamond \bar{0}^j$ ;

```

Let p be a polynomial bounding the runtime of h and f , as well as deciders for S and $<_S$. Let s be a stage and $x = \#elets(g_s)$.

Claim: There is a polynomial p' such that each while-loop will terminate after at most $\exp^k(|p'(x)|)$ steps. Let $m \in \{0, 1\}$. Clearly, $f_x(\sigma \diamond \bar{m} \diamond \bar{0}^n) <_S w \cdot k$. By runtime considerations and (4.20) we see $f_x(g_s \diamond \bar{m} \diamond \bar{0}^n) <_S w \cdot (k-1) + \underline{\exp(p(x_s + n + 1))}$;

²⁰ Let k be such that $\mathcal{O}(p) = \mathcal{O}(\lambda x \cdot x^k)$. By [RC94, §2.5, (9)], there are a, b such that $x \leq a \cdot 2^{|x|} + b$. Thus, there are c, d, c', d' such that $\forall x : p(x) \leq c \cdot x^k + d \leq c \cdot (a \cdot 2^{|x|} + b)^k + d \leq c' \cdot 2^{k \cdot |x|} + d'$.

hence, for some polynomial p_1 , $f_x(\sigma \diamond \bar{m} \diamond \bar{0}^{\overline{\exp(p_1(x_s))}}) <_S f_x(\sigma \diamond \bar{m} \diamond \bar{0}^{\overline{n+\exp(p(x_s+n+1))}}) <_S w \cdot k - 1$. Inductively one can now see that there is a polynomial p_{k-1} such that

$$f_x(\sigma \diamond \bar{m} \diamond \bar{0}^{\overline{\exp^{k-1}(p_{k-1}(x_s))}}) <_S w; \quad (4.28)$$

in particular, one can see that there is a polynomial p_k such that $x \in \perp(f, 0 \diamond \bar{m} \diamond \bar{0}^{\overline{\exp^k(p_k(x_s))}})$. \square (FOR CLAIM)

Using [RC94, Theorem 3.17], one can now see $g \in \mathbf{EXP}_{k+1}\mathbf{F}$. The rest of the proof is analogous to the proof of (a). \square (FOR (b))

4.4 Results mostly not Comparing Countdown Graphs

Akama and Zeugmann [AZ08] showed that local postdictive completeness is equivalent to global postdictive completeness for all finitely delayed versions thereof. The following theorem states that this result extends to arbitrary countdown graphs, as well as to postdictive consistency, and the proof is simple extension of the proof in [AZ08].

Theorem 4.4.1. Let $G \in \mathcal{G}$. We have

- (a) $\mathcal{TPcp}_G\mathbf{GEx} = \mathcal{TPcpl}_G\mathbf{GEx}$ and $\mathcal{TPcs}_G\mathbf{GEx} = \mathcal{TPcsl}_G\mathbf{GEx}$;
- (b) $\mathcal{RGPcp}_G\mathbf{Ex} = \mathcal{RGPcpl}_G\mathbf{Ex}$ and $\mathcal{RGPcs}_G\mathbf{Ex} = \mathcal{RGPcsl}_G\mathbf{Ex}$;
- (c) $\mathbf{GPcp}_G\mathbf{Ex} = \mathbf{GPcpl}_G\mathbf{Ex}$ and $\mathbf{GPcs}_G\mathbf{Ex} = \mathbf{GPcsl}_G\mathbf{Ex}$.

Proof: “ \subseteq ” are clear.

Let $I \in \{\mathcal{TPcpl}_G\mathbf{GEx}, \mathcal{RGPcpl}_G\mathbf{Ex}, \mathbf{GPcpl}_G\mathbf{Ex}, \mathcal{TPcsl}_G\mathbf{GEx}, \mathcal{RGPcsl}_G\mathbf{Ex}, \mathbf{GPcsl}_G\mathbf{Ex}\}$ and $\mathcal{S} \in I$ as witnessed by $\langle h, f \rangle$. Define now $h' \in \mathcal{P}$ such that

$$\forall \sigma : h'(\sigma) = \begin{cases} h'(\sigma^-), & \text{if } \sigma \neq \emptyset \wedge h(\sigma) = h(\sigma^-); \\ \text{patch}(\sigma, h(\sigma)), & \text{otherwise.} \end{cases}$$

It is obvious that for all $g \in \mathcal{R}$, if h converges on g to an index of g , then so does h' . Furthermore, if h is total, so is h' . What is left to show is summarized in the following claim.

Claim: Let $g \in \mathcal{R}$ such that $\langle h, f \rangle$ works locally postdictively completely (consistently) on g . Then $\langle h', f \rangle$ works postdictively completely (consistently) on g .

Let $\sigma, n \in \perp(f, \sigma)$. Let $\sigma_0 \subseteq \sigma$ be such that $h'(\sigma) = \text{patch}(\sigma_0, h(\sigma_0))$. If $n < \#\text{elets}(\sigma_0)$, then we have $\varphi_{h'(\sigma)}(n) = \sigma_0(n) = \sigma(n)$. Otherwise find σ_1 such that $\sigma_0 \subseteq \sigma_1 \subseteq \sigma$ such that $n \in \perp(f, \sigma_1)$. By construction of h' we have $h(\sigma_1) = h(\sigma_0)$; therefore,

$$\varphi_{h'(\sigma)}(n) = \varphi_{h'(\sigma_0)}(n) = \varphi_{\text{patch}(\sigma_0, h(\sigma_0))}(n) = \varphi_{h(\sigma_0)}(n) = \varphi_{h(\sigma_1)}(n). \quad (4.29)$$

□

The following theorem shows the relationship between the different learning criteria as defined in this chapter. As noted above, this is one of the main theorems in this chapter.

Theorem 4.4.2. We have the following.

$$\forall G \in \mathcal{G}_{\text{comp}} : \mathcal{TPcp}_G \mathbf{GEx} = \mathcal{TPcs}_G \mathbf{GEx}. \quad (4.30)$$

$$\mathbf{LinFTdPcs}_{\emptyset} \mathbf{Ex} \setminus \left(\bigcup_{G \in \mathcal{G}} \mathbf{GPcp}_G \mathbf{Ex} \right) \neq \emptyset. \quad (4.31)$$

$$\mathbf{LinFTdPcp}_{\emptyset} \mathbf{Ex} \setminus \left(\bigcup_{G \in \mathcal{G}} \mathcal{TPcp}_G \mathbf{GEx} \right) \neq \emptyset. \quad (4.32)$$

$$\mathbf{GPcp}_{\emptyset} \mathbf{Ex} \setminus \left(\bigcup_{G \in \mathcal{G}_{\text{comp}}} \mathcal{RGPcs}_G \mathbf{Ex} \right) \neq \emptyset. \quad (4.33)$$

Proof of (4.30). This proof of (4.30) above is an extension of Fulk's proof of the $G = \emptyset$ case [Ful88]. Let $G \in \mathcal{G}_{\text{comp}}$.

“ \subseteq ”: Clear.

“ \supseteq ”: Let $\mathcal{S} \in \mathcal{TPcs}_G \mathbf{GEx}$ as witnessed by $\langle h, f \rangle \in \mathcal{TPcs}_G$. Let R be a computable predicate such that, for all p, p', n, τ, t ,

$$\begin{aligned} R(p, p', n, \tau, t) \text{ iff } & [p' = h(\tau) \wedge n \in \perp(f, \tau) \wedge \\ & \forall n' < n : (n' \in \perp(f, \tau) \Rightarrow (\varphi_p(n') \downarrow = \tau(n') \text{ in } \leq t \text{ steps}))]. \end{aligned}$$

By 1-1 **ORT** (Theorem 2.3.8), there is a 1-1 $e \in \mathcal{R}$ such that

$$\forall \sigma, n : \varphi_{e(\sigma)}(n) = \begin{cases} \sigma(n), & \text{if } n \in \text{dom}(\sigma); \\ \tau_0(n), & \text{else if } \exists \langle \tau_0, t_0 \rangle = \\ & \mu \langle \tau, t \rangle \cdot R(e(\sigma), h(\sigma), n, \tau, t); \\ \uparrow, & \text{otherwise.}^{21} \end{cases} \quad (4.34)$$

Let S be a partial computable predicate such that, for all p, p', n, σ ,

$$\begin{aligned} S(p, p', n, \sigma) \text{ iff } & [p' = h(\sigma) \wedge \exists t : R(p, p', n, \sigma, t) \\ & \text{and with } \langle \tau_0, t_0 \rangle = \mu \langle \tau, t \rangle \cdot R(p, p', n, \tau, t) \\ & \text{we have } \sigma(n) = \tau_0(n)]. \end{aligned} \quad (4.35)$$

Define h' such that, for all σ ,

$$h'(\sigma) = \begin{cases} \text{if } \sigma \neq \emptyset \wedge h(\sigma) = h(\sigma^-) \wedge \text{with } \sigma_0 \text{ such} \\ h'(\sigma^-), & \text{that } h'(\sigma^-) = e(\sigma_0) : \forall n \in \perp(f, \sigma): \\ & S(h'(\sigma^-), h(\sigma_0), n, \sigma); \\ e(\sigma), & \text{otherwise.} \end{cases} \quad (4.36)$$

Claim 1: $h' \in \mathcal{R}$.

We prove $\forall \sigma \exists \sigma_0 \forall \sigma_1 : \sigma_0 \subseteq \sigma_1 \subseteq \sigma \Rightarrow h'(\sigma) \downarrow = e(\sigma_0)$ by induction on σ , trivial for $\sigma = \emptyset$. Let σ and σ_0 be given such that

$$\forall \sigma_1 : \sigma_0 \subseteq \sigma_1 \subseteq \sigma^- \Rightarrow h'(\sigma_1) \downarrow = e(\sigma_0). \quad (4.37)$$

²¹ Note that this application of (1-1) **ORT** employs only a special case of (1-1) **ORT**, namely (the 1-1 case of) Kleene's Parametric Recursion Theorem [Rog67, § 11]. This is because the $e(\sigma)$ of (4.34) generates (and employs) both σ and a copy of $e(\sigma)$, *but* $e(\sigma)$ does not also generate and employ a copy of $e(\sigma')$, for some $\sigma' \neq \sigma$.

Claim 1.1:

$$\forall n : (\exists \tau : S(e(\sigma_0), h(\sigma_0), n, \tau) \downarrow = \text{true}) \Rightarrow \varphi_{e(\sigma_0)}(n) \downarrow. \quad (4.38)$$

Proof of Claim 1.1. Let $n \in \mathbb{N}$ such that $\exists \tau : S(e(\sigma_0), h(\sigma_0), n, \tau) \downarrow = \text{true}$. By the definition of S , there are now τ, t such that $R(e(\sigma_0), h(\sigma_0), n, \tau, t)$. The definition of $e(\sigma_0)$ shows that $\varphi_{e(\sigma_0)}(n) \downarrow$. \square (FOR CLAIM 1.1)

Claim 1.2:

$$\forall n \in \perp(f, \sigma^-) : \varphi_{e(\sigma_0)}(n) \downarrow. \quad (4.39)$$

Proof of Claim 1.2. Let $n \in \perp(f, \sigma^-)$. If $n < \#elets(\sigma_0)$, then, trivially, $\varphi_{e(\sigma_0)}(n) \downarrow$. Suppose now $n \geq \#elets(\sigma_0)$. By choice of n and the definition of \perp , there is then σ_1 such that $\sigma_0 \subset \sigma_1 \subset \sigma$ and $n \in \perp(f, \sigma_1)$. From (4.37) we have that in the definition of $h'(\sigma_1)$ the first case holds. Thus, $S(e(\sigma_0), h(\sigma_0), n, \sigma_1) \downarrow = \text{true}$. By (4.38) we have $\varphi_{e(\sigma_0)}(n) \downarrow$.

\square (FOR CLAIM 1.2)

Obviously, it suffices now to show the following claim.

Claim 1.3: For all $n \in \perp(f, \sigma)$,

$$(\forall n' \in \perp(f, \sigma) : n' < n \Rightarrow S(e(\sigma_0), h(\sigma_0), n', \sigma) \downarrow = \text{true}) \Rightarrow S(e(\sigma_0), h(\sigma_0), n, \sigma) \downarrow. \quad (4.40)$$

Proof of Claim 1.3. Let $n \in \perp(f, \sigma)$ be such that the antecedent of (4.40) holds. Using (4.38) and (4.39) we now have

$$\forall n' \in \perp(f, \sigma) : n' < n \Rightarrow \varphi_{e(\sigma_0)}(n') \downarrow = \sigma(n). \quad (4.41)$$

We show $S(e(\sigma_0), h(\sigma_0), n, \sigma) \downarrow$. If we can show the second conjunct of $S(\dots)$ to hold, then the minimization in the third conjunct of S will also terminate. Hence, it remains to show $\exists t R(e(\sigma_0), h(\sigma_0), n, \sigma, t)$. $h(\sigma_0) = h(\sigma)$ and $n \in \perp(f, \sigma)$ are clear, the remainder follows by (4.41). \square (FOR CLAIM 1.3) \square (FOR CLAIM 1)

Claim 2: $\langle h', f \rangle \in \mathcal{TPcp}_G$.

Let $\sigma, \sigma_0 \in \text{Seq}$ be such that $h'(\sigma) = e(\sigma_0)$. Obviously, using induction, it now suffices to show $\forall n \in \perp(f, \sigma) : \varphi_{e(\sigma_0)}(n) \downarrow = \sigma(n)$. Let $n \in \perp(f, \sigma)$. We have $S(e(\sigma_0), h(\sigma_0), n, \sigma) \downarrow = \text{true}$. From the definitions of S and $e(\sigma_0)$ we now see $\varphi_{e(\sigma_0)}(n) = \sigma(n)$, as both minimizations give the same result. \square (FOR CLAIM 2)

Claim 3: $\langle h', f \rangle$ witnesses $\mathcal{S} \in \mathcal{TPcp}_G \mathbf{GEx}$.

Let $g \in \mathcal{S}$. There exists $\sigma_{\rightarrow} \subset g$ minimal such that $\forall \sigma : \sigma_{\rightarrow} \subseteq \sigma \subset g \Rightarrow h(\sigma) = h(\sigma_{\rightarrow})$ and $\varphi_{h(\sigma_{\rightarrow})} = g$.

We proceed by showing $\forall \sigma : \sigma_{\rightarrow} \subset \sigma \subset g \Rightarrow h'(\sigma) = h'(\sigma_{\rightarrow})$. Let σ be such that $\sigma_{\rightarrow} \subset \sigma \subset g$. Obviously, using induction, it suffices to show that $h'(\sigma)$ is defined according to the first case. Let σ_0 be such that $h'(\sigma^-) = e(\sigma_0)$. Note that, by the second conjunct in the cases for (4.36) and because of the minimality of σ_{\rightarrow} , $\sigma_{\rightarrow} \subseteq \sigma_0$; hence,

$$h(\sigma) = h(\sigma_{\rightarrow}) = h(\sigma_0). \quad (4.42)$$

Let $n \in \perp(f, \sigma)$. We show $S(e(\sigma_0), h(\sigma_0), n, \sigma)$. By (4.42), we have $h(\sigma_0) = h(\sigma)$. As shown in the proof of Claim 1, $\exists t : R(e(\sigma_0), h(\sigma_0), n, \sigma, t)$. Then the minimization in the definition of $S(e(\sigma_0), h(\sigma_0), n, \sigma)$ will terminate. Let $\langle \tau_0, t_0 \rangle = \mu \langle \tau, t \rangle . R(e(\sigma_0), h(\sigma_0), n, \tau, t)$. By definition of R we have now $h(\tau_0) = h(\sigma_0) \stackrel{(4.42)}{=} h(\sigma_{\rightarrow})$; hence,

$$\varphi_{h(\tau_0)}(n) = \varphi_{h(\sigma_{\rightarrow})}(n) = g(n) \downarrow;$$

therefore, $\varphi_{h(\tau_0)}(n) \downarrow$, and, by postdictive consistency, $\tau_0(n) = \varphi_{h(\tau_0)}(n) = g(n) = \sigma(n)$. \square (FOR CLAIM 3) \square (FOR (4.30))

Proof of (4.31). Let $\mathcal{S} = \{g \in \mathcal{R} \mid (\bar{0} \diamond (\pi_1 \circ g), g) \in \mathbf{Pcs}_{\emptyset} \mathbf{Ex}\}$. Obviously, $\mathcal{S} \in \mathbf{LinFTdPcs}_{\emptyset} \mathbf{Ex} \subseteq \mathcal{RGPs}_{\emptyset} \mathbf{Ex}$. Let $G \in \mathcal{G}$. We set up to use Lemma 4.2.5. Suppose by way of contradiction $\mathcal{S} \in \mathbf{GPcp}_G \mathbf{Ex}$, as witnessed by $\langle h, f \rangle$. Define, for all $e \in \mathbb{N}$,

$\mathcal{S}_e = \{g \in \mathcal{R} \mid \forall i : \pi_1(\pi_1(g(i))) = e\}$. Note that $[\mathcal{S}_e]$ is uniformly computable in e . Define $t \in \mathcal{P}$ by setting for all e, σ ,

$$t(e, \sigma) = \mu\langle \rho, s \rangle. \sigma \diamond \rho \in [\mathcal{S}_e] \wedge (h(\sigma)\downarrow \neq h(\sigma \diamond \rho)\downarrow \text{ each in } \leq s \text{ steps}).$$

Claim: Let $e \in \mathbb{N}, \sigma \in [\mathcal{S}_e]$, such that $\varphi_e = \sigma$. Then $t(e, \sigma)\downarrow$.

Let, for each $j \in \{0, 1\}$,

$$\begin{aligned} n_j &= \overline{\langle \langle e, 0 \rangle, j \rangle}; \\ i_j &= \mu i > 0. \#elets(\sigma) \in \perp(f, \sigma \diamond n_j^i); \\ \rho_j &= n_j^{i_j}. \end{aligned}$$

Note that i_j may not be defined and when defined not algorithmically extractable from e, σ and j , as \perp_G is not necessarily computable (since G is not necessarily computable). For all $j \in \{0, 1\}$ and $i \in \mathbb{N}$ we have $\sigma \diamond n_j^i \in [\mathcal{S}]$, as $\varphi_e = \sigma$; thus, as $\langle h, f \rangle$ **GPcp_GEx**-learns \mathcal{S} , $\langle h, f \rangle(\sigma \diamond n_j^i)\downarrow$. Hence, for each $j \in \{0, 1\}$, i_j and ρ_j are well defined. We have now

$$\varphi_{h(\sigma \diamond \rho_0)}(\#elets(\sigma))\downarrow = \rho_0(0) \neq \rho_1(0) = \varphi_{h(\sigma \diamond \rho_1)}(\#elets(\sigma))\downarrow;$$

thus, $h(\sigma \diamond \rho_0) \neq h(\sigma \diamond \rho_1)$ and $t(e, \sigma)\downarrow$.

□ (FOR CLAIM)

By setting $t_0 = t_1 = t$, we can now use Lemma 4.2.5 to show (4.31).

□ (FOR (4.31))

*Proof of (4.32).*²² Let $\mathcal{S} = \{g \in \mathcal{R} \mid (\bar{0} \diamond (\pi_1 \circ g), g) \in \mathbf{Pcp}_\emptyset \mathbf{Ex}\}$. Obviously, $\mathcal{S} \in \mathbf{LinFTdPcp}_\emptyset \mathbf{Ex} \subseteq \mathbf{RGpcp}_\emptyset \mathbf{Ex}$. Let $G \in \mathcal{G}$. We set up to use Lemma 4.2.5.

²² An anonymous referee pointed out that (4.32) can easily be proven by showing that all sets in $\mathbf{TPcp}_\emptyset \mathbf{GEx}$ can be reliably learned, as it is known that not all reliably learnable sets are $\mathbf{RGpcp}_G \mathbf{Ex}$ -learnable [CJSW04]. We retain our original proof of (4.32) herein, since it exercises, in a simple way, an application of Lemma 4.2.5.

Suppose by way of contradiction $\mathcal{S} \in \mathcal{TPcp}_G \mathbf{GEx}$ as witnessed by $\langle h, f \rangle$. Define, for all $e \in \mathbb{N}$, $\mathcal{S}_e = \{g \in \mathcal{R} \mid \forall i : \pi_1(\pi_1(g(i))) = e\}$. Note that $[\mathcal{S}_e]$ is uniformly computable in e . Define $t \in \mathcal{P}$ by setting for all e, σ ,

$$t(e, \sigma) = \mu \rho. \sigma \diamond \rho \in [\mathcal{S}_e] \wedge h(\sigma) \neq h(\sigma \diamond \rho).$$

Claim: Suppose $e \in \mathbb{N}, \sigma \in \text{Seq}$. Then $t(e, \sigma) \downarrow$.

