Inductive Inference of Σ_1^0- vs. Σ_2^0-Definitions for Computable Functions[1]

John Case and Mandayam Suraj[2]

Department of Computer and Information Sciences

University of Delaware

Newark, DE 19716

USA

February 13, 2002

[1] The first main theorem of this work preliminarily appeared in a brief abstract in [CS99]. This document contains excerpts from [Sur01].

[2] Email addresses of the authors are: case@cis.udel.edu and suraj@cis.udel.edu respectively.

Chapter 1

INDUCTIVE INFERENCE OF Σ_1^0- VS. Σ_2^0-DEFINITIONS FOR COMPUTABLE FUNCTIONS

1.1 Brief Summary of Results

This Chapter is on function learning. We recall that the basic paradigm of function learning involves a machine $\mathbf{M}$ being input the graph of a computable, number-theoretic function, and, as $\mathbf{M}$ is receiving its input, it outputs a sequence of definitions of (partial) computable functions. For success, the definitions output by the machine past some point should be definitions of the input function. Importantly, in this chapter, there are two types of output definitions featured. Each is a predicate form from the Kleene-Mostowski Arithmetical Hierarchy. The forms are Σ_1^0- and Σ_2^0-definitions. Of course (partial) computable functions are typically defined by ordinary computer programs. However, ordinary computer programs for (partial) computable functions are intercompilable with Σ_1^0-definitions for them. Moreover, a (partial) function defined by a Σ_2^0-form *can be*, but need not be, (partial) computable. In the present work the Σ_2^0-forms employed will define (partial) computable functions only.

One can think of an input to a function f as coding a possible experiment on some phenomenon, and $f(x)$ as coding the outcome of that experiment. At least a Σ_1^0-definition (or a program) for f enables the algorithmic prediction of the outcome of each experiment coded.

Case, Jain, and Sharma [CJS92] showed that, when one wants to prove (in any preassigned, computably axiomatized, true extension of first order Peano Arithmetic) *global* properties about the overall shape of computable functions, e.g., monotonicity, in *some* cases, this can be done from suitable Σ_2^0-definitions of a computable function, but it cannot be done from any Σ_1^0-definition of that computable function. They also showed that, for some classes $\mathcal{C}$ of computable functions, for some associated inductive inference machine $\mathbf{M}$, on each function $f \in \mathcal{C}$, $\mathbf{M}$ converges with its first output definition to a correct Σ_2^0-definition of f; however, for each inductive inference machine $\mathbf{M}$, for some f's in $\mathcal{C}$, $\mathbf{M}$ on f fails to converge to (a sequence of) correct Σ_1^0-definitions for f.

The first theorem in this chapter considerably strengthens these results (and another result in a later paper by Baliga, Case, Jain and Suraj [BCJS94]). This theorem yields some strong tradeoffs between inferring Σ_1^0- vs. Σ_2^0-definitions: there is a class of computable, monotone functions, $\mathcal{C}$, for which one can employ Σ_2^0-definitions, but *not* Σ_1^0-definitions for successful machine inductive inference *and* to prove monotonicity; *furthermore*, one cannot prove, from the Σ_2^0-definitions, any data points at all in the graphs of functions in $\mathcal{C}$. Hence, the output Σ_2^0-definitions are not, in principle, refutable by incorrect data point predictions, but *in principle*, they admit of being refuted by non-monotonicity (in the input data itself). This result also exhibits, then, some new subtleties re the application of Popper's Refutability or Falsifiability Principle: one cannot inductively infer $\mathcal{C}$ without being forced to accept a weaker refutability principle.

The second theorem is like the first *except* that, for the associated class $\mathcal{C}$, some machine $\mathbf{M}$ *can* successfully infer Σ_1^0-definitions for each function in $\mathcal{C}$. Here one can infer $\mathcal{C}$ without being forced to accept a weakened refutability principle; however, the cost of this would be to lose the ability to infer also the global (shape) property of monotonicity. To retain that ability, one is again forced to accept a weakened refutability principle.

1.2 Introduction

This work is within the theoretical area of machine inductive inference (or recursion-theoretic learning theory) [Put63, Gol67, KW80, CS83, AS83, OSW86, JORS99, Odi99] The reader is expected to be familiar with computability theory (or recursive function theory) as in [Rog67], but need not be familiar with recursion-theoretic inductive inference and learning theory.

The basic paradigm is illustrated by Figure 1.1.

Depiction At Discrete Time t:

$$d_t \quad \xleftarrow{\text{Out}} \quad \boxed{\mathrm{M}} \quad \xleftarrow{\text{In}} \quad (t, f(t))$$

Figure 1.1: Standard Model of Inductive Inference Machines **M**

Let $N = \{0, 1, 2, 3, \ldots\}$.