Let, for each $j \in \{0, 1\}$,

$$\begin{aligned} n_j &= \overline{\langle \langle e, 0 \rangle, j \rangle}; \\ i_j &= \mu i > 0. \# \text{elets}(\sigma) \in \perp(f, \sigma \diamond n_j^i); \\ \rho_j &= n_j^{i_j}. \end{aligned}$$

Note that i_j may not be defined and when defined not algorithmically extractable from e, σ and j , as \perp_G not necessarily computable (since G is not necessarily computable). As $\langle h, f \rangle \in \mathcal{TPcp}_G$, we have that, for each $j \in \{0, 1\}$, i_j and ρ_j are defined and we have

$$\varphi_{h(\sigma \diamond \rho_0)}(\# \text{elets}(\sigma)) \downarrow = \rho_0(0) \neq \rho_1(0) = \varphi_{h(\sigma \diamond \rho_1)}(\# \text{elets}(\sigma)) \downarrow;$$

thus, $h(\sigma \diamond \rho_0) \neq h(\sigma \diamond \rho_1)$; therefore, as $\sigma \diamond \rho_0, \sigma \diamond \rho_1 \in [\mathcal{S}_e]$, $t(e, \sigma) \downarrow$.

□ (FOR CLAIM)

By setting $t_0 = t_1 = t$, we can now use Lemma 4.2.5 to show $\mathcal{S} \notin \mathcal{TPcp}_G \mathbf{GEx}$, a contradiction. □ (FOR (4.32))

Proof of (4.33). Let $\mathcal{S} = \{g \in \mathcal{R} \mid (\bar{0} \diamond \lambda n. \langle \varphi_{g(n)}(0), 0 \rangle, g) \in \mathbf{Pcp}_\emptyset \mathbf{Ex}\}$. Obviously, $\mathcal{S} \in \mathbf{GPcp}_\emptyset \mathbf{Ex}$. Let $G \in \mathcal{G}_{\text{comp}}$. We set up to use Lemma 4.2.5. Suppose now, by way of contradiction, $\mathcal{S} \in \mathcal{RGPcs}_G \mathbf{Ex}$, as witnessed by

$$\langle h, f \rangle \in \mathcal{R}. \tag{4.43}$$

By 1-1 **ORT** (Theorem 2.3.8), there is a 1-1 function $P \in \mathcal{R}$ such that

$$P = \lambda\langle d, j, e, \sigma, n \rangle. = \begin{cases} \ell & \text{if } d = 0; \\ p_{j,e,\sigma}(n) & \text{otherwise,} \end{cases} \quad (4.44)$$

where ℓ and $p_{j,e,\sigma}$ are defined just below.²³

$$\forall j, e, \sigma : \varphi_\ell(j, e, \sigma) = \mu i > 0. \#elets(\sigma) \in \perp(f, \sigma \diamond p_{j,e,\sigma}[i]) \quad (4.45)$$

$$\forall e, \sigma, n, x : \varphi_{p_{0,e,\sigma}(n)}(x) = \begin{cases} \text{patch}_0(\sigma \diamond p_{0,e,\sigma}[n+1]), & \text{if } h(\sigma) = h(\sigma \diamond p_{0,e,\sigma}[\varphi_\ell(0, e, \sigma)]) \\ e, & \text{otherwise; and} \end{cases} \quad (4.46)$$

$$\forall n, x \forall j > 0 : \varphi_{p_{j,e,\sigma}(n)}(x) = e. \quad (4.47)$$

Clearly, as P above is total, and by (4.44), we have that each function $p_{j,e,\sigma}$ is total. Hence, by (4.43) and (4.45) we have that φ_ℓ is total. Therefore, by (4.46), for all j, e, σ , $\varphi_{p_{j,e,\sigma}}$ is total. For each $j \in \{0, 1\}$, define

$$t_j(e, \sigma) = p_{j,e,\sigma}[\varphi_\ell(j, e, \sigma)]. \quad (4.48)$$

By the discussion before (4.48) we have for all j ,

$$t_j \in \mathcal{R}. \quad (4.49)$$

Let, for all $e \in \mathbb{N}$, $\mathcal{S}_e = \{g \in \mathcal{R} \mid \forall n \in \mathbb{N} : \varphi_{g(n)}(0) = e\}$. We apply Lemma 4.2.5 with $\mathcal{C} = \mathcal{R}$ and $\delta = \mathbf{Pcs}_G \mathbf{Ex}$. (i) is trivial. To show (ii), let $e \in \mathbb{N}, \sigma \in [\mathcal{S}_e]$. (a) follows from (4.43) and (4.49). The conclusion of (b) is trivial from (4.48). (c) follows from (4.45), (4.46) and (4.48). (d) is trivial from (4.47). We show (e) by showing the contrapositive: Suppose $\varphi_{h(\sigma \diamond \tau_1)} \in \mathcal{R}$. Define, for each $j \in \{0, 1\}$, $\tau_j = t_j(e, \sigma)$. Note that, as P is 1-1, we have $\tau_0(0) = p_{0,e,\sigma}(0) \neq p_{1,e,\sigma}(0) = \tau_1(0)$. Then, by (4.45),

$$\varphi_{h(\sigma \diamond \tau_0)}(\#elets(\sigma)) = \tau_0(0) \neq \tau_1(0) = \varphi_{h(\sigma \diamond \tau_1)}(\#elets(\sigma)).$$

²³ Recall that the properties of patch_0 are listed in Definition 2.2.3.

Thus, $h(\sigma \diamond \tau_0) \neq h(\sigma \diamond \tau_1)$, which shows (iie). Lemma 4.2.5 gives now $\mathcal{S} \notin \mathcal{RGPcs}_G \mathbf{Ex}$, a contradiction. \square (FOR (4.33))

[Bär74b] and [BB75] show independently that \mathbf{GEx} is *not* closed under binary union.

However, some learnability classes are shown in the literature to be closed even under ce unions, for example Minicozzi showed this for $\text{Rel}_{\mathcal{R}} \mathbf{GEx}$ [Min76, BB75].

[AZ08] gives an analogous result for total postdictive completeness with *finite* countdowns, which we further generalize to arbitrary countdown graphs in the following theorem.

Theorem 4.4.3. Let $G \in \mathcal{G}$. Then $\mathcal{TPcp}_G \mathbf{GEx}$ is closed under ce unions.

Proof: Suppose, for each $i \in \mathbb{N}$,

$$\langle h^i, f^i \rangle \text{ witnesses } \mathcal{S}_i \in \mathcal{TPcp}_G \mathbf{GEx}, \quad (4.50)$$

such that $\lambda i, \sigma. \langle h^i(\sigma), f^i(\sigma) \rangle$ is computable.

Define $n, y, z, h^\infty, f^\infty \in \mathcal{R}$ such that, for all σ and for all $k < \#\text{elets}(\sigma)$,

$$n(\sigma) = \begin{cases} \mu m \leq \#\text{elets}(\sigma). (\exists i \leq \#\text{elets}(\sigma)) (\forall j | m \leq j < \#\text{elets}(\sigma)) : \\ \quad h^i(\sigma) = h^i(\sigma[j]); \end{cases} \quad (4.51)$$

$$y(\sigma) = \mu i \leq \#\text{elets}(\sigma). (\forall j | n(\sigma) \leq j < \#\text{elets}(\sigma)) : h^i(\sigma) = h^i(\sigma[j]); \quad (4.52)$$

$$z(\sigma) = \mu i \leq \#\text{elets}(\sigma). (\forall j | i \leq j < \#\text{elets}(\sigma)) : y(\sigma) = y(\sigma[j]); \quad (4.53)$$

$$h^\infty(\sigma) = \text{patch}(\sigma[z(\sigma)], h^{y(\sigma)}(\sigma)); \quad (4.54)$$

$$f^\infty(\sigma) = f^{y(\sigma)}(\sigma). \quad (4.55)$$

Intuitively, y indicates which learner to use when seeing σ , and z indicates at how much of σ was the last mind change of y . $\langle h^\infty, f^\infty \rangle$ is our learner for the union.

Claim 1: $\langle h^\infty, f^\infty \rangle$ works postdictively completely on all $g \in \mathcal{R}$.

Let $\sigma \in \text{Seq}$, let $k \in \perp(f, \sigma)$. Let $z_0 = z(\sigma)$.

Case 1: $k < z_0$.

Then we have

$$\varphi_{h^\infty(\sigma)}(k) \stackrel{(4.54)}{=} \varphi_{\text{patch}(\sigma[z_0], h^{y(\sigma)}(\sigma))}(k) \stackrel{k < z_0}{=} \sigma(k).$$

Case 2: $k \geq z_0$.

Then we have

$$\begin{aligned} & \text{row}(k, f^\infty, \sigma) \\ & \stackrel{(4.10)}{=} \lambda i \leq \#\text{elets}(\sigma) - k. f^\infty(\sigma[i+k])(\#\text{elets}(\sigma)) \\ & \stackrel{(4.54)}{=} \lambda i \leq \#\text{elets}(\sigma) - k. f^{y(\sigma[i+k])}(\sigma[i+k])(\#\text{elets}(\sigma)) \\ & \stackrel{k \geq z_0}{=} \lambda i \leq \#\text{elets}(\sigma) - k. f^{y(\sigma)}(\sigma[i+k])(\#\text{elets}(\sigma)) \\ & \stackrel{(4.10)}{=} \text{row}(k, f^{y(\sigma)}, \sigma). \end{aligned} \tag{4.56}$$

Hence, we have

$$k \in \perp(f^\infty, \sigma) \stackrel{(4.11)}{\Leftrightarrow} \text{row}(k, f^\infty, \sigma) \notin \vec{G} \tag{4.57}$$

$$\stackrel{(4.56)}{\Leftrightarrow} \text{row}(k, f^{y(\sigma)}, \sigma) \notin \vec{G} \tag{4.58}$$

$$\stackrel{(4.11)}{\Leftrightarrow} k \in \perp(f^{y(\sigma)}, \sigma) \tag{4.59}$$

$$\stackrel{(4.50)}{\Rightarrow} \varphi_{h^{y(\sigma)}(\sigma)}(k) = \sigma(k) \tag{4.60}$$

$$\stackrel{(4.54)}{\Leftrightarrow} \varphi_{h^\infty(\sigma)}(k) = \sigma(k). \tag{4.61}$$

□ (FOR CLAIM 1)

Claim 2: h^∞ converges on all $g \in \cup_j \mathcal{S}_j$ to a program number for g .

Let $g \in \cup_j \mathcal{S}_j$. Then $\lambda k.n(g[k])$ converges. Hence, also $\lambda k.y(g[k])$ converges. Let y_0 be the limit of $\lambda k.y(g[k])$. Therefore, $\langle h^{y_0}, f^{y_0} \rangle$ converges on g ; hence $\langle h^\infty, f^\infty \rangle$ converges on g . As $\langle h^\infty, f^\infty \rangle$ is postdictively complete by Claim 1, Claim 2 follows.

□ (FOR CLAIM 2)

□

The following theorem states that there are **GEx**-learnable classes which are *not* learnable in *any* of the restricted criteria discussed in the present chapter.

Theorem 4.4.4. We have

$$\bigcup_{G \in \mathcal{G}} \mathbf{GPcs}_G \mathbf{Ex} \subset \mathbf{GEx}. \quad (4.62)$$

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

Proof: “ \subseteq ” is trivial.

“ \neq ”: Let $h_0 = \lambda\langle c, x \rangle. \pi_1(x)$, let $\mathcal{S} = \mathbf{TdEx}(h)$. Obviously, $\mathcal{S} \in \mathbf{LinFTdEx} \subseteq \mathbf{GEx}$. Suppose, by way of contradiction, there are $G \in \mathcal{G}$ and $\langle h, f \rangle \in \mathcal{P}$ such that $\langle h, f \rangle$ witnesses $\mathcal{S} \in \mathbf{GPcs}_G \mathbf{Ex}$. Note that

$$[\mathcal{S}] = \text{Seq}. \quad (4.63)$$

Hence, $\langle h, f \rangle \in \mathcal{R}$. We set up to use Lemma 4.2.5. Let, for all e , $\mathcal{S}_e = \{g \in \mathcal{R} \mid \forall i : \pi_1(g(i)) = e\}$. $[\mathcal{S}_e]$ is uniformly computable in e . Define $t \in \mathcal{P}$ such that

$$\forall e, \sigma : t(e, \sigma) = \mu\tau. (\sigma \diamond \tau \in [\mathcal{S}_e] \wedge h(\sigma) \neq h(\sigma \diamond \tau)). \quad (4.64)$$

Claim: For all $e \in \mathbb{N}, \sigma \in [\mathcal{S}_e], t(e, \sigma) \downarrow$.

Suppose, by way of contradiction, there are $e \in \mathbb{N}, \sigma \in [\mathcal{S}_e]$ such that $t(e, \sigma) \uparrow$. Hence,

$$\forall \tau : \sigma \diamond \tau \in [\mathcal{S}_e] \Rightarrow h(\sigma) = h(\sigma \diamond \tau). \quad (4.65)$$

Obviously, there are τ, τ' such that $\sigma \diamond \tau, \sigma \diamond \tau' \in [\mathcal{S}_e]$, $\#elets(\sigma) \in \perp(f, \sigma \diamond \tau)$, $\#elets(\sigma) \in \perp(f, \sigma \diamond \tau')$ and $\tau(0) \neq \tau'(0)$. As $\sigma \diamond \tau, \sigma \diamond \tau' \in [\mathcal{S}_e] \subseteq [\mathcal{S}]$, and $\langle h, f \rangle$ works postdictively consistently on \mathcal{S} , we have with (4.65), $\varphi_{h(\sigma)}(\#elets(\sigma)) \uparrow$. Let $g \in \mathcal{S}_e$ be an extension of σ . By (4.65), h on g converges to $h(\sigma)$, which is not a program number for g (as $\varphi_{h(\sigma)}$ is not total), a contradiction. □ (FOR CLAIM)

We apply Lemma 4.2.5 with $t_0 = t_1 = t$, $\mathcal{C} = \mathcal{P}$ and $\delta = \mathbf{Pcs}_G \mathbf{Ex}$. Therefore, $\mathcal{S} \notin \mathbf{GPcs}_G \mathbf{Ex}$, a contradiction. \square

4.5 Dependencies on the Countdown Graphs

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

Definition 4.5.1. For two graphs G, G' we write $G \leq_{CD} G'$ (read: G is *countdown reducible to* G') iff there is a $k \in \mathcal{R}$, such that

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

Intuitively, k maps any G -path into a vertex of G' .²⁴ Clearly, \leq_{CD} is a pre-order.

The following proposition is technically useful for later proofs.

Proposition 4.5.2. Let $G, G' \in \mathcal{G}$. Let $k \in \mathcal{R}$. The following are equivalent.

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

Proof: “(a) \Rightarrow (b)”: Let $\tau \in \vec{G}$. (b) is immediate if $\#elets(\tau) = 0$. If $\#elets(\tau) = 1$ with, for some $y \in G$, $\tau = \bar{y}$, then (i) of Definition 4.5.1 gives $k(\bar{y}) \in G'$. Suppose now $\#elets(\tau) > 1$. For all $i < \#elets(\tau) - 1$ we have $k(\tau[i+1]) \rightarrow_G k(\tau[i+2])$ by (ii) of Definition 4.5.1. Hence, in each case, $(\lambda i < \#elets(\tau) . k(\tau[i+1])) \in \vec{G}'$.

“(b) \Rightarrow (a)”: For all $y \in G$, we have $\overline{k(\bar{y})} \in \vec{G}'$. Furthermore, for all $\tau \diamond \bar{y} \in \vec{G}$ such

²⁴ Neither of mapping G vertices into G' vertices nor mapping G paths into G' paths will give us the same characterization results that we have in Theorem 4.5.6 below.

that $\#elets(\tau) > 0$, we have $\lambda i < (\#elets(\tau) + 1) \cdot k((\tau \diamond \bar{y})[i + 1]) \in \vec{G}'$. Thus, $k(\tau) \rightarrow_{G'} k(\tau \diamond \bar{y})$. \square

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

Definition 4.5.3. We will use the following computability-theoretic notions.

- A set $A \subseteq \mathbb{N}$ is called *semi-recursive* iff (by a characterization by McLaughlin and Appel, cited in [Joc68, Theorem 4.1(iii)]) A is an initial segment of some computable linear ordering of the natural numbers.
- A set $A \subseteq \mathbb{N}$ is called *immune* iff A is infinite and does not contain a *ce* set [Rog67, § 8.2].
- A set $A \subseteq \mathbb{N}$ is called *hyper-immune* iff A is infinite and for the unique $r \in \mathfrak{R}$ strictly monotonic increasing such that $\text{range}(r) = A$ we have $\forall f \in \mathcal{R} \exists x \in \mathbb{N} : f(x) < r(x)$ [Rog67, § 9.5]. Note that every hyper-immune set is immune [Rog67, § 9.5].

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

Theorem 4.5.4. There are a computable total ordering \leq_R on \mathbb{N} and a set $A \subseteq \mathbb{N}$ such that A is semi-recursive, A and \bar{A} are hyperimmune, hence immune, $\leq_R|_A$ an initial segment of \leq_R , $\leq_R|_A$ is of order-type ω and $\leq_R|\bar{A}$ is of order-type ω^{-1} . In particular, \leq_R is of order-type $\omega + \omega^{-1}$ and there are no computable infinitely descending chains with respect to \leq_R ; hence, $(\mathbb{N}, >_R)$ is a countdown graph.

Proof: By [Joc68, Theorem 5.2], there is a semi-recursive, hyper-immune set A , such that \bar{A} is hyper-immune. As A semi-recursive, there exists a computable total ordering \leq_R on \mathbb{N} such that A is an initial segment of this ordering. As A (and \bar{A}) are not computable, $\leq_R|_A$ does not have a maximal element, and $\leq_R|\bar{A}$ does not have

a minimal element. As A and \bar{A} are both immune, we now have by [Cas76, Lemma 2], that $\leq_R|_A$ is of order-type ω and $\leq_R|\bar{A}$ is of order-type ω^{-1} .

Every infinitely descending chain is therefore a subset of \bar{A} . As \bar{A} is immune, these chains are not computable. \square

For the rest of this section, let A and \leq_R be as in Theorem 4.5.4, and let R denote the countdown graph $(\mathbb{N}, >_R)$.

Example 4.5.5. Let $(\mathcal{N}, \leq_{\mathcal{N}}), (\mathcal{N}', \leq_{\mathcal{N}'})$ be computably related systems of ordinal notations. Then we have

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

Proof: For all $u \in \mathcal{N}$, define $M_u = \{\tau \mid \tau \diamond \bar{u} \text{ is an } \mathcal{N}\text{-path}\}$. Clearly, for all $u \in \mathcal{N}$, $M_u \neq \emptyset$.

Proof of (a). Suppose $\mathcal{N} \leq_{CD} \mathcal{N}'$ as witnessed by k . Obviously, it suffices to show the following claim.

Claim: $\forall u \in \mathcal{N}, \forall \tau \in M_u : \nu_{\mathcal{N}}(u) \leq \nu_{\mathcal{N}'}(k(\tau \diamond \bar{u}))$.

Proof of Claim. We prove the claim by transfinite induction on $\nu_{\mathcal{N}}(u)$ for $u \in \mathcal{N}$. The base case is trivial. Suppose $u \in \mathcal{N}$ is such that $\nu_{\mathcal{N}}(u) > 0$ and the claim holds for all $v <_{\mathcal{N}} u$. Let $\tau \in M_u$. For all $v <_{\mathcal{N}} u$ we now have

$$\nu_{\mathcal{N}}(v) \underset{(IH)}{\leq} \nu_{\mathcal{N}'}(k(\tau \diamond \bar{u} \diamond \bar{v})) < \nu_{\mathcal{N}'}(k(\tau \diamond \bar{u})). \quad (4.66)$$

Thus,

$$\nu_{\mathcal{N}}(u) = \sup_{v <_{\mathcal{N}} u} (\nu_{\mathcal{N}}(v) + 1) \underset{(4.66)}{\leq} \nu_{\mathcal{N}'}(k(\tau \diamond \bar{u})). \quad (4.67)$$

□ (FOR CLAIM)

□ (FOR (a))

Proof of (b). “ \Rightarrow ”: Suppose $\mathcal{N} \leq_{CD} R$ as witnessed by $k \in \mathcal{R}$. Suppose, by way of contradiction, \mathcal{N} gives a notation to all ordinals $< \omega \cdot 2$. Let w be a notation in S for ω . It is straightforward that, for all $\tau \in M_w$, $k(\tau \diamond \bar{w}) \in \bar{A}$. We have that $M = \{\tau^- \mid \forall i < \#elets(\tau) - 1 : \tau(i+1) \text{ is predecessor of } \tau(i) \wedge \text{last}(\tau) \text{ is a notation for a limit-ordinal}\}$ is a ce subset of M_w . Hence, $T = \bigcup_{\tau \in M} \text{range}(\tau)$ is a ce subset of \bar{A} . As \bar{A} is immune, T is finite. Let n be the cardinality of T . Let $\tau \in M$ be a sequence of length $n+1$. Hence, $\{k(\tau[i+1]) \mid i \leq n\}$ is a subset of T of size $n+1$, a contradiction.

“ \Leftarrow ”: Let $i \in \{0, 1\}$, $j \in \mathbb{N}$ and \mathcal{N} systems of ordinal notations having notations for all and only the ordinals $< \omega \cdot i + j$. It is easy to see from the definition of a system of ordinal notations, that there is a computable function $f \in \mathcal{R}$ such that $\forall u \in \mathcal{N} : f(u) = \langle a, b \rangle \Leftrightarrow \nu_{\mathcal{N}}(u) = \omega \cdot a + b$.

Let r be a finite sequence strictly increasing with respect to \leq_R in \bar{A} of length $\max(j, 1)$. As \mathcal{N} computably related, there exists $k \in \mathcal{R}$ such that

$$\forall \tau \in \vec{\mathcal{N}} : k(\tau) = \begin{cases} r(b), & \text{if } f(\tau(\#elets(\tau) - 1)) = \langle 1, b \rangle \text{ for some } b \in \mathbb{N}; \\ \rho_0(\#elets(\tau_1)), & \text{otherwise, with } \tau = \tau_0 \diamond \bar{v} \diamond \tau_1, \text{ where } \tau_0 \text{ does not} \\ & \text{contain any notation for a finite ordinal, } f(v) = \\ & \langle 0, b' \rangle \text{ and } \rho_0 = \mu \rho \in \vec{R}. \#elets(\rho) = b' + 1 \wedge \forall i \leq b' : \\ & \rho(i) <_R r(0). \end{cases}$$

□ (FOR (b))

Proof of (c). Suppose, by way of contradiction, otherwise, as witnessed by k . Let r be an infinite decreasing sequence in \leq_R . Then $\lambda i. k(r[i+1])$ is an infinite strictly decreasing sequence in \mathcal{N} , a contradiction. □ (FOR (c))

Next we give one of our main theorems for this chapter, characterizing the dependence of the learning power of postdictively complete learning on the countdown graph used in terms of \leq_{CD} .

Theorem 4.5.6. Let $G \in \mathcal{G}_{\text{comp}}$, $G' \in \mathcal{G}$. We have

$$\mathcal{TPcp}_G \mathbf{GEx} \subseteq \mathcal{TPcp}_{G'} \mathbf{GEx} \Leftrightarrow G \leq_{CD} G'.$$

Proof: “ \Leftarrow ”: Suppose $G \leq_{CD} G'$ as witnessed by $k \in \mathcal{R}$. Let $\mathcal{S} \in \mathcal{TPcp}_G \mathbf{GEx}$ as witnessed by $\langle h, f \rangle$. We now set f' such that $\langle h, f' \rangle$ witnesses $\mathcal{S} \in \mathcal{TPcp}_{G'} \mathbf{GEx}$.

$$\begin{aligned} \forall \sigma \forall l < \# \text{elets}(\sigma) : f'_l(\sigma) &= k(\text{row}(l, f, \sigma)); \\ \forall \sigma : f'(\sigma) &= \langle f'_0(\sigma), \dots, f'_{\# \text{elets}(\sigma)-1}(\sigma) \rangle. \end{aligned}$$

“ \Rightarrow ”: Suppose $\mathcal{TPcp}_G \mathbf{GEx} \subseteq \mathcal{TPcp}_{G'} \mathbf{GEx}$. Let

$$f_0(\emptyset) = \langle \rangle; \tag{4.68}$$

$$\forall \sigma, e, m, t : f_0(\sigma \diamond \overline{\langle e, m, t \rangle}) = m; \tag{4.69}$$

and

$$h_0(\emptyset) = 0 \tag{4.70}$$

$$\forall \sigma, e, m, t : h_0(\sigma \diamond \overline{\langle e, m, t \rangle}) = \begin{cases} e, & \text{if } \sigma \neq \emptyset \wedge \forall x \in \perp_G(f_0, \sigma) : \varphi_e(x) \downarrow_t = \sigma(x); \\ \text{patch}_0(\sigma), & \text{otherwise.} \end{cases} \tag{4.71}$$

Obviously, $\langle h_0, f_0 \rangle \in \mathcal{TPcp}_G$. Let

$$\mathcal{S} = \mathbf{GEx}(h_0). \tag{4.72}$$

Hence, $\mathcal{S} \in \mathcal{TPcp}_G \mathbf{GEx}$; therefore, $\mathcal{S} \in \mathcal{TPcp}_{G'} \mathbf{GEx}$. Let $\langle h, f \rangle$ be such that

$$\langle h, f \rangle \text{ witnesses } \mathcal{S} \in \mathcal{TPcp}_{G'} \mathbf{GEx}. \tag{4.73}$$

In particular, we have now

$$\langle h, f \rangle \in \mathcal{R}. \tag{4.74}$$

For each $e \in \mathbb{N}$, define

$$\mathcal{S}_e = \{g \in \mathcal{S} \mid \forall i : \pi_1(\pi_1(g(i))) = e\}. \quad (4.75)$$

Note that, by a simple recursion theoretic argument,

$$[\mathcal{S}] = \text{Seq}. \quad (4.76)$$

We set $\max(\emptyset) = 0$. Define, for each $j \in \{0, 1\}$, $e \in \mathbb{N}$ and $\sigma, \tau \in \text{Seq}$,

$$r_j(e, \sigma, \tau) = \langle \lambda i < \#\text{elets}(\tau) \cdot \langle \langle e, (\tau(i))^{\#\text{elets}(\sigma)+i} \rangle, \max_{x < \#\text{elets}(\sigma)} (\Phi_e(x)) + j \rangle \rangle_{\text{Seq}}; \quad (4.77)$$

Obviously, the i -th component of $r_j(e, \sigma, \tau)$ does not depend on any components of τ besides the i -th. Furthermore, note that for all $j \in \{0, 1\}$, $e \in \mathbb{N}$, $\sigma \in [\mathcal{S}_e]$ and $\tau \in \vec{G}$,

$$(\sigma \subseteq \varphi_e \wedge r_j(e, \sigma, \tau) \downarrow) \Rightarrow \sigma \diamond r_j(e, \sigma, \tau) \in [\mathcal{S}_e]. \quad (4.78)$$

$$t(e, \sigma) = \begin{cases} & \text{if } \langle j, \tau \rangle \text{ is first number found in a dove-} \\ r_j(e, \sigma, \tau), & \text{tailing search such that } j \in \{0, 1\}, \tau \in \vec{G} \\ & \text{and } h(\sigma) \neq h(\sigma \diamond r_j(e, \sigma, \tau) \downarrow); \\ \uparrow, & \text{if no such } \langle j, \tau \rangle \text{ is found.} \end{cases} \quad (4.79)$$

Claim: $(\exists e \in \mathbb{N} \exists \sigma \in [\mathcal{S}_e] : \varphi_e = \sigma \wedge t(e, \sigma) \uparrow) \Rightarrow G \leq_{CD} G'$.

Suppose $e \in \mathbb{N}$ and $\sigma \in [\mathcal{S}_e]$ are such that

$$\varphi_e = \sigma \wedge t(e, \sigma) \uparrow. \quad (4.80)$$

Therefore, we have for all $\tau \in \vec{G}$ and $j \in \{0, 1\}$, by $\tau, \tau' \neq \emptyset$, (4.77) and the first conjunct of (4.80),

$$r_j(e, \sigma, \tau) \downarrow. \quad (4.81)$$

Note that we have now, for all $\tau, \tau' \in \vec{G}$, by (4.77) and (4.81),

$$r_0(e, \sigma, \tau)(0) \downarrow \neq r_1(e, \sigma, \tau')(0) \downarrow. \quad (4.82)$$

By (4.79), the second conjunct of (4.80) and (4.81) we have, for all $\tau \in \vec{G}$ and $j \in \{0, 1\}$,

$$h(\sigma) = h(\sigma \diamond r_j(e, \sigma, \tau)). \quad (4.83)$$

Obviously, if $\varphi_{h(\sigma)}(\#\text{elets}(\sigma))$ is defined, it can be at most one element in the set $\{r_j(e, \sigma, \tau)(0) \mid j \in \{0, 1\}, \tau \in \vec{G}\}$. Hence, with (4.82), we can (possibly not constructively) fix $j \in \{0, 1\}$ such that

$$\varphi_{h(\sigma)}(\#\text{elets}(\sigma)) \notin \{r_j(e, \sigma, \tau)(0) \mid \tau \in \vec{G}\}. \quad (4.84)$$

By (4.73), (4.83) and (4.84) we have

$$\forall \tau \in \vec{G} : \#\text{elets}(\sigma) \notin \perp_{G'}(f, \sigma \diamond r_j(e, \sigma, \tau)). \quad (4.85)$$

By (4.11), this is equivalent to

$$\forall \tau \in \vec{G} : \text{row}(\#\text{elets}(\sigma), f, \sigma \diamond r_j(e, \sigma, \tau)) \in \vec{G}'. \quad (4.86)$$

We define $k \in \mathcal{R}$ such that, for all $\tau \in \mathbb{N}$,

$$k(\tau) = \begin{cases} f(\sigma \diamond r_j(e, \sigma, \tau))(\#\text{elets}(\sigma)), & \text{if } \tau \in \vec{G}; \\ 0, & \text{otherwise.} \end{cases} \quad (4.87)$$

or all $\tau \in \vec{G}$, since for all $i < \#\text{elets}(\tau)$, $\tau[i+1] \in \vec{G}$, we have

$$\begin{aligned} & \lambda i < \#\text{elets}(\sigma).k(\tau[i+1]) \\ & \stackrel{(4.87)}{=} \lambda i < \#\text{elets}(\sigma).f(\sigma \diamond r_j(e, \sigma, \tau[i+1]))(\#\text{elets}(\sigma)) \\ & \stackrel{(4.77)}{=} \lambda i < \#\text{elets}(\sigma).f(\sigma \diamond (r_j(e, \sigma, \tau)[i+1]))(\#\text{elets}(\sigma)) \\ & \stackrel{(4.10)}{=} \text{row}(\#\text{elets}(\sigma), f, \sigma \diamond r_j(e, \sigma, \tau)). \end{aligned} \quad (4.88)$$

(4.86), (4.88) and Proposition 4.5.2 show $G \leq_{CD} G'$. □ (FOR CLAIM)

Suppose, by way of contradiction, $G \not\leq_{CD} G'$. Hence, by the claim,

$$\forall e \forall \sigma \in [\mathcal{S}_e] : \varphi_e = \sigma \Rightarrow t(e, \sigma) \downarrow. \quad (4.89)$$

We apply Lemma 4.2.5 with $t_0 = t_1 = t$, $\mathcal{C} = \mathcal{TPcp}_{G'}$ and $\delta = \mathbf{Ex}$. (i) is trivial from the definitions. (ii)(a) follows with (4.74) and (4.89). (ii)(b), (ii)(c) and (ii)(d) are straight from (4.78) and (4.79). Furthermore, by (4.79), we get directly $t_0(e, \sigma) \downarrow \Rightarrow h(\sigma) \neq h(\sigma \diamond t_0(e, \sigma))$; hence, the antecedent of (ii)(e) is false.

Therefore, $\mathcal{S} \notin \mathcal{TPcp}_{G'} \mathbf{GEx}$, a contradiction. \square

Next are three corollaries to Theorem 4.5.6 (or its proof). The first two are regarding the other restricted learnability notions of the present paper. The third is our hierarchy theorem for ordinal notations.

First, we observe that the set \mathcal{S} as in (4.72) in the proof of Theorem 4.5.6 does depend only on G , not on G' . Therefore, we can give the following strong corollary.

Corollary 4.5.7. Let $G \in \mathcal{G}_{\text{comp}}$. We have

$$\mathcal{TPcp}_G \mathbf{GEx} \setminus \bigcup_{\substack{G' \in \mathcal{G}_{\text{comp}} \\ G \not\prec_{CD} G'}} \mathbf{GPcs}_{G'} \mathbf{Ex} \neq \emptyset.$$

Proof: Let \mathcal{S} be as given in (4.72) in the proof of Theorem 4.5.6. Let $G' \in \mathcal{G}_{\text{comp}}$ such that $G \not\prec_{CD} G'$. The proof of Theorem 4.5.6 showed $\mathcal{S} \notin \mathcal{TPcp}_{G'} \mathbf{GEx}$. Since $[\mathcal{S}] = \text{Seq}$ by (4.76) and $\mathcal{TPcp}_{G'} \mathbf{GEx} \stackrel{(4.30)}{=} \mathcal{TPcs}_{G'} \mathbf{GEx}$, we have, by the contrapositive of (4.14) given in Remark 4.2.4,

$$\mathcal{S} \notin \mathbf{GPcs}_{G'} \mathbf{Ex}.$$

\square

Next is a characterization of the graph dependence of relative learning power for the restricted learning criteria not covered by Theorem 4.5.6.

Corollary 4.5.8. For all $G, G' \in \mathcal{G}_{\text{comp}}$ we have

$$G \leq_{CD} G' \Leftrightarrow \mathcal{RGPcp}_G \mathbf{Ex} \subseteq \mathcal{RGPcp}_{G'} \mathbf{Ex} \quad (4.90)$$

$$\Leftrightarrow \mathcal{RGPcs}_G \mathbf{Ex} \subseteq \mathcal{RGPcs}_{G'} \mathbf{Ex} \quad (4.91)$$

$$\Leftrightarrow \mathbf{GPcp}_G \mathbf{Ex} \subseteq \mathbf{GPcp}_{G'} \mathbf{Ex} \quad (4.92)$$

$$\Leftrightarrow \mathbf{GPcs}_G \mathbf{Ex} \subseteq \mathbf{GPcs}_{G'} \mathbf{Ex}. \quad (4.93)$$

Proof: Obviously, all right-hand-sides are implied by $G \leq_{CD} G'$, just as in the proof of “ \Leftarrow ” of Theorem 4.5.6. By Corollary 4.5.7, $G \not\leq_{CD} G'$ implies the negation of each right-hand-side. \square

Recall that, from Definition 4.2.1, for a graph $G \in \mathcal{G}$ and $m \in G$, we ambiguously use m to refer to the countdown-graph $\{n \in G \mid m \rightarrow^+ n\}$. For two sets M, N we write $M \# N$ iff $(M \not\subseteq N \wedge N \not\subseteq M)$.

Corollary 4.5.9. Let $(\mathcal{N}, \leq_{\mathcal{N}})$ be a computably related system of ordinal notations. Let $u, v \in \mathcal{N}$. Then we have

$$u <_{\mathcal{N}} v \Leftrightarrow u <_{CD} v \quad (4.94)$$

$$\Leftrightarrow \mathcal{TPcp}_u \mathbf{GEx} \subset \mathcal{TPcp}_v \mathbf{GEx}. \quad (4.95)$$

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

$$\mathcal{TPcp}_{\mathcal{N}} \mathbf{GEx} \# \mathcal{TPcp}_{\mathcal{R}} \mathbf{GEx}. \quad (4.96)$$

Proof of (4.94). “ \Rightarrow ”: Suppose $u <_{\mathcal{N}} v$. Hence, considering u and v as graphs, we have $u \subset v$. Then we have $u \leq_{CD} v$ is witnessed by any $k \in \mathcal{R}$ such that $\forall \tau \in \vec{\mathcal{N}} : k(\tau) = \text{last}(\tau)$, so that Theorem 4.5.6 applies. By Example 4.5.5(a), $v \not\leq_{CD} u$

“ \Leftarrow ”: This follows directly from Example 4.5.5(a).

Proof of (4.95). By Theorem 4.5.6.

Proof of (4.96). This follows directly from Theorem 4.5.6 and Example 4.5.5. \square

4.6 Characterization of Reliable Identification

Theorem 4.6.1 just below answers a question of Thomas Zeugmann [Zeu08] regarding the possibility of characterizing reliable learning²⁵ by suitable countdown graph delayed postdictively complete learning.

Let \mathcal{G}_{wf} be the set of all *well-founded* countdown graphs, i.e., the set of all countdown graphs without *any* (not just computable) infinite paths. Recall that \mathcal{R} and \mathfrak{R} are, *respectively*, the sets of: all computable and all total functions, each $\mathbb{N} \rightarrow \mathbb{N}$.

Theorem 4.6.1.

$$\bigcup_{G \in \mathcal{G}} \mathcal{TPcp}_G \mathbf{GEx} = \text{Rel}_{\mathcal{R}} \mathbf{GEx}; \quad (4.97)$$

$$\bigcup_{G \in \mathcal{G}_{\text{wf}}} \mathcal{TPcp}_G \mathbf{GEx} = \text{Rel}_{\mathfrak{R}} \mathbf{GEx}. \quad (4.98)$$

Proof: “ \subseteq ”: easy for both (4.97) and (4.98).

“ \supseteq ”: We will show (4.97) and (4.98) simultaneously. Let $\mathcal{S} \in \text{Rel}_{\mathcal{R}} \mathbf{GEx}$ as witnessed by $\langle h, f \rangle$. Define (G, \rightarrow_G) to be as follows.

$$G = \text{Seq}; \quad (4.99)$$

$$\forall \sigma, x : \sigma \rightarrow_G \sigma \diamond \bar{x} \Leftrightarrow [h(\sigma) = h(\sigma \diamond \bar{x}) \wedge \neg(\sigma \sqsubseteq \varphi_{h(\sigma \diamond \bar{x})})]. \quad (4.100)$$

The following claim and its proof show simultaneously that (G, \rightarrow_G) is a countdown graph (as h is reliable on \mathcal{R}), and that (G, \rightarrow_G) is a well-founded countdown graph if h is reliable on \mathfrak{R} .

Claim 1: (G, \rightarrow_G) is a countdown graph, and, if h is reliable on \mathfrak{R} , also well-founded. Suppose, by way of contradiction, G has an infinite computable (respectively, arbitrary) path. Any infinite path will start with some finite sequence σ and then,

²⁵ See Definition 3.2.5.

in each step, add one more element to the end of σ . Hence, as there exists a computable (respectively, arbitrary) such path, there is $g \in \mathcal{R}$ (respectively, $g \in \mathfrak{A}$) and n_0 , such that $(g[i])_{i \geq n_0}$ is an infinite path in G .

In particular, $\forall i \geq n_0 : g[i] \rightarrow_G g[i+1]$; hence, $\forall i \geq n_0 : h(g[i]) = h(g[i+1])$. As h is reliable on \mathcal{R} (respectively, on \mathfrak{A}), we have $\varphi_{h(g[n_0+1])} = g \supseteq g[n_0]$, a contradiction to the second conjunct in the definition of $g[n_0] \rightarrow_G g[n_0+1]$.

□ (FOR CLAIM 1)

There are $h', f' \in \mathcal{R}$, and, using 1-1 s-m-n, there is a 1-1 $p \in \mathcal{R}$, such that

$$\forall e, \sigma, x : \varphi_{p(e, \sigma)}(x) = \begin{cases} \sigma(x), & \text{if } x < \#\text{elets}(\sigma); \\ \varphi_e(x), & \text{otherwise;} \end{cases} \quad (4.101)$$

$$\forall \sigma : h'(\sigma) = \begin{cases} p(h(\sigma), \emptyset), & \text{if } \sigma = \emptyset; \\ h'(\sigma^-), & \text{else if } h(\sigma) = h(\sigma^-); \\ p(h(\sigma), \sigma), & \text{otherwise;} \end{cases} \quad (4.102)$$

$$\forall \sigma : f'(\sigma) = \sigma^{\#\text{elets}(\sigma)}. \quad (4.103)$$

Obviously, $\langle h', f' \rangle$ **GEx**-learns \mathcal{S} . To show that $\langle h', f' \rangle$ is postdictively complete with respect to G , it suffices to show that $\forall \sigma, x, \tau : (\sigma \not\vdash_G \sigma \diamond \bar{x} \Rightarrow \sigma \subseteq \varphi_{h'(\sigma \diamond \bar{x} \diamond \tau)})$. Let σ, x be such that $\sigma \not\vdash_G \sigma \diamond \bar{x}$, let τ be arbitrary. Define $\rho = \sigma \diamond \bar{x} \diamond \tau$ and let n_0 be minimal such that $(\forall i | n_0 \leq i < \#\text{elets}(\rho)) : h'(\rho[i]) = h'(\rho)$. Then

$$h'(\rho) = p(h(\rho[n_0]), \rho[n_0]). \quad (4.104)$$

Case 1: $n_0 \geq \#\text{elets}(\sigma)$.

Then $\sigma \subseteq \rho[n_0] \subseteq \varphi_{p(h(\rho[n_0]), \rho[n_0])} \stackrel{(4.104)}{=} \varphi_{h'(\rho)}$.

Case 2: $n_0 < \#\text{elets}(\sigma)$.

From (4.102) and p 1-1, we have that

$$\forall \sigma \neq \emptyset : h(\sigma) \neq h(\sigma^-) \Leftrightarrow h'(\sigma) \neq h'(\sigma^-). \quad (4.105)$$

Then $h(\sigma) = h(\sigma \diamond \bar{x})$ and thus, as $\sigma \not\vdash_G \sigma \diamond \bar{x}$,

$$\sigma \subseteq \varphi_{h(\sigma \diamond \bar{x})}. \quad (4.106)$$

By (4.105) we have $h(\sigma \diamond \bar{x}) = h(\rho[n_0])$; therefore,

$$\sigma \stackrel{(4.106)}{\subseteq} \varphi_{h(\sigma \diamond \bar{x})} = \varphi_{h(\rho[n_0])}; \quad (4.107)$$

hence,

$$\sigma \subseteq \varphi_{p(h(\rho[n_0]), \rho[n_0])} \stackrel{(4.104)}{=} \varphi_{h'(\rho)}. \quad (4.108)$$

□

It is well known that $\text{Rel}_{\mathcal{R}}\mathbf{GEx}$ and $\text{Rel}_{\mathfrak{R}}\mathbf{GEx}$ separate (see [SZ02] for a discussion). This allows us to conclude the following nice corollary, stating a trade-off between learning power and well-foundedness of graphs used: There are classes learnable postdictively completely with non-well-founded countdown graphs only.

Corollary 4.6.2.

$$\bigcup_{G \in \mathcal{G}_{\text{wf}}} \mathcal{TPcp}_G \mathbf{GEx} \subset \bigcup_{G \in \mathcal{G}} \mathcal{TPcp}_G \mathbf{GEx}. \quad (4.109)$$

The proof of “ \supseteq ” of Theorem 4.6.1 above uses a Π_1 -graph as given in (4.99) and (4.100).²⁶ Therefore, we can conclude that no additional learning power is gained by using graphs computationally more complex than Π_1 .

Let \mathcal{G}_{Π_1} and $\mathcal{G}_{\text{wf}, \Pi_1}$, respectively, be the set of all and all well-founded Π_1 -graphs.

Corollary 4.6.3.

$$\bigcup_{G \in \mathcal{G}} \mathcal{TPcp}_G \mathbf{GEx} = \bigcup_{G \in \mathcal{G}_{\Pi_1}} \mathcal{TPcp}_G \mathbf{GEx}; \quad (4.110)$$

$$\bigcup_{G \in \mathcal{G}_{\text{wf}}} \mathcal{TPcp}_G \mathbf{GEx} = \bigcup_{G \in \mathcal{G}_{\text{wf}, \Pi_1}} \mathcal{TPcp}_G \mathbf{GEx}. \quad (4.111)$$

²⁶ A set is Π_1 iff the set is the complement of a **ce** set [Rog67, § 14.1]. A graph is Π_1 iff both its set of vertices and its set of edges is Π_1 .

Chapter 5

DIFFICULTIES IN FORCING FAIRNESS

Yoshinaka [Yos09] claims that, for efficient learning in the limit from positive data, the combination of *postdictive completeness*, *conservativeness* and *prudence*¹ is restrictive enough to prevent all Pitt-style postponement tricks.²

In the present chapter, 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 See Example 3.2.4.

² See Section 1.3.

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 formally introduced in Example 3.2.1 above 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 5.1.5 to cover *just* his particular hypothesis spaces.

unfairness (Theorem 5.1.5 in Section 5.1 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 3.1.1 above.

In all of our settings, any combination of just the two restrictions of conservativeness and prudence allows for *arbitrary* postponement tricks (Theorem 5.3.1 in Section 5.3).

In each of our three 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 postponement tricks.⁶

The three residual settings are: 1. TxtEx-learning with *certain other* uniformly decidable hypothesis spaces (Section 5.2), e.g., the (efficiently coded), explicitly clocked, multi-tape Turing Machines which halt in linear time [RC94, Chapter 6],⁷ 2. TxtEx-learning with a general purpose hypothesis space (Section 5.3) and

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 postponement 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 postponement tricks than postdictive completeness alone. Again: we already know, though, that all three restrictions do *not* avoid *all* postponement tricks.

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

3. function learning with a general purpose hypothesis space.

The theorems that postdictive completeness forbids *some* postponement tricks in these last three settings are Theorems 5.2.1, 5.3.2 and 5.4.3.

The theorems that postdictive completeness does *not* forbid *all* postponement tricks in these last three settings are Theorems 5.2.2, 5.3.3 and 5.4.4.

For each of our theorems asserting some or all postponement tricks are *not* forbidden by the three restrictions, in its proof, the witnessing learner can be seen to explicitly employ postponement tricks.

Note that, in this chapter, some proofs will depend on strictly later results and their proofs. However, there is no circularity in the mathematical justification.

5.1 Learning with Uniformly Polynomial time 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 (5.1) as follows holds for U in place of V .

$$\begin{aligned} \exists s_{\cap} \in \mathcal{R} : \forall e, D : V_{s_{\cap}(e,D)} &= V_e \cap D \wedge \\ \exists s_{\cup} \in \mathcal{R} : \forall e, D : V_{s_{\cup}(e,D)} &= V_e \cup D \wedge \\ \exists s_{\setminus} \in \mathcal{R} : \forall e, D : V_{s_{\setminus}(e,D)} &= V_e \setminus D. \end{aligned} \tag{5.1}$$

Note that, in practice, many effective numberings of ce languages allow s_{\cap} , s_{\cup} and s_{\setminus} as in (5.1) to be computable in polynomial or even linear time.

Codings for DFAs or CFGs with Prohibition are example such Us (see Example 3.2.1).

Interestingly, Theorem 5.1.2 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.⁸ This leads to the main theorem in this section (Theorem 5.1.5), that, for hypothesis space U , no combination of the three restrictions of postdictive completeness, conservativeness and prudence will forbid *arbitrary* postponement tricks.

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

Lemma 5.1.1. We have

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

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

Let P be the predicate such that

$$\forall D', D : P(D', D) \Leftrightarrow [D' \subseteq D \wedge h(D') \downarrow_{|D|} \wedge D \subseteq U_{h(D')}]. \quad (5.2)$$

As $\lambda x, e. x \in U_e \in \mathbf{PF}$ and by Lemma 2.2.6 (specifically items (i), (viii), (ix), (x) and (xiv)), P is computable in polynomial time.

As P and r are computable in polynomial time and by Lemma 2.2.6 (specifically items (i), (vi) and (vii)), there is $h' \in \mathbf{PF}$ such that

$$\forall D : h'(D) = \begin{cases} h(D'), & \text{if there is } \leq\text{-maximum } D' \leq |D| \text{ such that } P(D', D); \\ r(D), & \text{otherwise.} \end{cases} \quad (5.3)$$

Postdictive completeness is straightforward.

⁸ See Examples 3.2.3 and 3.2.4.

Claim: Let D'' and E be such that $P(D'', E)$. Then $h(D'') = h(E)$.

From $P(D'', E)$ we have $E \subseteq U_{h(D'')}$; hence, as h is conservative, $h(D'') = h(E)$.

□ (FOR CLAIM)

Regarding conservativeness of h' : Let $D \subset E$ with $E \subseteq U_{h'(D)}$. Then $h'(D) \neq r(D)$. Let D' be as found by Case 1 in the definition of h' . Therefore, $E \subseteq U_{h(D')}$, and, thus, $P(D', E)$. Hence, there is D'' as found by Case 1 in the definition of h' . We have $P(D'', E)$, and, therefore,

$$h'(D) = h(D') \underset{\text{Claim}}{=} h(E) \underset{\text{Claim}}{=} h(D'') = h'(E). \quad (5.4)$$

Regarding convergence to correct indices: Let $L \in \mathcal{L}$.

Case 1: L is finite.

As h is set-driven and identifies L , $L = U_{h(L)}$. By the Claim, $h'(L) \in \{r(L), h(L)\}$; hence, $L = U_{h'(L)}$.

Case 2: L is infinite.

Let $D' \subseteq L$ be such that $L = U_{h(D')}$. Since L is infinite and $L = U_{h(D')}$, we can find D with $D' \subseteq D \subseteq L$ and $P(D', D)$. For all E with $D \subseteq E \subseteq L$, since $L = U_{h(D')}$, we have $P(D', E)$, and, then, using the Claim, $h'(E) = h(D')$. □

Theorem 5.1.2. We have

$$\mathcal{TPrudPFT}(\text{PcpConv})\text{TxtSdEx}_U = \text{TxtGConvEx}_U.$$

Proof: “ \subseteq ” is trivial. Regarding “ \supseteq ”: First we apply Proposition 5.2.3 to get total conservativeness. Then we use Theorem 5.2.4 to obtain total postdictive completeness. We use Theorem 5.3.5 to make the learner set-driven. By Lemma 5.1.1, such a learner can be postponed to be computable in polynomial time. By Proposition 5.3.6, this learner is automatically totally prudent. □

Proposition 5.1.3 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 5.1.3. Let $\mathcal{D} \in \{\text{Id}, \text{Prud}\}$ and $\mathcal{D}' \in \{\text{Id}, \mathcal{T}\text{Prud}\}$. We have

$$\begin{aligned} \mathcal{D}\text{TxtPsdEx}_U &= \mathcal{D}\text{TxtGEx}_U \text{ and} \\ \mathcal{D}'\text{TxtPsdEx}_U &= \mathcal{D}'\text{TxtGEx}_U. \end{aligned}$$

Proof: “ \subseteq ” is straightforward. Let $h \in \mathcal{P}$. Let $f, h' \in \mathcal{P}$ be such that

$$\forall D, n : f(D, n) = \mu\sigma. \left[\begin{array}{l} \text{content}(\sigma) \subseteq D \wedge \\ (\forall \tau < n | \sigma \subseteq \tau, \text{content}(\tau) \subseteq D) [h(\sigma) = h(\tau)] \end{array} \right]; \quad (5.5)$$

$$\forall D, n : h'(D, n) = h(f(D, n)). \quad (5.6)$$

It is easy to see that $\mathcal{D}\text{TxtGEx}_U(h) \subseteq \mathcal{D}\text{TxtPsdEx}_U(h')$ and $\mathcal{D}'\text{TxtGEx}_U(h) \subseteq \mathcal{D}'\text{TxtPsdEx}_U(h')$ □

We can use postponement tricks on partially set-driven learning just as we did for set-driven learning in Lemma 5.1.1, resulting in Lemma 5.1.4 just below.

Lemma 5.1.4. We have

$$\begin{aligned} \text{PrudPF}\mathcal{T}\text{PcpTtxtPsdEx}_U &= \text{PrudTtxtGEx}_U \text{ and} \\ \mathcal{T}\text{PrudPF}\mathcal{T}\text{PcpTtxtPsdEx}_U &= \mathcal{T}\text{PrudTtxtGEx}_U. \end{aligned}$$

Proof: “ \subseteq ” is straightforward. We use Proposition 5.1.3, and let $h \in \mathcal{P}$ be a partially set-driven learner. Fix a program for h . Using Lemma 2.2.6 (specifically items (i), (ii), (iii), (vi), (vii), (viii), (ix), (x), (xiii) and (xiv)), there are $f, h' \in \mathbf{PF}$ such that

$$\forall D, n : f(D, n) = \begin{cases} \text{if there is } \leq\text{-maximum } D' \leq |n| \text{ s.t. } \exists i \leq |n|: \\ \langle D', i \rangle & \text{card}(D') \leq i \wedge D' \subseteq D \wedge h(D', i) \downarrow_{|n|} \wedge \\ & D \subseteq U_{h(D', i)}; \\ -1, & \text{otherwise;} \end{cases} \quad (5.7)$$

$$\forall D, n : h'(D, n) = \begin{cases} h(f(D, n)), & \text{if } f(D, \text{card}(D)) \neq -1; \\ r(D), & \text{otherwise.} \end{cases} \quad (5.8)$$

Postdictive completeness and prudence are straightforward. Convergence is as in the proof of Lemma 5.1.1. \square

The next theorem is main result of the present section. As noted in the introduction to the present chapter, it says that the three restrictions of postdictive completeness, conservativeness and prudence allow maximal *unfairness*.

Theorem 5.1.5. Let $\delta \in \{\mathcal{R}^2, \mathbf{Pcp}, \mathbf{Conv}, \mathbf{PcpConv}\}$, $\mathcal{D} \in \{\mathbf{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathbf{Id}, \mathcal{T}\mathbf{Prud}\}$. Then

$$\begin{aligned} \mathcal{D}\mathbf{PFTxtG}\delta\mathbf{Ex}_U &= \mathcal{D}\mathbf{TtxtG}\delta\mathbf{Ex}_U \text{ and} \\ \mathcal{D}'\mathbf{PFT}\delta\mathbf{TtxtGEx}_U &= \mathcal{D}'\mathcal{T}\delta\mathbf{TtxtGEx}_U. \end{aligned}$$

Proof: Use Theorem 5.1.2, as well as Theorem 5.3.1 and Lemma 5.1.4. \square

5.2 Learning with Uniformly Decidable Hypothesis Spaces

For this section, we let $V : \mathbb{N} \rightarrow \mathcal{E}$ range over effective numberings of some \mathbf{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 $\forall D : V_{r(D)} = D$.⁹

Examples of such numberings V include the classes of all linear time, polynomial time, exponential time, doubly exponential time, ... decidable languages, each represented by efficiently numerically coded programs in a suitable subrecursive programming system for deciding languages [RC94]. Note that these example classes are *not* uniformly linear time, polynomial time, exponential time, doubly exponential time, ... decidable.

For uniformly decidable hypothesis spaces, we get mixed results. We have already seen from Theorem 5.1.5 in Section 5.1 above that there are uniformly decidable hypothesis spaces where we have arbitrary postponement tricks for all combinations of postdictive completeness, conservativeness and prudence. Next is the first main 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* postponement tricks. By contrast, according to Theorem 5.3.1, any combination of just the two restrictions of conservativeness and prudence allows for *arbitrary* postponement tricks.

The proof of the following theorem makes (indirect) use of function learning results from Chapter 4, carried over to language learning using meta-theorems from Section 3.3.

Theorem 5.2.1. There *exists* a uniformly decidable numbering V such that, for each $\delta \in \{\mathcal{R}^2, \mathbf{Pcp}, \mathbf{Conv}, \mathbf{PcpConv}\}$, $\mathcal{D} \in \{\mathbf{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathbf{Id}, \mathcal{T}\mathbf{Prud}\}$,

$$\begin{aligned} \mathcal{D}\mathbf{PFTxtG}\delta\mathbf{Ex}_V \subset \mathcal{D}\mathcal{R}\mathbf{TtxtG}\delta\mathbf{Ex}_V &\Leftrightarrow \delta \subseteq \mathbf{Pcp} \text{ and} \\ \mathcal{D}'\mathbf{PFT}\delta\mathbf{TtxtGEx}_V \subset \mathcal{D}'\mathcal{T}\delta\mathbf{TtxtGEx}_V &\Leftrightarrow \delta \subseteq \mathbf{Pcp}. \end{aligned}$$

⁹ Note that, in practice, many effective numberings of some ce languages allow r to be computable in polynomial or even linear time.

Proof: Fix any uniformly decidable numbering V that has an index for each exponential time decidable set such that there is a polynomial time computable f such that $\forall e : W_{f(e)} = V_e$.

Use Theorem 5.3.1 for “ \Rightarrow ”.

Regarding “ \Leftarrow ”: By Proposition 3.3.2,

$$\mathbf{EXPF} \in \mathbf{DPFTxtG\delta Ex}_V \Rightarrow \mathbf{EXPF} \in \mathbf{DPFTxtG\delta Ex}_W. \quad (5.9)$$

Hence, by (5.64) in Section 5.3 below,

$$\mathbf{EXPF} \notin \mathbf{DPFTxtG\delta Ex}_V. \quad (5.10)$$

□

Next is our second main result of the present section (Theorem 5.2.2). 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 postponement trick on an enumeration technique [Gol67, BB75], and our result, then, entails, as advertised in the introduction above to the present chapter, that *some* postponement tricks are *not* forbidden in the setting of the present section.

Note that Theorem 4.3.1 shows how to get a result similar to Theorem 5.2.2, but with the graphs of all *polynomial time* computable functions in the setting of function learning. However, Theorem 4.3.1 does *not* trivially extend to the setting of language learning, as the canonical order of presentation was exploited in the proof Theorem 4.3.1.

Let $\theta^{\mathcal{L}^{time}}$ be an efficiently coded programming system from [RC94, Chapter 6] for \mathbf{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) $\in \mathbf{LinF}$.¹⁰ Note that $V^{\mathcal{L}time}$ does *not* satisfy the condition at the beginning of the present section on V s 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) linear time s-m-n (see Theorem 2.3.1).

Theorem 5.2.2.

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

Proof: Using linear time s-m-n in $V^{\mathcal{L}time}$, we let $s \in \mathbf{LinF}$ be 1-1 such that $\text{range}(s)$ is decidable in linear time, D is recoverable in linear time from $s(D)$ and, if D is single-valued, then $V_{s(D)}^{\mathcal{L}time}$ is the 0-extension of the partial graph given by D . Using [RC94, § 3.2.2], there is $f \in \mathbf{LinF}$ such that $\forall e, x : \varphi_e(x) \downarrow_{f(e) \cdot |x| + f(e)}$. Intuitively, for each e , coefficients of a linear time-bound for program e are efficiently computable from e . Let SV be the set of all finite, single-valued subsets of \mathbb{N} .

Let P_0 be the following predicate.

$$\forall D, \sigma : P_0(D, \sigma) \Leftrightarrow \forall \langle x, d \rangle \in \text{content}(\sigma) : \langle x, d \rangle \notin D \Rightarrow d = 0. \quad (5.11)$$

Clearly, P_0 is computable in polynomial time, and

$$\forall D \in SV, \sigma : P_0(D, \sigma) \Leftrightarrow \text{content}(\sigma) \subseteq \varphi_{s(D)}. \quad (5.12)$$

Let P_1 be the following predicate.

$$\forall e, \sigma : P_1(e, \sigma) \Leftrightarrow \forall \langle x, d \rangle \in \text{content}(\sigma) : \varphi_e(x) \downarrow_{f(e) \cdot |x| + f(e)} = d. \quad (5.13)$$

Clearly, P_1 is computable in polynomial time, and

$$(\forall e, \sigma | \text{content}(\sigma) \in SV) : P_1(e, \sigma) \Leftrightarrow \text{content}(\sigma) \subseteq \varphi_e. \quad (5.14)$$

¹⁰ In fact, $\theta^{\mathcal{L}time}$ is a mapping from \mathbb{N} to \mathcal{E} , mapping each program number to (the graph) of the function computed by that program. Hence, $\theta^{\mathcal{L}time}$ is our desired mapping $V^{\mathcal{L}time}$. We introduce this additional notation to indicate its use as an hypothesis space.

Let R be the following predicate.

$$\forall p, \sigma : R(p, \sigma) \Leftrightarrow (\exists D \in SV : p = s(D) \wedge P_0(D, \sigma)) \vee (\neg(\exists D \in SV : p = s(D)) \wedge P_1(p, \sigma)). \quad (5.15)$$

Clearly, R is computable in polynomial time, and

$$(\forall p, \sigma | \text{content}(\sigma) \in SV) : R(p, \sigma) \Leftrightarrow \text{content}(\sigma) \subseteq \varphi_p. \quad (5.16)$$

It is easy to see that the following algorithm computes in polynomial time.

```

1 function learnLin( $\sigma$ ) is
2   if  $\sigma = \emptyset$  then
3     return  $s(\emptyset)$ ;
4    $p \leftarrow s(\emptyset)$ ;
5   for  $i = 1$  to #elets( $\sigma$ ) do
6     if  $\neg R(p, \sigma[i-1])$  then
7        $k \leftarrow -1$ ;
8       for  $e = 0$  to  $|\sigma[i]|$  do
9         if  $k = -1 \wedge |\sigma[i]| \geq f(e) \wedge P(e, \sigma[i])$  then
10            $k \leftarrow e$ ;
11       if  $k \neq -1$  then
12          $p \leftarrow k$ ;
13       else
14          $p \leftarrow s(\text{content}(\sigma[i]))$ ;
15   return  $p$ ;

```

It is easy to see that the algorithm learnLin computes the function h such

that

$$\forall \sigma : h(\sigma) = \begin{cases} s(\emptyset), & \text{if } \sigma = \emptyset; \\ h(\sigma^-), & \text{else if } \text{content}(\sigma) \subseteq \varphi_{h(\sigma^-)}; \\ e_0, & \text{else if there is an } e \leq |\sigma| \text{ such that} \\ & |\sigma| \geq f(e) \text{ and } \text{content}(\sigma) \subseteq \varphi_e, \\ & \text{where } e_0 \text{ is the least such } e; \\ s(\text{content}(\sigma)), & \text{otherwise.} \end{cases} \quad (5.17)$$

Obviously, h is postdictively complete. From the second case in h , it is easy to see that h is conservative. Furthermore, each of h 's hypotheses is a program for a linear time computable function. \square

Next we present 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 V , conservative learnability implies *total* conservative learnability. It is used proving for Theorem 5.1.2 in Section 5.1.

Proposition 5.2.3. We have

$$\mathcal{T}\text{Conv}\text{TxtGEx}_V = \text{TxtGConvEx}_V.$$

Proof: “ \subseteq ” is immediate. Let $h \in \mathcal{R}$ and $\mathcal{L} = \text{TxtGConvEx}_V(h)$.¹¹ Fix any φ -program for h . There is $h' \in \mathcal{R}$ such that

$$\forall \sigma : h'(\sigma) = \begin{cases} h(\emptyset), & \text{if } \sigma = \emptyset; \\ h'(\sigma^-), & \text{else if } \text{content}(\sigma) \subseteq U_{h'(\sigma^-)}; \\ h(\sigma), & \text{otherwise.} \end{cases} \quad (5.18)$$

¹¹ Because of Theorem 5.3.1 we can assume any TxtGConvEx_V -learner to be total.

Total conservativeness is trivial. h' identifies all of \mathcal{L} , as conservativeness of h implies

$$\forall L \in \mathcal{L}, (\forall \sigma | \text{content}(\sigma) \subseteq L) : h'(\sigma) = h(\sigma). \quad (5.19)$$

□

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

Theorem 5.2.4. We have

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

Proof: “ \subseteq ” is immediate. Let $h \in \mathcal{R}$ and $\mathcal{L} = \mathcal{T}\mathbf{Conv}\mathbf{TxtGEx}_V(h)$. Fix any φ -program for h . There is $h' \in \mathcal{R}$ such that

$$\forall \sigma : h'(\sigma) = \begin{cases} h(\emptyset), & \text{if } \sigma = \emptyset; \\ h'(\sigma^-), & \text{else if } \text{content}(\sigma) \subseteq V_{h'(\sigma^-)}; \\ h(\sigma[j_0]), & \text{else if there is a } j \leq \#\text{elets}(\sigma) : \\ & (h(\sigma[j]) \downarrow_{|\sigma|} \wedge \text{content}(\sigma) \subseteq V_{h(\sigma[j])}), \\ & \text{where } j_0 \text{ is the maximum such } j; \\ r(\text{content}(\sigma)), & \text{otherwise.} \end{cases} \quad (5.20)$$

Let P be a predicate such that $\forall \sigma : P(\sigma)$ iff h' on σ is defined according to the “otherwise” case just above.

Conservativeness and postdictive completeness are straightforward.

Let T be a text for $L \in \mathcal{L}$.

Since $T \in \mathbf{Txt}(L)$ and h learns L ,

$$\exists c \text{ min: } \forall i \geq c : h(T[i]) = h(T[c]); \quad (5.21)$$

fix such a c .

Since $\forall n \exists j \geq c : |T[j]| \geq n \wedge h \in \mathcal{R}$,

$$\exists c' \geq c \text{ min:} h(T[c]) \downarrow_{T[c']}. \quad (5.22)$$

Case 1: L is finite. Since L is finite,

$$\exists c'' \geq c' : \text{content}(T[c'']) = L. \quad (5.23)$$

Let k be minimal such that

$$h'(T[c'']) = h'(T[k]). \quad (5.24)$$

Then we have, as h' postdictively complete, (5.23) and (5.24),

$$L \subseteq V_{h'(T[k])}. \quad (5.25)$$

Subcase 1.1: $P(T[k])$. Since (5.20) and $P(T[k])$,

$$V_{h'(T[k])} = \text{content}(T[k]). \quad (5.26)$$

Since (5.25) and (5.26),

$$V_{h'(T[k])} = L. \quad (5.27)$$

As h' is conservative and (5.27),

$$\forall i \geq k : h'(T[k]) = h'(T[i]) \wedge V_{h'(T[k])} = L. \quad (5.28)$$

Subcase 1.2: Not $P(T[k])$.

Then h' on $T[k]$ is defined according to either the first or the third case in (5.20) by choice of k minimal. Let $j \leq k$ be such that

$$h'(T[k]) = h(T[j]). \quad (5.29)$$

Since (5.25) and (5.29),

$$L \subseteq V_{h(T[j])}. \quad (5.30)$$

As h is conservative, h identifies L and (5.30),

$$\forall i \geq j : h(T[i]) = h(T[j]) \wedge V_{h(T[j])} = L. \quad (5.31)$$

Since (5.29) and (5.31),

$$V_{h'(T[k])} = L. \quad (5.32)$$

As h' is conservative and (5.32),

$$\forall i \geq k : h'(T[i]) = h'(T[k]) \wedge V_{h'(T[k])} = L. \quad (5.33)$$

Case 2: L is infinite. Since L is infinite,

$$\exists c'' \geq c' : \text{content}(T[c'']) \not\subseteq \text{content}(T[c']). \quad (5.34)$$

Let k be minimal such that

$$h'(T[c'']) = h'(T[k]). \quad (5.35)$$

Then we have, as h' is postdictively complete and (5.35),

$$\text{content}(T[c'']) \subseteq V_{h'(T[k])}. \quad (5.36)$$

Since (5.20),

$$P(T[k]) \Rightarrow V_{h'(T[k])} = \text{content}(T[k]). \quad (5.37)$$

Since (5.34), (5.36) and (5.37),

$$P(T[k]) \Rightarrow k > c'. \quad (5.38)$$

Since (5.20) and (5.22),

$$\forall i \geq c' : \neg P(T[i]). \quad (5.39)$$

Since (5.38) and (5.39),

$$\neg P(T[k]). \quad (5.40)$$

Then h' on $T[k]$ is defined according to either the first or the third case in (5.20).

Let $j \leq k$ be such that

$$h'(T[k]) = h(T[j]). \quad (5.41)$$

We now have that, as (5.36) and (5.41),

$$\text{content}(T[c'']) \subseteq V_{h(T[j])}. \quad (5.42)$$

Since h is conservative and c is minimal,

$$\forall i < c : \text{content}(T[c]) \not\subseteq V_{h(T[i])}. \quad (5.43)$$

Since (5.42) and (5.43),

$$j \geq c. \quad (5.44)$$

Since (5.21) and (5.44),

$$V_{h(T[j])} = L. \quad (5.45)$$

Since (5.41) and (5.45),

$$V_{h'(T[k])} = L. \quad (5.46)$$

As h' is conservative and (5.46),

$$\forall i \geq k : h'(T[i]) = h'(T[k]) \wedge V_{h'(T[k])} = L. \quad (5.47)$$

□

Finally, we mention the following straightforward result that, without loss of generality, any **TxtGEx_V**-learner can be assumed totally postdictively complete.

Proposition 5.2.5. We have

$$\mathcal{TPcp}\mathbf{TxtGEx}_V = \mathbf{TxtGEx}_V. \quad (5.48)$$

5.3 Learning of ce Sets

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

Next (and mentioned above in the introduction to the present chapter) is our first main result which says that any combination of just the two restrictions of conservativeness and prudence allows for *arbitrary* postponement tricks.

Theorem 5.3.1. Let $\delta \in \{\mathcal{R}^2, \mathbf{Conv}\}$, $\mathcal{D} \in \{\mathbf{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathbf{Id}, \mathcal{T}\mathbf{Prud}\}$. Then

$$\begin{aligned} \mathcal{D}\mathbf{P}\mathbf{F}\mathbf{T}\mathbf{x}\mathbf{t}\mathbf{G}\delta\mathbf{E}\mathbf{x}_V &= \mathcal{D}\mathbf{T}\mathbf{x}\mathbf{t}\mathbf{G}\delta\mathbf{E}\mathbf{x}_V \text{ and} \\ \mathcal{D}'\mathbf{P}\mathbf{F}\mathcal{T}\delta\mathbf{T}\mathbf{x}\mathbf{t}\mathbf{G}\mathbf{E}\mathbf{x}_V &= \mathcal{D}'\mathcal{T}\delta\mathbf{T}\mathbf{x}\mathbf{t}\mathbf{G}\mathbf{E}\mathbf{x}_V. \end{aligned}$$

Proof: “ \subseteq ” is immediate. Let $h \in \mathcal{P}$ and $\mathcal{L} \in \{\mathcal{D}\mathbf{T}\mathbf{x}\mathbf{t}\mathbf{G}\delta\mathbf{E}\mathbf{x}_V(h), \mathcal{D}'\mathcal{T}\delta\mathbf{T}\mathbf{x}\mathbf{t}\mathbf{G}\mathbf{E}\mathbf{x}_V(h)\}$. Fix a program for h . By Lemma 2.2.6 (specifically items (i), (vi), (vii) with (2.7), (x), and (xiii)), there is $h' \in \mathbf{PF}$ such that

$$\forall \sigma : h'(\sigma) = \begin{cases} h(\sigma[j]), & \text{if there is a } \leq\text{-maximal } j \leq \#\text{elets}(\sigma) \\ & \text{s.t. } h(\sigma[j]) \downarrow_{|\sigma|}; \\ h(\emptyset), & \text{otherwise.} \end{cases} \quad (5.49)$$

Convergence to correct indices is trivial. Let $\sigma \subseteq \tau$ be such that

$$(\forall \rho, \rho' | \rho \subseteq \rho' \subseteq \sigma)[h(\rho) \neq h(\rho') \Rightarrow \text{content}(\rho') \notin V_{h(\rho)}] \quad (5.50)$$

and

$$h'(\sigma) \neq h'(\tau) \quad (5.51)$$

and let $\sigma' \subseteq \sigma$ and $\tau' \subseteq \tau$ be such that

$$h'(\sigma) = h(\sigma'); \text{ and} \quad (5.52)$$

$$h'(\tau) = h(\tau'). \quad (5.53)$$

Clearly, from the definition of h' , $\sigma' \subseteq \tau'$ and we have

$$h(\sigma') = h'(\sigma) \neq h'(\tau) = h(\tau'). \quad (5.54)$$

We have, then,

$$\text{content}(\tau) \supseteq \text{content}(\tau') \underset{(5.50)\&(5.54)}{\not\subseteq} V_{h(\sigma')} \underset{(5.52)}{=} V_{h'(\sigma)}. \quad (5.55)$$

This shows conservativeness in all relevant cases.

□

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* postponement tricks iff postdictive completeness is part of the combination.

The proof of this theorem makes use of function learning results from Chapter 4, carried over to language learning using meta-theorems from Section 3.3.

Theorem 5.3.2. Let $\delta \in \{\mathcal{R}^2, \mathbf{Pcp}, \mathbf{Conv}, \mathbf{PcpConv}\}$, $\mathcal{D} \in \{\mathbf{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathbf{Id}, \mathcal{T}\mathbf{Prud}\}$. Then

$$\begin{aligned} \mathcal{D}\mathbf{PFTxtG}\delta\mathbf{Ex}_W \subset \mathcal{D}\mathcal{R}\mathbf{TxtG}\delta\mathbf{Ex}_W &\Leftrightarrow \delta \subseteq \mathbf{Pcp} \text{ and} \\ \mathcal{D}'\mathbf{PFT}\delta\mathbf{TxtG}\mathbf{Ex}_W \subset \mathcal{D}'\mathcal{T}\delta\mathbf{TxtG}\mathbf{Ex}_W &\Leftrightarrow \delta \subseteq \mathbf{Pcp}. \end{aligned}$$

Proof: Use Theorem 5.3.1 for “ \Rightarrow ”.

Regarding “ \Leftarrow ”: As $\mathbf{EXPF} \in \mathbf{UComp}$, we have by 5.4.2,

$$\mathbf{EXPF} \in \mathcal{T}\mathbf{Prud}\mathcal{T}(\mathbf{ConvPcp})\mathbf{ArbG}\mathbf{Ex}_\varphi. \quad (5.56)$$

Using Propositions 3.3.2 and 3.3.3, we see

$$\mathbf{EXPF} \in \mathcal{T}\mathbf{Prud}\mathcal{T}(\mathbf{ConvPcp})\mathbf{TxtG}\mathbf{Ex}_W. \quad (5.57)$$

On the other hand, suppose, by way of contradiction,

$$\mathbf{EXPF} \in \mathbf{PFTxtGPcp}\mathbf{Ex}_W. \quad (5.58)$$

Using linear time s-m-n in φ , there is a function $s \in \mathbf{LinF}$ such that

$$\forall e, x : \varphi_{s(e)}(x) = \pi_1(\mu(y, t) \cdot (\Phi_e(\langle x, y \rangle) \leq t)). \quad (5.59)$$

Obviously, we get, for each e such that W_e is single valued,¹²

$$\forall e : \varphi_{s(e)} = W_e. \quad (5.60)$$

Again with linear time s-m-n in W , there is $s' \in \mathbf{LinF}$ such that

$$\forall e : W_{s'(e)} = \varphi_{s(e)}. \quad (5.61)$$

Let h witness (5.58). Then $s' \circ h$ witnesses (5.58) as well and, for each $e \in \text{range}(s' \circ h)$, e is a W -index for a single-valued set by (5.61). We use Propositions 3.3.2 with W as V , φ as V' , s as f and $s' \circ h$ as h to get that $s \circ s' \circ h \in \mathbf{PF}$ witnesses

$$\mathbf{EXPF} \in \mathbf{PFTxtGPcpEx}_\varphi. \quad (5.62)$$

Using Propositions 3.3.3 and 3.3.4, we get

$$\mathbf{EXPF} \in \mathbf{PFGPcpEx}_\varphi, \quad (5.63)$$

a contradiction to Theorem 4.3.2. Therefore,

$$\mathbf{EXPF} \notin \mathbf{PFTxtGPcpEx}_W. \quad (5.64)$$

□

Next is our second main result for this section. As noted above in the introduction to the present chapter, this theorem says that, in the *general* setting of the present section, postdictive completeness does *not* forbid *all* postponement tricks.

Theorem 5.3.3. We have

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

¹² Recall that, for each e , $W_e = \text{dom}(\varphi_e) = \text{dom}(\Phi_e)$.

Proof: Recall the effective numbering $V^{\mathcal{L}time}$ defined just before Theorem 5.2.2. By linear time s-m-n in W , there is a function $s \in \mathbf{LinF}$ such that

$$\forall e : W_{s(e)} = \{\langle x, y \rangle \mid \langle x, y \rangle \in V_e^{\mathcal{L}time}\},^{13} \quad (5.65)$$

i.e.,

$$\forall e : W_{s(e)} = V_e^{\mathcal{L}time}. \quad (5.66)$$

Thus, Theorem 5.2.2 above, with the help of Proposition 3.3.2,¹⁴ shows the claim. \square

The following theorem implies the remarkable result that every set of languages identifiable totally prudently, totally postdictively completely and totally consistently is necessarily a subset of a uniformly decidable set of languages.

Theorem 5.3.4. We have

$$\mathcal{TPrudT}(\mathbf{PcpConv})\mathbf{TxtGEx} \subset \{L \mid \exists L' \in \mathbf{UDec} : L \subseteq L'\}.$$

Proof: $\mathbf{UDec} \not\subseteq \mathbf{TxtGEx}$ is well-known [Gol67]. Let $h \in \mathcal{R}$ and $\mathcal{L} = \mathcal{TPrudT}(\mathbf{PcpConv})\mathbf{TxtGEx}(h)$. For all $x, e \in \mathbb{N}$,

$$\exists \sigma : [h(\sigma) = e \wedge h(\sigma \diamond \bar{x}) = e] \Rightarrow x \in W_e; \text{ and} \quad (5.67)$$

$$\exists \sigma : [h(\sigma) = e \wedge h(\sigma \diamond \bar{x}) \neq e] \Rightarrow x \notin W_e. \quad (5.68)$$

Hence, there is an algorithm computing $\lambda x, i. x \in W_{h(i)}$ by checking for the left-hand sides of (5.67) and (5.68) in parallel. \square

Theorem 5.3.5 just below says that certain kinds 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 5.1.2.

¹³ Recall that $\langle \cdot, \cdot \rangle$ is onto from Section 2.1.

¹⁴ $s \in \mathbf{LinF}$ helps preserving polynomial time learnability.

Theorem 5.3.5. Let V be such that (5.1) holds for V . We have

$$\mathbf{TxtSdPcpConvEx}_V = \mathbf{TxtGPcpConvEx}_V; \quad (5.69)$$

$$\mathcal{R}\mathbf{TxtSdPcpConvEx}_V = \mathcal{R}\mathbf{TxtGPcpConvEx}_V; \quad (5.70)$$

$$\mathcal{T}(\mathbf{PcpConv})\mathbf{TxtSdEx}_V = \mathcal{T}(\mathbf{PcpConv})\mathbf{TxtGEx}_V. \quad (5.71)$$

Proof: “ \subseteq ” are immediate. We prove “ \supseteq ” for (5.70); the proofs for the other two inclusions are slight modifications of the proof we give just below.

Let $h \in \mathcal{R}$, let $\mathcal{L} = \mathbf{TxtGPcpConvEx}_V(h)$. Let

$$A = \{\sigma \in \text{Seq} \mid \text{card}(\text{content}(\sigma)) = \#\text{elets}(\sigma)\}; \quad (5.72)$$

$$B = \{\sigma \in A \mid \sigma = \emptyset \vee h(\sigma) \neq h(\sigma^-)\}. \quad (5.73)$$

Intuitively, A is the set of all 1-1 sequences without pauses, and B is the set of all 1-1 sequences without pauses where h outputs a new hypothesis. Note that B is decidable. For all D , let

$$B_D = \{\sigma \in B \mid \begin{array}{l} \text{range}(\sigma) \subseteq D \wedge \\ (\forall \tau \in A \mid \sigma \subseteq \tau \wedge \text{range}(\tau) \subseteq D) h(\sigma) = h(\tau) \end{array}\}. \quad (5.74)$$

Note that $\lambda D. B_D$ is computable.

We fix an arbitrary computable total ordering $<$ on B such that, for any two sequences σ, τ ,

$$\max(\text{content}(\sigma)) < \max(\text{content}(\tau)) \Rightarrow \sigma < \tau. \quad (5.75)$$

Note that $<$ is a well-ordering, i.e., every non-empty set of sequences has a $<$ -minimal element.

Claim 1: For all D , $B_D \neq \emptyset$.

Let D be a finite set. Let $C = \{\sigma \in M \mid \text{content}(\sigma) \subseteq D\}$. Note that $\emptyset \in C$ and, for all $\sigma \in C$, $\#\text{elets}(\sigma) \leq \text{card}(D)$. Fix $\sigma \in C$ of maximum length. We show that $\sigma \in B_D$. Immediately we have $\sigma \in A$ and $\text{content}(\sigma) \subseteq D$. Let $\tau \in A$ be such that

$\sigma \subseteq \tau$ and $\text{range}(\tau) \subseteq D$. Further suppose, by way of contradiction, τ is of minimum length such that $h(\tau) \neq h(\sigma)$ (then $\#\text{elets}(\tau) > \#\text{elets}(\sigma)$). As τ is of minimum length with that property, we have $h(\tau) \neq h(\tau^-)$. Hence, $\tau \in C$, a contradiction to σ of maximal length in C . \square (FOR CLAIM)

There are p, s and $h' \in \mathcal{R}$ such that¹⁵

$$\forall D : p(D) = h(\min_{<} B_D); \quad (5.76)$$

$$\forall e : V_{s(e, E)} = (V_e \setminus \{x \mid \exists y \in E : x \leq y\}) \cup E; \quad (5.77)$$

$$\forall D : h'(D) = s(p(D), \{x \in D \mid \exists y \in \text{content}(\min_{<} B_D) : x \leq y\}). \quad (5.78)$$

Claim 2: Let $L \in \mathcal{L}$, let $x \in L$ and $D \subseteq L$ be such that $x \in V_{h'(D)}$. Then $h'(D) = h'(D \cup \{x\})$.

Trivial if $x \in D$. Suppose $x \notin D$. Let $\sigma = \min_{<} B_D$ and $e = h(\sigma)$. Let E be such that $h'(D) = s(e, E)$. As $x \notin D$, $x \notin E$, and, thus, $x \in V_e$ and

$$\forall y \in \text{content}(\sigma) : x > y. \quad (5.79)$$

As $x \in V_e$ and h is conservative on \mathcal{L} , h cannot change its hypothesis when seeing x ; hence, $\sigma \in B_{D \cup \{x\}}$. Let $D' = \text{content}(\sigma)$. By (5.79), (5.75) gives

$$(\forall \tau, \tau' \mid \tau \in B_{D'} \wedge \tau' \in B_{D \cup \{x\}} \setminus B_D) \tau < \tau'. \quad (5.80)$$

Thus, $\sigma = \min_{<} B_{D \cup \{x\}}$ and $h'(D) = h'(D \cup \{x\})$. \square (FOR CLAIM 2)

Claim 3: Let $L \in \mathcal{L}$, and let T be a text for L . Then h' converges on T to an e .

Let

$$C = \{\tau \in B \mid \begin{array}{l} \text{content}(\tau) \subseteq L \wedge \\ (\forall \tau' \in A \mid \text{content}(\tau') \subseteq L \wedge \tau \subseteq \tau') h(\tau) = h(\tau') \end{array}\}. \quad (5.81)$$

¹⁵ We use (5.1), plus some simple operations on canonical indices for finite sets, to obtain s .

Claim 3.1: $C \neq \emptyset$.

Let T_0 be a text for L such that every natural number occurs at most once, and such that no pause occurs before a natural number. As h **TextGEx_V**-learns L , there is a c minimal such that $\forall i \geq c : h(T_0[i]) = h(T_0[c])$. Because of the postdictive completeness and conservativeness of h , h cannot change its mind when seeing a pause, so $T_0[c] \in B$ from (5.73). Let $\tau = T_0[c]$. From conservativeness alone we get

$$(\forall \tau' \in A | \text{content}(\tau') \subseteq L \wedge \tau \subseteq \tau') h(\tau) = h(\tau'). \quad (5.82)$$

□ (FOR CLAIM 3.1)

As $C \neq \emptyset$, we can now let

$$\sigma = \min_{\prec} C. \quad (5.83)$$

Intuitively, this σ is the \prec -minimal locking-sequence¹⁶ in B of h on L , where σ only locks against sequences in A .

Let $D = \text{content}(\sigma)$. As σ is chosen to be the \prec -minimal locking sequence, for all (finitely many) $\tau \in B_D$ with $\tau \prec \sigma$, there is an extension $D_\tau \supseteq D$ such that $\tau \notin B_{D_\tau}$. Unioning all the sets D_τ we get a set D' such that $\sigma = \min_{\prec} B_{D'}$, and, hence, $(\forall D'' | D' \subseteq D'' \subseteq L) \min_{\prec} B_{D''} = \sigma$. By the definition of h' , this ensures convergence of h' on any text for L . □ (FOR CLAIM 3)

Postdictive completeness of h' is straightforward, completing the proof of (5.70). □

The following proposition shows that total postdictive complete and total conservative, set-driven learners are automatically totally prudent. This, too, is of important technical use for proving Theorem 5.1.2.

¹⁶ See Definition 3.3.5.

Proposition 5.3.6. Let δ be sequence acceptance criterion, let $\mathcal{C} \subseteq \mathcal{R}$. Let $h \in \mathcal{P}$. We have

$$\mathcal{TPrudCT}(\mathbf{PcpConv})\mathbf{TxtSd}\delta\mathbf{Ex}_V(h) = \mathcal{CT}(\mathbf{PcpConv})\mathbf{TxtSd}\delta\mathbf{Ex}_V(h).$$

Proof: Let $D \subseteq \mathbb{N}$, let $e = h(D)$ and $L = W_e$. We need to show that $L \in \mathbf{TxtSdEx}_V(h)$. By conservativeness, we have that $(\forall D' | D \subseteq D' \subseteq L)h(D') = e$ (hence, D is a locking-set for h on L , see Definition 3.3.5). Let T be a text for L , let c be such that $D \subseteq \text{content}(T[c])$. Then we have $\forall i \geq c : h(\text{content}(T[i])) = e$. \square

5.4 Learning of Functions

As mentioned in the introduction to the present chapter, Theorem 5.4.3 below is the main theorem for this section. It states that, for the general effective numbering of all (partial) computable functions, φ , combinations of postdictive completeness, conservativeness and prudence forbid *some* postponement tricks iff postdictive completeness is part of the combination. Theorem 5.4.2 leads up to Theorem 5.4.3.

Finally, Theorem 5.4.4 and its proof show that even the combination of postdictive completeness, conservativeness and prudence does *not* forbid *all* postponement tricks.

Theorem 5.3.1 above shows that conservative and prudent *text* learning can be arbitrarily postponed. This has the simple corollary that conservative and prudent *function* learning can also be arbitrarily postponed. However, even stronger, Theorem 5.4.1 just below states that, regarding conservative and prudent function learning, every set learnable by a partial computable learner learning from canonical text is also learnable by a *polynomial time* learner, learning from *arbitrary* text.

Theorem 5.4.1. Let $\delta \in \{\mathcal{R}^2, \mathbf{Conv}\}$, $\mathcal{D} \in \{\mathbf{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\mathbf{Id}, \mathcal{TPrud}\}$. Then

$$\begin{aligned} \mathcal{DPFArbG}\delta\mathbf{Ex}_\varphi &= \mathcal{DG}\delta\mathbf{Ex}_\varphi \\ \mathcal{D}'\mathcal{PFT}\delta\mathbf{ArbG}\delta\mathbf{Ex}_\varphi &= \mathcal{D}'\mathcal{T}\delta\mathbf{GEx}_\varphi \end{aligned}$$

Proof: “ \subseteq ” is immediate. Let $h \in \mathcal{P}$ and $\mathcal{S} \in \{\mathcal{DG}\delta\mathbf{Ex}_\varphi(h), \mathcal{D}'\mathcal{T}\delta\mathbf{GEx}_\varphi(h)\}$. Fix a program for h . By Lemma 2.2.6 (specifically items (i), (ii), (iii), (vi), (vii), (viii), (ix) with (2.7), (x), (xi), (xiii) and (xiv)), there is $h' \in \mathbf{PF}$ such that

$$\forall \sigma : h'(\sigma) = \begin{cases} \text{if there is a } \leq\text{-maximal } \tau \leq |\sigma| : \\ h(\tau), & \text{content}(\tau) \subseteq \text{content}(\sigma) \wedge h(\tau) \downarrow_{|\sigma|} \wedge \\ & \forall i < \#\text{elets}(\tau) : \pi_1(\tau(i)) = i; \\ h(\emptyset), & \text{otherwise.} \end{cases} \quad (5.84)$$

□

Imposing the restrictions of total prudence significantly lowers learning power, as stated by the next theorem.¹⁷

Theorem 5.4.2. We have

$$\{\mathcal{S} \mid \exists \mathcal{S}' \in \mathbf{UComp} : \mathcal{S} \subseteq \mathcal{S}'\} = \mathcal{TPrudGEx}_\varphi \quad (5.85)$$

$$= \mathcal{TPrudT}(\mathbf{ConvPcp})\mathbf{ArbGEx}_\varphi. \quad (5.86)$$

Proof: “ $\mathcal{TPrudT}(\mathbf{ConvPcp})\mathbf{ArbGEx}_\varphi \subseteq \mathcal{TPrudGEx}_\varphi$ ” is immediate. Regarding “ $\mathcal{TPrudGEx}_\varphi \subseteq \{\mathcal{S} \mid \exists \mathcal{S}' \in \mathbf{UComp} : \mathcal{S} \subseteq \mathcal{S}'\}$ ”: see [CS83].

Regarding “ $\{\mathcal{S} \mid \exists \mathcal{S}' \in \mathbf{UComp} : \mathcal{S} \subseteq \mathcal{S}'\} \subseteq \mathcal{TPrudT}(\mathbf{ConvPcp})\mathbf{ArbGEx}_\varphi$ ”: Let $e \in \mathcal{R}$ be such that $\forall i : \varphi_{e(i)} \in \mathcal{R}$. Let $e' \in \mathcal{R}$ be such that

$$\forall i, \sigma : \varphi_{e'(i)} = \begin{cases} \varphi_{e(j)}, & \text{if } i = 2j; \\ \sigma \diamond 0^\infty, & \text{otherwise, with } \sigma \text{ such that } i = 2\langle \sigma \rangle_{\text{Seq}} + 1. \end{cases} \quad (5.87)$$

¹⁷ It is known that a learner h restrained to output nothing but programs for total functions (which \mathcal{TPrud} does) is known to restrict the learning power of h to a uniformly computable set of functions (see, for example, [CS83, Theorems 2.19, 2.21 & 2.23]).

Note that each sequence σ is extendible to some function that has a program enumerated by e' . Let h' be such that

$$\forall \sigma : h'(\sigma) = e'(\mu j. \text{content}(\sigma) \subseteq \varphi_{e'(j)}). \quad (5.88)$$

□

The first main result of the present section says, as pointed out above, for the setting of function learning, combinations of postdictive completeness, conservativeness and prudence forbid *some* postponement tricks iff postdictive completeness is part of the combination.

Theorem 5.4.3. Let $\delta \in \{\mathcal{R}^2, \mathbf{Pcp}, \mathbf{Conv}, \mathbf{PcpConv}\}$, $\mathcal{D} \in \{\text{Id}, \mathbf{Prud}\}$ and $\mathcal{D}' \in \{\text{Id}, \mathcal{T}\mathbf{Prud}\}$. Then

$$\begin{aligned} \mathcal{D}\mathbf{PFG}\delta\mathbf{Ex}_\varphi \subset \mathcal{D}\mathbf{RG}\delta\mathbf{Ex}_\varphi &\Leftrightarrow \delta \subseteq \mathbf{Pcp} \\ \mathcal{D}'\mathbf{PFT}\delta\mathbf{GEx}_\varphi \subset \mathcal{D}'\mathcal{T}\delta\mathbf{GEx}_\varphi &\Leftrightarrow \delta \subseteq \mathbf{Pcp}. \end{aligned}$$

Proof: We show both equivalences simultaneously. “ \Rightarrow ”: By Theorem 5.3.1. “ \Leftarrow ”: By 4.3.2, **EXPF**, the class of exponential time computable functions, is *not* learnable postdictively completely with polynomial time update:

$$\mathbf{EXPF} \not\subseteq \mathbf{PFGPcpEx}_\varphi. \quad (5.89)$$

As **EXPF** \in \mathbf{UComp} , we have

$$\mathbf{UComp} \not\subseteq \mathbf{PFGPcpEx}_\varphi, \quad (5.90)$$

and Theorem 5.4.2 completes the proof. □

Next is our second main result for this section. As noted above in the introduction to the present chapter, this theorem says that, in the *general* setting of the present section, postdictive completeness, conservativeness and prudence do *not* forbid *all* postponement tricks.

Theorem 5.4.4. We have

$$\mathbf{LinF} \in \mathcal{TPrudPFT}(\mathbf{PcpConv})\mathbf{ArbGEx}_\varphi.^{18} \quad (5.91)$$

Proof: Recall the effective numbering $V^{\mathcal{L}time}$ defined just before Theorem 5.2.2, based on the programming system $\theta^{\mathcal{L}time}$. By linear time s-m-n in φ , there is a function $s \in \mathbf{LinF}$ such that

$$\forall e : \varphi_{s(e)} = \theta_e^{\mathcal{L}time}. \quad (5.92)$$

Thus, Theorem 5.2.2 above,¹⁹ with the help of Propositions 3.3.2 and 3.3.3, shows (5.91). \square

¹⁸ By [RC94, Theorem 6.5] no natural or practical hypothesis space for \mathbf{LinF} can be polynomial time decidable.

¹⁹ The proof of Theorem 5.2.2 explicitly employs postponement tricks; hence, so does the proof of Theorem 5.4.4.

Chapter 6

DYNAMIC MODELING

In the table given in Section 1.2, we have seen that there is a missing, not heretofore studied category entry for *offline* with *reactive learner*. We referred to this category as *dynamic modeling*, and it is the subject of the present chapter. **XBc**-learning exemplifies the category of *dynamic modeling*.

Recall from Chapter 3, *h XBc-learns* a target g iff,¹ 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 .²

In cognitive science, *theory of mind* refers to one's having a model (or models) of another's thoughts, emotions, and perspectives — including those different from one's 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

¹ Recall from Definition 3.1.4, **X** we pronounce *cross*, and it is short for *crossfeed*. Of course crossfeeding of data is common to both the categories of coordination and dynamic modeling. In Chapter 1 we use the **X** also in talking about the former category. **Bc** stands for *behaviorally correct* [CS83].

² Note that h 's outputs are *not* required to be programs for the *function* g , but rather for g 's output sequence.

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 present chapter.

Section 6.1 involves cooperativeness vs. secretiveness in dynamic modeling. Considered are dynamic modelers which may or may not, in return, be dynamically modeled themselves. Importantly, Proposition 6.1.1 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 .³

Surprisingly, Theorem 6.1.3 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 6.1.3 is particularly elegant.

We say that computable h is *extremely uncooperative* iff $\{\text{computable } g \mid h \text{ XBc-learns } g \wedge g \text{ XBc-learns } h\} = \emptyset$. Another main result of this chapter, Theorem 6.1.7, says there are extremely uncooperative computable h which, nonetheless, are infinitely successful, i.e, such that h **XBc**-learns infinitely many computable g . Such h s openly disclose models of g s they **XBc** learn while keeping secret their own models from these g s. For comparison and contrast, a *zero-knowledge proof* [BSMP91] permits open, convincing disclosure of its existence without disclosing how it works.

³ Actually, Proposition 6.1.1 gives a stronger result.

Section 6.2 features a theorem saying that *any computably enumerable set of computable functions* is **XBc**-learnable⁴ (Theorem 6.2.5) and two general and powerful correspondence theorems (Theorems 6.2.12 and 6.2.14) regarding many of the criteria employing **X**.

The first of these correspondence theorems *immediately* yields Corollary 6.2.13 which implies, for *example*, that quadratic time **XBc**-learnability is strictly more powerful than linear time **XBc**-learnability.

Theorem 6.2.14 *immediately* yields Corollary 6.2.15 which provides a number of learning criteria hierarchies and separations. An *example*: the powers of **XBc**-learning and of Coord-learning are incomparable.

Regarding fairness⁵ of runtime restricted learning in dynamic modeling, we have that Theorem 6.2.12 implies a tight hierarchy in learning power in dependence on the runtime restriction. This is exemplified, as mentioned above, in Corollary 6.2.13. Hence, dynamic modeling *naturally* forbids *some* postponement tricks.⁶ However, as we'll see from the proof of Theorem 6.2.5, *not all* postponement tricks are forbidden.

For the remainder of this chapter, we will exclusively use φ as our hypothesis space; thus, we will omit any further mention thereof.

For this Chapter only, the default target presenter is **Immediate**.

6.1 Cooperation and Secretiveness

The main emphasis of the present chapter, as mentioned in its introduction, features **XBc**-learning, but, based on the thinking of other chapters of this thesis,

⁴ Theorem 6.2.5 actually gives a stronger result.

⁵ See Section 1.3.

⁶ As we have seen in previous chapters, nothing like this happens for, for example, Ex-learning (see Section 1.3), where learning with linear time learners is just as powerful as with partial computable learners.

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 such g , then, *easily* **XEx**-learns h .⁷

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 6.1.3 further below.

Proposition 6.1.1. 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** (Theorem 2.3.3) and linear time padding (Theorem 2.3.6), e_n such that, with

$$p = \lambda x. \text{pad}(n, e_n), \quad (6.2)$$

$$\forall x : \varphi_{e_n}(x) = g(p[x]). \quad (6.3)$$

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

$$\forall \sigma : h_n(\sigma) = \text{pad}(n, e_n). \quad (6.4)$$

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

$$\varphi_{p(t)}(x) \stackrel{(6.2)}{=} \varphi_{\text{pad}(n, e_n)}(x) \stackrel{(2.38)}{=} \varphi_{e_n}(x) \stackrel{(6.3)}{=} g(p[x]) \stackrel{\text{choice of } q}{=} q(x). \quad (6.5)$$

⁷ There are obvious learners that **GEx**-learn all almost everywhere constant functions. For example, a learner g would initially conjecture a program for the everywhere 0 function, and then, on input $\sigma \neq \emptyset$, g would conjecture a canonical program $\text{last}(\sigma)$ -extension of σ . Then, for such g , we have,

$$\forall h \in \mathcal{R} : [h \text{ **XEx**-learns } g \Rightarrow g \text{ **XEx**-learns } h]. \quad (6.1)$$

Hence, intuitively, **XEx**-learners can *trivially* not keep models of themselves secret.

Hence, h_n **XBc**-learns g . Trivially, using (2.38), we have for all $l \neq m$, $h_l \neq h_m$. This shows that there are infinitely many different (constant) functions **EX**-learning g . \square

Next we give some definitions regarding cooperation and secretiveness.

Definition 6.1.2. We define the following sequence acceptance criteria.⁸

- 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:

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

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 $\lambda\sigma.\#\text{elets}(\sigma)$.⁹ At first glance, it seems likely that all computable functions can be **XBcs**-learned, as, by Proposition 6.1.1 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 6.1.3. 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

⁸ For each sequence acceptance criterion δ we let $\bar{\delta} = \{(p, q) \in \mathfrak{P}^2 \mid (p, q) \notin \delta\}$ be the *complement* of δ , and let $\delta^{-1} = \{(q, p) \in \mathfrak{P}^2 \mid (p, q) \in \delta\}$ be the *inverse* of δ .

⁹ For all $g \in \mathcal{R}$, and $p, q \in \mathcal{R}$ such that $\mathbf{X}(\lambda\sigma.\#\text{elets}(\sigma), g) = (p, q)$, we have that p is the identity on \mathbb{N} ; hence, $\#\text{elets} = \lambda\sigma.\#\text{elets}(\sigma)$ does not **XBc**-learn g .

hand, Theorem 6.1.3 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 $EC = \{h \in \mathcal{R} \mid \forall g \in \mathcal{R} : g \text{ XBc-learns } h \Rightarrow h \text{ XBc-learns } g\}$.¹⁰

Theorem 6.1.3 (Secretiveness Fails).

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

Proof: By 1-1 linear time **ORT** (Theorem 2.3.8), there is 1-1 $g \in \mathbf{LinF}$ such that

$$\forall \tau, x : \varphi_{g(\tau)}(x) = \text{last}(g^{-1}(\varphi_{\text{last}(\tau)}(x+1))).^{11} \quad (6.6)$$

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

$$\forall n \geq n_0 : \varphi_{p(n)} = q. \quad (6.7)$$

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

We have

$$\forall n \geq n_0 + 1, x : \varphi_{p(n-1)}(x+1) \stackrel{(6.7)}{=} q(x+1) \stackrel{\text{choice of } q}{=} g(p[x+1]). \quad (6.8)$$

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

$$\begin{aligned} \varphi_{q(n)}(x) &\stackrel{\text{choice of } q}{=} \varphi_{g(p[n])}(x) \stackrel{(6.6)}{=} \text{last}(g^{-1}(\varphi_{p(n-1)}(x+1))) \\ &\stackrel{(6.8)}{=} \text{last}(g^{-1}(g(p[x+1]))) = \text{last}(p[x+1]) = p(x). \end{aligned} \quad (6.9)$$

□ (FOR CLAIM)

¹⁰ It is easy to see that $EC = \{h \in \mathcal{R} \mid \mathbf{XBcs}^{-1}(h) = \emptyset\}$.

¹¹ Note that $\varphi_{g(\tau)}(x)$ might be undefined for various reasons, for example, because last is not total. Furthermore, note that accessing g^{-1} *is* a valid use of **ORT**.

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

[MO99], in effect, examined uncooperativeness of *coordinators*. In particular, two sets of total computable functions are constructed such that any learner coordinating with *all* of the functions from one of the sets cannot coordinate with *any* function from the other set. Furthermore, [CJM⁺05] extended this result showing that, for all $k \geq 2$, one can find k such sets of uncooperative learners. Just below, we give an analog of this result for cooperation in the \mathbf{XBcc} -sense (Theorem 6.1.4). This theorem provides an infinite family of uncooperative sets, so that, any learner that can \mathbf{XBc} -learn *any* of the functions in one of the sets, cannot \mathbf{XBc} -learn *any* of the functions of any of the other sets.

Theorem 6.1.4 (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 \mathbf{XBcc} -learn each other, while each function $h \in \mathcal{P}$ \mathbf{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 [Cas74, Theorem 23] we find a 1-1 $e \in \mathcal{R}$ such that, for all m, n , $\varphi_{e(m,n)}$ total and

$$\forall \tau, x : \varphi_{\varphi_{e(m,n)}(\tau)}(x) = \begin{cases} n, & \text{if } \tau = \emptyset; \\ \varphi_{e(m,n)}(x), & \text{else if } \#\text{elets}(\tau) = 1 \text{ and} \\ & \varphi_{\tau(0)}(0) \downarrow = n; \\ f(e(m,n), \tau(1), x), & \text{else if } \varphi_{\tau(0)}(0) \downarrow = n; \\ \uparrow, & \text{otherwise.} \end{cases} \quad (6.10)$$

Intuitively, each $\varphi_{e(m,n)}$ declares its “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 $g = \varphi_{e(m,n)}$.

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 $\#\text{elets}(\tau) \geq 1$ we have $\varphi_{\tau(0)}(0) = \varphi_{p(0)} \stackrel{\text{def } \mathbf{X}}{=} \varphi_{h(\emptyset)}(0) \neq n$; thus, by (6.10) $\forall x : \varphi_{g(\tau)}(x) = \varphi_{\varphi_{e(m,n)}(\tau)} \uparrow$. Hence, $\forall^\infty t : \varphi_{q(t)} \stackrel{\text{def } \mathbf{X}}{=} \varphi_{g(p[t])} \neq p$. \square (FOR CLAIM 1)

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 $g \in \mathbf{XBc}(h)$.

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

$$\varphi_{p(0)}(0) \stackrel{\text{def } \mathbf{X}}{=} \varphi_{h(\emptyset)}(0) \stackrel{(6.10)}{=} n; \quad (6.11)$$

$$\varphi_{q(0)}(0) \stackrel{\text{def } \mathbf{X}}{=} \varphi_{g(\emptyset)}(0) \stackrel{(6.10)}{=} n. \quad (6.12)$$

Hence,

$$\varphi_{q(1)} \stackrel{\text{def } \mathbf{X}}{=} \varphi_{g(p[1])} \stackrel{(6.10) \ \& \ (6.11)}{=} g. \quad (6.13)$$

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

$$\begin{aligned}
\varphi_{p(t)}(x) & \stackrel{\text{def } \mathbf{X}}{=} \varphi_{g(q[t])}(x) \\
& \stackrel{(6.10) \& (6.12)}{=} f(e(m,n), q(1), x) \\
& \stackrel{\text{def } f}{=} \mathbf{X}_2(\varphi_{e(m,n)}, \varphi_{q(1)})(x) \\
& \stackrel{(6.13)}{=} \mathbf{X}_2(h, g)(x) \\
& \stackrel{\text{def } \mathbf{X}_2}{=} q(x).
\end{aligned} \tag{6.14}$$

□ (FOR CLAIM 2)

□

As a contrast to the extremely cooperative functions as defined above, we say that $h \in \mathcal{R}$ is *extremely uncooperative* iff $\mathbf{XBcc}(h) = \emptyset$ (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 \mathbf{XBc} -learn any function. However, many of the functions $h \in EU$ will not \mathbf{XBc} -learn anything. We define a (computable) operator Ψ 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 6.1.7 below states the existence of $h' \in EU$ with $\mathbf{XBc}(h')$ infinite.

Definition 6.1.5. For all $h \in \mathcal{R}$ there is, by linear time **ORT** (Theorem 2.3.8), $h' \in \mathbf{LinF}$ such that

$$\forall \sigma, x : \varphi_{h'(\sigma)}(x) = \begin{cases} \varphi_{h(\sigma)}(x), & \text{if } \sigma = \emptyset \vee \exists s \geq \#\text{elets}(\sigma), t : \\ & \varphi_{\varphi_{h(\sigma)}(s)}(t) \downarrow \neq h'(\varphi_{h(\sigma)}[t]) \downarrow; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{6.15}$$

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 $h \in \mathcal{R}$, $\Psi(h) \in \mathbf{LinF}$.

Next is a lemma, saying that $\text{range}(\Psi) \subseteq EU$ as claimed above.

Lemma 6.1.6. 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

$$\forall^\infty n : \varphi_{p(n)} = q; \quad (6.16)$$

$$\forall^\infty n : \varphi_{q(n)} = p. \quad (6.17)$$

Thus,

$$\forall^\infty n : \varphi_{h'(q[n])} \stackrel{\text{def } \mathbf{X}}{=} \varphi_{p(n)} \stackrel{(6.16)}{=} q \in \mathcal{R}. \quad (6.18)$$

For each $n \in \mathbb{N}$, $\varphi_{h'(q[n])} \in \mathcal{R}$ implies that in (6.15) for $\sigma = q[n]$ we have that the first case holds; that is,

$$\forall^\infty n \exists s > n, t : \varphi_{\varphi_{h(q[n])}(s)}(t) \downarrow \neq h'(\varphi_{h(q[n])}[t]) \downarrow. \quad (6.19)$$

Thus,

$$\forall^\infty n : \varphi_{h(q[n])} \stackrel{(6.15)}{=} \varphi_{h'(q[n])} \stackrel{(6.18)}{=} q. \quad (6.20)$$

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

$$\varphi_{\varphi_{h(q[n])}(s)}(t) \stackrel{(6.20)}{=} \varphi_{q(s)}(t) \stackrel{(6.17)}{=} p(t) \stackrel{\text{def } \mathbf{X}}{=} h'(q[t]) \stackrel{(6.18) \& (6.20)}{=} h'(\varphi_{h(q[n])}[t]), \quad (6.21)$$

a contradiction to (6.19). \square

Theorem 6.1.7 (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 \mathbf{XBc} -learned by $\Psi(h)$ can \mathbf{XBc} -learn $\Psi(h)$).

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

$$\forall \sigma, x : \varphi_{h(\sigma)}(x) = \begin{cases} 0, & \text{if } \sigma = \emptyset; \\ \sigma(0), & \text{otherwise.} \end{cases} \quad (6.22)$$

There is an infinite set S such that for all $e \in S$, $\varphi_e(0) = \Psi(h)(\emptyset) + 1$. Obviously, $\Psi(h)$ **XBcs**-learns $\{\lambda x.e \mid e \in S\}$.

□

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

Corollary 6.1.8.

$$EUXBcs \subset \mathcal{R}XBcs.^{12}$$

We defer the proof until after Theorem 6.2.11.

6.2 General Crossfeeding

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

Proposition 6.2.1. Let δ be a sequence acceptance criterion, let $\mathcal{C} \subseteq \mathcal{P}$. Then we have

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

Proof: “ \Rightarrow ”: Let h witness $\mathcal{R} \in \mathcal{C}X\delta$. Let $g \in \mathcal{R}$. Let $p \in \mathcal{P}$ be such that $\mathbf{G}(h, g) = (p, g)$. Now we have $\mathbf{G}(h, g) \stackrel{(3.4)}{=} \mathbf{X}(h, \lambda \sigma.g(\#\text{elets}(\sigma))) \in \delta$.

¹² Less surprisingly, one can also show $ECXBcc \subset \mathcal{R}XBcc$. One could, for example, show this analogously to the proof of Corollary 6.1.8, which uses notions from Section 6.2. For example, a learner for the almost everywhere constant functions as in Footnote 7, would be maximal (see Definition 6.2.9), but not extremely cooperative.

“ \Leftarrow ”: Let h witness $\mathcal{R} \in \mathcal{CG}\delta$. Let $g \in \mathcal{R}$. Let $p, q \in \mathcal{P}$ be such that $\mathbf{X}(h, g) = (p, q)$. Now we have $\mathbf{X}(h, g) \stackrel{(3.2)}{=} \mathbf{G}(h, q) \in \delta$. \square

We get the following corollary by a theorem of Harrington, cited in [CS83].

Corollary 6.2.2.

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

Next we give the definition of a useful operator μ^{lin} which, in a sense made precise in the succeeding proposition, provides a *linear time approximation of the minimization operator*.

Definition 6.2.3. Let $z \in \mathbb{N}$ be such that $\varphi_z \in \mathcal{P}_{0,1}$. Define a (total) computable predicate $R_z \in \mathcal{R}_{0,1}$ such that

$$\forall n, \sigma : R_z(n, \sigma) = \begin{cases} 0, & \text{if } \exists \tau \leq |\sigma| : \tau \subseteq \sigma \wedge \varphi_z(n, \tau) \downarrow_{|\sigma|} = 0; \\ 1, & \text{otherwise.} \end{cases} \quad (6.23)$$

Let, for all σ ,

$$f_z(\sigma) = \begin{cases} n, & \text{for the } \leq\text{-least } n \leq |\sigma| \text{ s.t. } R_z(n, \sigma) = 1; \\ 0, & \text{if no such } n \text{ exist,} \end{cases} \quad (6.24)$$

and

$$\mu^{\text{lin}}n.\varphi_z(n, \sigma) = f_z(\sigma[i]), \text{ for the } \leq\text{-max } i \leq |\sigma| \text{ such that } |\sigma[i]| \leq \log(|\sigma|). \quad (6.25)$$

The following proposition gives useful properties of the linear time approximation of the minimization operator, μ^{lin} .

Proposition 6.2.4. Let $z \in \mathbb{N}$ be such that $\varphi_z \in \mathcal{P}_{0,1}$. We have

$$[\lambda\sigma.\mu^{\text{lin}}n.\varphi_z(n, \sigma)] \in \mathbf{LinF}. \quad (6.26)$$

For each text T , let $E_T = \{n \mid \forall i : \varphi_z(n, T[i])\}$. If $\varphi_z \in \mathcal{R}_{0,1}$, we have, for all texts T ,

$$[E_T \neq \emptyset \Rightarrow \lambda i.\mu^{\text{lin}}n.\varphi_z(n, T[i]) \rightarrow \min(E_T)]. \quad (6.27)$$

Proof: From Lemma 2.2.6 (specifically items (i), (ii), (vi), (viii), (ix), (x) and (xiii)) we get, for all z , R_z computable in polynomial time. Hence, also from Lemma 2.2.6 (specifically items (i), (ii), (iii), (vi), (vii) and (ix)) we get, for all z , f_z is computable in polynomial time.

The required $\sigma[i]$ in the definition of μ^{lin} is, by basic reasoning about multi-tape Turing machines in the φ -system, computable in linear time.¹³ Let Q be a monotone increasing polynomial bounding the runtime of f . Therefore, for *all but finitely many* σ , the call on f_z in the definition of μ^{lin} will take no more than

$$Q(|\sigma[i]|) \leq Q(\log(|\sigma|)) \leq |\sigma| \tag{6.28}$$

time, using basic reasoning about polynomials and \log . Using Lemma 2.2.6 (specifically items (vi) and (xiii)) we get (6.26).

Regarding (6.27): Let T be a text such that $E_T \neq \emptyset$. Obviously,

$$\forall n, i, j : [(i \leq j \wedge R_z(n, T[i]) = 0) \Rightarrow R_z(n, T[j]) = 0]. \tag{6.29}$$

Let $m = \min(E_T)$. We have

- $\forall i : R(m, T[i]) = 1$;

¹³ Given a sequence σ , a multi-tape TM could extract $\sigma[i]$ with $i \leq |\sigma| \leq$ -maximal such that $|\sigma[i]| \leq \log(|\sigma|)$ as follows. First, using Lemma 2.2.6 (i) and (iv), to compute and store $\log(|\sigma|)$ on a separate TM tape. We consider the content of this tape the value of a counter c .

Note that, due to our coding of sequences from Section 2.1, $\sigma = \overline{\sigma(0)} \overline{\sigma(1)} \dots \overline{\sigma(\#\text{elems}(\sigma) - 1)}$. We process the coded elements \overline{x} of σ from left to right. For each \overline{x} , we decrement c once for each bit in \overline{x} (the proof of [RC94, Lemma 3.2(1)] shows that this can be achieved within appropriate timebounds). We terminate if we cannot decrement c any more (due to having reached 0). Otherwise, we copy over the just processed \overline{x} to our output tape, and proceed with the next element $\overline{x'}$ in σ . When we finally terminate, we have $\sigma[i]$ as desired on our output tape.

- by (2.7), $\forall^\infty i : m < \log(|T[i]|)$;
- as m minimal in E_T , $\forall n < m \exists i : R(n, T[i]) = 0$; hence, using (6.29), $\forall n < m \forall^\infty i : R(n, T[i]) = 0$.

By the definition of μ^{lin} , we can now conclude

$$\forall^\infty i : \mu^{\text{lin}} n \cdot \varphi_z(n, T[i]) = m. \quad (6.30)$$

□

Gold [Gol67] 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**-learnable.

Theorem 6.2.5 (Dynamic Modeling by Enumeration). Let $r \in \mathcal{R}$ be an algorithmic enumeration of program numbers of (total) computable functions. We have

$$\{\varphi_{r(n)} \mid n \in \mathbb{N}\} \in \mathbf{LinFXEx}^{14}$$

Because of the vagaries of crossfeeding, the proof of the theorem is not as straightforward as it is for **GEx**. Our proof makes use of **ORT**.

Proof: By linear time **ORT** (see Theorem 2.3.5), there are programs e, e', e'' , as well as an infinite enumeration $s \in \mathbf{LinF}$ of programs such that, with $h = \varphi_e$, $u = \varphi_{e'}$ and

¹⁴ In fact, with a slight modification of the proof, **Ex** could be replaced by any δ from a wide set of sequence acceptance criteria, for example, $\delta = \mathbf{ConvBc}$. Note that, by Theorem 6.2.14, **XEx** and **XConvBc** are incomparable.

$$P = \varphi_{e''},$$

$$\forall n, \sigma : P(n, \sigma) = \begin{cases} \uparrow, & \text{if } \forall i < \#\text{elets}(\sigma) : \mathbf{X}_2(h, \varphi_{r(n)})(i) \downarrow;^{15} \\ 1, & \text{else if } \sigma \subseteq \mathbf{X}_2(h, \varphi_{r(n)}); \\ 0, & \text{otherwise.} \end{cases} \quad (6.31)$$

$$\forall \sigma : u(\sigma) = \mu^{\text{lin}} n. \varphi_{e''}(n, \sigma);^{16} \quad (6.32)$$

$$\forall m : \varphi_{s(m)} = \mathbf{X}_2(h, \varphi_{r(m)}); \quad (6.33)$$

$$\forall \sigma : h(\sigma) = s(u(\sigma)). \quad (6.34)$$

By (6.26) in Proposition 6.2.4, $u \in \mathbf{LinF}$; hence, $h \in \mathbf{LinF}$. Therefore, P is total (thus, $P \in \mathcal{R}_{0,1}$).

Let $g \in \{\varphi_{r(n)} \mid n \in \mathbb{N}\}$. We show that h **EX**-learns g . Let n be such that $\varphi_{r(n)} = g$ and let $p, q \in \mathcal{R}$ be such that $\mathbf{X}(h, g) = \mathbf{X}(h, \varphi_{r(n)}) = (p, q)$. Obviously, for all i , $q[i] \subseteq \mathbf{X}_2(h, \varphi_{r(n)})$, verifying the antecedent of (6.27) from Proposition 6.2.4 with q for T . Hence, we have the conclusion of (6.27) with for q in the place of T , showing that there is $m \in \mathbb{N}$ such that

$$\lambda i. u(q[i]) \rightarrow m, \quad (6.35)$$

and, also applying (6.31),

$$\forall i : q[i] \subseteq \mathbf{X}_2(h, \varphi_{r(m)}). \quad (6.36)$$

Hence,

$$q = \mathbf{X}_2(h, \varphi_{r(m)}). \quad (6.37)$$

The following completes the proof.

$$\forall^\infty i : \varphi_{p(i)} \stackrel{\text{def } p}{=} \varphi_{h(q[i])} \stackrel{(6.34)}{=} \varphi_{s(u(q[i]))} \stackrel{(6.35)}{=} \varphi_{s(m)} \stackrel{(6.33)}{=} \mathbf{X}_2(h, \varphi_{r(m)}) \stackrel{(6.37)}{=} q. \quad (6.38)$$

¹⁵ See Definition 3.1.4 for \mathbf{X}_2 .

¹⁶ See Definition 6.2.3 for μ^{lin} . Note that there are postponement tricks used for the definition of μ^{lin} .

□

Note that 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 6.2.13 below, for which we now give definitions to set it up.

We give the following technically useful definitions.

Definition 6.2.6. Let δ be a sequence acceptance criterion.

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

The following examples illustrate the first two of the just above definitions.

Example 6.2.7.

- \mathbf{Bc}^* is a trivial sequence acceptance criterion, while $\mathbf{Ex}, \mathbf{Bc}, \mathbf{Bcs}, \mathbf{Bcc}$ and \mathbf{M} are non-trivial.
- \mathbf{Ex}, \mathbf{Bc} and \mathbf{Bc}^* allow for linear time path finding for \mathcal{R} as witnessed by $r = \lambda x, e, \sigma.e$. \mathbf{M} allows for linear time path finding for total finite variants of constant functions.¹⁷

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

We give the following technical lemma to help with later proofs.

Lemma 6.2.8. Let δ, δ' be such that δ' is non-trivial. Let $h \in \mathcal{P}$. Let $\mathcal{S} = \mathbf{X}\delta(h)$. Then we have, for all $h' \in \mathcal{P}$ and σ , if $h(\sigma) \neq h'(\sigma)$ and $\mathcal{S} \subseteq \mathbf{X}\delta'(h')$, then, $\sigma \notin [\mathcal{S}]_h$.

¹⁷ \mathbf{M} allows for not necessarily linear time path finding for \mathcal{R} .

Proof: Let $h' \in \mathcal{P}$ be such that $\mathcal{S} \subseteq \mathbf{X}\delta'(h')$ and there is σ such that $h(\sigma) \neq h'(\sigma)$. Suppose, by way of contradiction, there is $\sigma \sqsubseteq$ -minimal such that

$$h(\sigma) \neq h'(\sigma) \quad (6.39)$$

and

$$\exists g \in \mathcal{S} : \sigma \subseteq \mathbf{X}_2(h, g). \quad (6.40)$$

Let $c = \#\text{elets}(\sigma)$.

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

$$\forall \tau : f'(\tau) = \begin{cases} g(\tau), & \text{if } \tau \subseteq \mathbf{X}_1(h, g); \\ f(\#\text{elets}(\tau) - (c + 1)), & \text{otherwise.} \end{cases} \quad (6.41)$$

Obviously, for all $f \in \mathcal{R}$, $f' \in \mathcal{S}$. Therefore,

$$\forall f \in \mathcal{R} : \mathbf{X}(h, f') \in \delta'. \quad (6.42)$$

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

$$\forall f \in \mathcal{R} : \mathbf{X}(h', f') \stackrel{(6.39)}{=} (r, \sigma \diamond f) \stackrel{(3.2)}{=} \mathbf{G}(h', \sigma \diamond f). \quad (6.43)$$

From (6.42) and (6.43), we get

$$\forall f \in \mathcal{R} : h \text{ } \mathbf{G}\delta'\text{-learns } \sigma \diamond f. \quad (6.44)$$

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

Below we develop the concept of maximal learners.

Definition 6.2.9. Let δ be a sequence acceptance criterion and $h \in \mathcal{P}$. h is called $\mathbf{X}\delta$ -maximal iff $\forall h' \in \mathcal{P} : [\mathbf{X}\delta(h) \subseteq \mathbf{X}\delta(h') \Rightarrow h = h']$. Let $\mathcal{M}_{\mathbf{X}\delta}$ be the set of all $\mathbf{X}\delta$ -maximal learners.

It is a consequence of the results in [CF99], that there are no maximal learners in the sense $\mathbf{G}\delta$, for $\delta \in \{\mathbf{E}\mathbf{x}^a, \mathbf{B}\mathbf{c}^n \mid a \in \mathbb{N} \cup \{*\}, n \in \mathbb{N}\}$ (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 $\mathbf{G}\mathbf{B}\mathbf{c}^*$ and $\mathbf{X}\mathbf{M}$ learners (Corollary 6.2.2 and [MO99, Corollary 16], respectively). The proof of Corollary 6.2.12 below employs the existence of $\mathbf{X}\delta$ -maximal learners, for various δ including \mathbf{M} . The following proposition shows the value of maximal learners for proving separations.

Proposition 6.2.10. Let δ be non-trivial. Let $\mathcal{C}, \mathcal{C}' \subseteq \mathcal{P}$.

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

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 6.2.11. Let δ be non-trivial such that $\forall \tau, \sigma \exists (p, q) \in \delta \cap \mathcal{R}^2 : \tau \subseteq p \wedge \sigma \subseteq q$. Let $h \in \mathcal{P}$. The following are equivalent.

- (i) h is $\mathbf{X}\delta$ -maximal.
- (ii) $[\mathbf{X}\delta(h)]_h = \text{Seq}$.
- (iii) $\forall \delta'$ non-trivial, $\forall h' \in \mathcal{P} : \mathbf{X}\delta(h) \subseteq \mathbf{X}\delta'(h') \Rightarrow h = h'$.

Proof: “(i) \Rightarrow (ii)”: Straightforward by showing the contrapositive.

“(ii) \Rightarrow (iii)”: Use Lemma 6.2.8.

“(iii) \Rightarrow (i)”: Trivial.

□

Note that Theorem 6.2.11 applies to $\delta = \mathbf{M}$.

We are now ready to prove Corollary 6.1.8 from the previous section.

Restatement of Corollary 6.1.8

$$E\mathbf{U}\mathbf{X}\mathbf{B}\mathbf{c}\mathbf{s} \subset \mathcal{R}\mathbf{X}\mathbf{B}\mathbf{c}\mathbf{s}.$$

Proof: Note that **Bcs** is *non-trivial*. By Proposition 6.2.10, it suffices to show that there is a **XBcs**-maximal learner in $\mathcal{R} \setminus EU$. With padding from Theorem 2.3.2, let $h \in \mathcal{R}$ be such that, for all σ with $\#\text{elets}(\sigma) > 0$, $h(\sigma) = \text{unpad}_1(\text{last}(\sigma))$. Let $r \in \mathcal{R}$ be such that $\forall e, p : \varphi_{r(e,p)} = \mathbf{X}_1(\varphi_e, \varphi_p)$.

Claim 1: $h \notin EU$.

Let $p \in \mathbb{N}$ be such that $\varphi_p = h$. By **KRT** (Theorem 2.3.3), there is $e \in \mathbb{N}$ such that

$$\forall x : \varphi_e(x) = \text{pad}(r(e, p), r(p, e)). \quad (6.45)$$

It is easy to see that $\mathbf{X}(h, g) \in \mathbf{Bcc}$. □ (FOR CLAIM 1)

By Theorem 6.2.11, it suffices to show $[\mathbf{XBcs}(h)]_h = \text{Seq}$ to apply Proposition 6.2.10.

Claim 2: $[\mathbf{XBcs}(h)]_h = \text{Seq}$.

Let $\sigma \in \text{Seq}$, let p be such that $\varphi_p = \lambda x. \uparrow$. By **KRT** there is $e \in \mathbb{N}$ such that

$$\forall x : \varphi_e(x) = \begin{cases} \sigma(x), & \text{if } x < \#\text{elets}(\sigma); \\ \text{pad}(r(e, p), p) & \text{otherwise.} \end{cases} \quad (6.46)$$

It is easy to see that $\mathbf{X}(h, g) \in \mathbf{Bcs}$ and $\sigma \subseteq \mathbf{X}_2(h, g)$. □ (FOR CLAIM 2)

□

Two of the main results in this section are the Learner Correspondence Theorem (Theorem 6.2.12 below) and the Sequence Acceptance Correspondence Theorem (Theorem 6.2.14 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 6.2.12 and 6.2.14 do not inform about trade-offs in learning power between more restricted learner admissibilities and less restricted sequence

acceptance criteria, or vice versa. For example, the relationship between **LinFXBc** and **XEx** cannot be directly determined by the just above mentioned theorems.

The next theorem and its proof features the items from Definition 6.2.6.

Theorem 6.2.12 (Learner Correspondence). Let δ be a non-trivial sequence acceptance criterion such that δ allows for linear time path finding for all 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

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

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 6.2.11. 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 $\forall \sigma : u(\sigma) = \max(2, \#\text{elets}(\hat{\sigma}))$. Let $r \in \mathbf{LinF}$ witness that δ allows for linear time path finding for total finite variants of constant functions. Define $h \in \mathcal{R}$ by¹⁸

$$\forall \sigma : h(\sigma) = \begin{cases} 0, & \text{if } \sigma = \emptyset; \\ f(\sigma(0)), & \text{if } \#\text{elets}(\sigma) = 1; \\ r(\#\text{elets}(\sigma) - u(\sigma), \text{patch}_0(\hat{\sigma}), \sigma[u(\sigma)]), & \text{otherwise.} \end{cases} \quad (6.47)$$

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 $f = h \circ \lambda n.\bar{n} \in \mathcal{C}'$. Using Proposition 6.2.10 and Theorem 6.2.11, it suffices to show $[\mathbf{X}\delta(h)]_h = \mathbb{Seq}$. Let σ be any sequence. Define $g = \lambda \rho. [(\sigma \diamond \lambda x.0)(\#\text{elets}(\rho))]$. We have $\mathbf{X}_2(h, g) \stackrel{(3.4)}{=} \sigma \diamond \lambda x.0$. From (6.47) and the choice of r it is now straightforward to verify that h **X** δ -learns g : Let $p, q \in \mathcal{R}$ be such that $\mathbf{X}(h, g) = (p, q)$. Then

$$q = \sigma \diamond \lambda x.0. \quad (6.48)$$

¹⁸ See Definition 2.2.3 regarding patch_0 .

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

$$q[\hat{n}] = \hat{\sigma}; \quad (6.49)$$

hence, as

$$\forall \tau, \tau' : \hat{\tau} = \hat{\tau}' \Rightarrow u(\tau) = u(\tau'), \quad (6.50)$$

we have

$$p(n) = h(q[n]) = r(n \dot{-} u(\sigma), \text{patch}_0(\hat{\sigma}), \sigma[u(\sigma)]). \quad (6.51)$$

Obviously, $\varphi_{\text{patch}_0(\hat{\sigma})} = q$. Thus, $\sigma \in [\mathbf{X}\delta(h)]_h$. \square

A direct application of Theorem 6.2.12, e.g., Corollary 6.2.13, implies that *dynamic modeling* forbids *some* postponement tricks.¹⁹ We let α , as from [CLRS01, §21.4], be a *very slow growing*, unbounded, linear time computable function \leq an inverse of Ackermann's function; let $\mathbf{LinF}^{+\varepsilon} = \{\varphi_e \in \mathcal{R} \mid \exists k \forall n : \Phi_e(n) \leq k \cdot |n| \cdot \log(|n|) \cdot \alpha(|n|) + k\}$. The classes \mathbf{LinF} and $\mathbf{LinF}^{+\varepsilon}$ have long been known to separate [HS65, HU79].

The following corollary gives a sample of the universal power of Theorem 6.2.12.

Corollary 6.2.13 (Learner Complexity Matters). Let $\delta \in \{\mathbf{Ex}, \mathbf{Bc}, \mathbf{M}\}$.

- (a) $\mathbf{LinFX}\delta \subset \mathbf{LinF}^{+\varepsilon}\mathbf{X}\delta$.
- (b) $\mathbf{PFX}\delta \subset \mathbf{EXPFX}\delta$.

A variant of the proof of “ \Rightarrow ” of Theorem 6.2.12 can be used to show $\mathcal{R}\mathbf{X}\delta \subset \mathcal{P}\mathbf{X}\delta$ for δ as in Corollary 6.2.13.²⁰

¹⁹ However, as we saw from Theorem 6.2.5 above, dynamic modeling does *not* forbid *all* postponement tricks.

²⁰ In this variant, we employ a partial computable function f with no (total) computable extension [Rog67, Theorem II, p. 37].

The following is our last main result for the present chapter, and it characterizes the relationship in learning power for criteria $\mathbf{X}\delta$ with varying δ .

Theorem 6.2.14 (Sequence Acceptance Correspondence). Let δ, δ' be sequence acceptance such that $\delta, \delta' \subseteq \mathcal{R}^2$ is non-trivial. We have

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

Proof: “ \Leftarrow ”: immediate.

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

Let $h \in \mathcal{R}$ be such that $\forall \sigma : h(\sigma) = p(\#\text{elets}(\sigma))$. Let $\mathcal{S} = \mathbf{X}\delta(h)$. Obviously, $\mathcal{S} \in \mathbf{X}\delta$.

Let $g \in \mathcal{R}$ such that $\forall \sigma : g(\sigma) = q(\#\text{elets}(\sigma))$. 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: $\forall n : h'(q[n]) \downarrow = p(n)$.

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 $h'(q[c]) \downarrow \neq p(c)$.

We use Lemma 6.2.8 to infer that $\mathcal{S} \notin \mathbf{X}\delta'(h')$, a contradiction. \square

The following corollary gives a sample of the universal power of Theorem 6.2.14.

Corollary 6.2.15 (Hierarchies and Separations).

- (a) For all $a, b \in \mathbb{N} \cup \{*\}$: $\mathbf{X}\mathbf{B}\mathbf{c}^a \not\subseteq \mathbf{X}\mathbf{E}\mathbf{x}^b$.
- (b) For all $a, b \in \mathbb{N} \cup \{*\}$: $\mathbf{X}\mathbf{E}\mathbf{x}^a \subseteq \mathbf{X}\mathbf{B}\mathbf{c}^b \Leftrightarrow a \leq b$.
- (c) For all $n \in \mathbb{N}$: $\mathbf{X}\mathbf{M} \not\subseteq \mathbf{X}\mathbf{E}\mathbf{x}^*, \mathbf{X}\mathbf{B}\mathbf{c}^n$.
- (d) For all $n \in \mathbb{N}$: $\mathbf{X}\mathbf{E}\mathbf{x}^*, \mathbf{X}\mathbf{B}\mathbf{c}^n \not\subseteq \mathbf{X}\mathbf{M}$.

Proof: By (3.6) and Theorem 6.2.14. \square

BIBLIOGRAPHY

- [ACJS04] A. Ambainis, J. Case, S. Jain, and M. Suraj. Parsimony hierarchies for inductive inference. *Journal of Symbolic Logic*, 69:287–328, 2004.
- [Ang80] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
- [AZ08] Y. Akama and T. Zeugmann. Consistent and coherent learning with δ -delay. *Information and Computation*, 206(11):1362–1374, 2008.
- [Bār71] J. Bārzdīņš. Prognostication of automata and functions. *Information Processing*, 1:81–84, 1971.
- [Bār74a] J. Bārzdīņš. Inductive inference of automata, functions and programs. In *Proceedings of the 20th International Congress of Mathematicians, Vancouver, Canada*, pages 455–560, 1974. English translation in, *American Mathematical Society Translations: Series 2 109 (1977)*, pp. 107–112.
- [Bār74b] J. Bārzdīņš. Two theorems on the limiting synthesis of functions. In *Theory of Algorithms and Programs, Latvian State University, Riga*, 210:82–88, 1974.
- [BB75] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.
- [BF72] J. Bārzdīņš and R. Freivalds. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.
- [BSMP91] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.
- [Bur05] M. Burgin. 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*, pages 143–147. Society for Modeling and Simulation (isbn: 1-56555-295-4), 2005.

- [Cas74] J. Case. Periodicity in generations of automata. *Mathematical Systems Theory*, 8:15–32, 1974.
- [Cas76] J. Case. Sortability and extensibility of the graphs of r.e. partial and total orders. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 22:1–18, 1976.
- [Cas94] J. Case. Infinitary self-reference in learning theory. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:3–16, 1994.
- [Cas09] J. Case, 2009. Private communication.
- [CCJ09] L. Carlucci, J. Case, and S. Jain. Learning correction grammars. *Journal of Symbolic Logic*, 74(2):489–516, 2009.
- [CCJS07] L. Carlucci, J. Case, S. Jain, and F. Stephan. Results on memory-limited u-shaped learning. *Information and Computation*, 205(10):1551–1573, 2007.
- [CF99] J. Case and M. Fulk. Maximal machine learnable classes. *Journal of Computer and System Sciences*, 58:211–214, 1999.
- [CJM+05] J. Case, S. Jain, F. Montagna, G. Simi, and A. Sorbi. On learning to coordinate: Random bits help, insightful normal forms, and competency isomorphisms. *Journal of Computer and System Sciences*, 71(3):308–332, 2005. Special issue for selected learning theory papers from COLT’03, FOCS’03, and STOC’03.
- [CJNM94] J. Case, S. Jain, and S. Ngo Manguelle. Refinements of inductive inference by Popperian and reliable machines. *Kybernetika*, 30:23–52, 1994.
- [CJSW04] J. Case, S. Jain, F. Stephan, and R. Wiehagen. Robust learning – rich and poor. *Journal of Computer and System Sciences*, 69:123–165, 2004.
- [CK08a] J. Case and T. Kötzing. Dynamic modeling in inductive inference. In *19th International Conference on Algorithmic Learning Theory (ALT’08)*, volume 5254 of *Lecture Notes in Artificial Intelligence*, pages 404–418. Springer, 2008.
- [CK08b] J. Case and T. Kötzing. Dynamically delayed postdictive completeness and consistency in learning. In *19th International Conference on Algorithmic Learning Theory (ALT’08)*, volume 5254 of *Lecture Notes in Artificial Intelligence*, pages 389–403. Springer, 2008.

- [CKP07] J. Case, T. Kötzing, and T. Paddock. Feasible iteration of feasible learning functionals. In M. Hutter, R. Servedio, and E. Takimoto, editors, *18th International Conference on Algorithmic Learning Theory (ALT'07)*, volume 4754 of *Lecture Notes in Artificial Intelligence*, pages 26–40. Springer-Verlag, Berlin, 2007.
- [CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.
- [Cob64] A. Cobham. The intrinsic computational difficulty of functions. *Proceedings of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science, Jerusalem, Israel*, pages 24–30, 1964.
- [Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.
- [CPK07] J. Case, T. Paddock, and T. Kötzing. Feasible iteration of feasible learning functionals, 2007. Work in progress.
- [CR09] J. Case and J. Royer. Program size complexity of correction grammars, 2009. Working draft.
- [CS83] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.
- [Edm65] J. Edmonds. Maximum matching and a polyhedron with 0,1 vertices. *Journal of research of the National Bureau of Standards.*, 69 B:125–130, 1965.
- [FKS95] R. Freivalds, E. B. Kinber, and C. H. Smith. On the intrinsic complexity of learning. *Information and Computation*, 123(1):64–71, 1995.
- [FS93] R. Freivalds and C. Smith. On the role of procrastination in machine learning. *Information and Computation*, 107(2):237–271, 1993.
- [Ful85] M. Fulk. *A Study of Inductive Inference Machines*. PhD thesis, SUNY at Buffalo, 1985.
- [Ful88] M. Fulk. Saving the phenomenon: Requirements that inductive machines not contradict known data. *Information and Computation*, 79:193–209, 1988.
- [Gol67] E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

- [HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.
- [HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory Languages and Computation*. Addison-Wesley Publishing Company, 1979.
- [JB82] K. Jantke and H. Beick. Combining postulates of naturalness in inductive inference. *Elektronische Informationverarbeitung und Kybernetik*, 17:465–484, 1982.
- [JLZ06] S. Jain, S. Lange, and S. Zilles. Towards a better understanding of incremental learning. In *ALT*, volume 4264 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2006.
- [Joc68] C. Jockusch. Semirecursive sets and positive reducibility. *Transactions of the AMS*, 131:420–436, 1968.
- [JORS99] S. Jain, D. Osherson, J. Royer, and A. Sharma. *Systems that Learn: An Introduction to Learning Theory*. MIT Press, Cambridge, Mass., second edition, 1999.
- [Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
- [Knu73] D. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
- [LV08] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, third edition, 2008.
- [Min76] E. Minicozzi. Some natural properties of strong identification in inductive inference. *Theoretical Computer Science*, pages 345–360, 1976.
- [MO99] F. Montagna and D. Osherson. Learning to coordinate: A recursion theoretic perspective. *Synthese*, 118:363–382, 1999.
- [Moe09] S. Moelius, 2009. Private communication.
- [MV05] P. Manolios and D. Vroon. Ordinal arithmetic: Algorithms and mechanization. *Journal of Automated Reasoning*, 34(4):387–423, 2005.
- [MY78] M. Machtey and P. Young. *An Introduction to the General Theory of Algorithms*. North Holland, New York, 1978.

- [OSW86] D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, Cambridge, Mass., 1986.
- [Pit89] L. Pitt. Inductive inference, DFAs, and computational complexity. In *Analogical and Inductive Inference, Proceedings of the Second International Workshop (AII'89)*, volume 397 of *Lecture Notes in Artificial Intelligence*, pages 18–44. Springer-Verlag, Berlin, 1989.
- [Pod74] K. Podnieks. Comparing various concepts of function prediction. *Theory of Algorithms and Programs*, 210:68–81, 1974.
- [RC94] J. Royer and J. Case. *Subrecursive Programming Systems: Complexity and Succinctness*. Research monograph in *Progress in Theoretical Computer Science*. Birkhäuser Boston, 1994.
- [Rog58] H. Rogers. Gödel numberings of partial recursive functions. *Journal of Symbolic Logic*, 23:331–341, 1958.
- [Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York, 1967. Reprinted by MIT Press, Cambridge, Massachusetts, 1987.
- [Sch91] Y. Schabes. Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 106–113. Association for Computational Linguistics, 1991.
- [Sie65] W. Sierpinski. *Cardinal and ordinal numbers*. PWN –Polish Scientific Publishers, 1965. Second revised edition.
- [SR84] G. Schäfer-Richter. *Über Eingabeabhängigkeit und Komplexität von Inferenzstrategien*. PhD thesis, RWTH Aachen, 1984.
- [SSV04] A. Sharma, F. Stephan, and Y. Ventsov. Generalized notions of mind change complexity. *Information and Computation*, 189:235–262, 2004.
- [SZ02] F. Stephan and T. Zeugmann. Learning classes of approximations to non-recursive functions. *Theoretical Computer Science*, 288:309–341, 2002.
- [vEB90] P. van Emde Boas. Machine models and simulations. In J. Van Leeuwen, editor, *Handbbook of Theoretical Computer Science. Volume A: Algorithms and Complexity*, pages 3–66. MIT Press/Elsevier, 1990.

- [WC80] K. Wexler and P. Culicover. *Formal Principles of Language Acquisition*. MIT Press, Cambridge, Mass, 1980.
- [Wei82] S. Weinstein, 1982. Private communication at the *Workshop on Learnability Theory and Linguistics*, University of Western Ontario.
- [Wie76] R. Wiehagen. Limes-erkennung rekursiver funktionen durch spezielle strategien. *Elektronische Informationverarbeitung und Kybernetik*, 12:93–99, 1976.
- [Wie78] R. Wiehagen. *Zur Theorie der Algorithmischen Erkennung*, 1978. Dissertation B, Humboldt University of Berlin.
- [WL76] R. Wiehagen and W. Liepe. Characteristic properties of recognizable classes of recursive functions. *Elektronische Informationverarbeitung und Kybernetik*, 12:421–438, 1976.
- [Yos09] R. Yoshinaka. Learning efficiency of very simple grammars from positive data. *Theoretical Computer Science*, 410:1807–1825, 2009. Special Issue for ALT’07.
- [Zeu08] T. Zeugmann, 2008. Private communication.