In Figure 1.1, **M** represents a machine (algorithmic device) for inductive inference. At discrete time $t \in N$, $(t, f(t))$, a data point from (the graph of) a computable function $f : N \to N$, is input to **M**, and **M** outputs a definition d_t. **M**'s success will depend on whether the d_t's converge to definitions of (close variants of) f.

What it can mean more exactly for **M** to be successful will be discussed in Section 1.3.3 below. What type of definitions (output by an inductive inference machine) are we interested in this chapter will be discussed in the present section below.

We can think of a (computable) function $f : N \to N$ as encoding the set of all possible deterministic outcomes of the experiments on some corresponding phenomenon (e.g., from chemistry): an input x to f represents the code of a possible experiment, e.g., mixing particular volumes of particular chemicals in a test tube, and $f(x)$ represents the outcome, e.g., the mixture turned blue and fizzed (see, for example, [BB75, CS83]). A definition d for the function corresponds, then, to a *explanation* for the associated phenomenon. Were d a program for f, then d would be a *predictive* explanation — since d could be used to compute, for any associated experiment x, the outcome $f(x)$.[1]

For simplicity we did not show the following in Figure 1.1. For technical purposes, we will allow $\mathbf{M}$ to initially output ?'s to indicate that it has not yet decided on its first definition to output. Once it outputs some definition, it is not allowed to output ?'s again.

Why is algorithmic inductive inference interesting — besides mathematically and as a machine model of (an aspect of) science? In a computable universe or even a universe with only computable, statistically expected behaviors[2] (e.g., a discrete quantum mechanical universe — which ours might be), computability provides a boundary around the possible behaviors of entities within that universe — including human scientists and communities thereof. Hence, under the computability assumption, the theorems in this area can also be thought of as theorems on the limitations of science. Just as a conservation assumption from physics provides boundaries on and insight into the physically possible, so too the computability assumption on scientific inference provides boundaries on and insight into the scientifically possible.

Recursion-theoretic inductive inference has received prior attention in the

---

[1] For justification of the use of *total* functions as summaries of phenomena to be investigated experimentally see [Ful85]. Fulk [Ful85] also argues that the set of distinguishable experiments *one can actually do and record* on a phenomenon is countable.

[2] For an oracle O that generates random answers but with a computable probability distribution, and, for any algorithmic procedure P which accesses O, P will have computable statistically expected input/output behavior (and constructively so) [dMSS56, Gil72, Gil77].

philosophy of science literature (e.g., [Put63, Gly85, KG89, KG90, KS95, Kel96, KSJ97, Sch99]).

The definitions output by an inductive inference machine can, in general, be definitions of partial computable functions. Of course, a definition for a computable function is a definition of a total partial computable function.

Our results concern two types of definitions of partial computable functions. We will treat a partial function as a *set* of ordered pairs, and, then, consider defining them as sets. In the Kleene-Mostowski Arithmetical Hierarchy [Rog67] we have the definitional forms Σ_n^0 and Π_n^0. Σ_0^0 and Π_0^0-forms consist of a computable predicate. The Σ_{n+1}^0-forms consist of a numerical existential quantifier in front of a Π_n^0-form, and Π_{n+1}^0-forms consist of a numerical universal quantifier in front of a Σ_n^0-form. Our results herein pertain to Σ_1^0- and Σ_2^0-definitional forms only. Σ_1^0-predicates are, then, of the form a numerical existential quantifier in front of computable predicate, and Σ_2^0-predicates are of the form a numerical existential quantifier in front of a numerical universal quantifier in front of a computable predicate.

Our first result (Theorem 1.3 in Section 1.4 below) extends partial results from [CJS92] (and a result from [BCJS94]). It refers to an arbitrary, preassigned computably (or recursively) axiomatized first order theory **T** which extends Peano Arithmetic (**PA**) and in which one cannot prove sentences that are false in the standard model ([Men97]). In this way, the set of theorems of **T** is an r.e. set; hence, there is a (not necessarily efficient) automatic theorem prover for **T**. One can think of such a prover, in the present context, as an algorithmic device for extracting correct information. When we speak informally of 'prove' below, we mean with respect to any such preassigned **T**.

Theorem 1.3 yields some strong tradeoffs between inferring Σ_1^0- vs. Σ_2^0-definitions: there is a class of computable, monotone functions, $\mathcal{C}$, for which one can employ Σ_2^0-definitions, but *not* Σ_1^0-definitions for successful machine inductive inference *and* to prove monotonicity; *furthermore*, one cannot prove, from the Σ_2^0-definitions, any data points at all in the graphs of functions in $\mathcal{C}$. Hence, the output Σ_2^0-definitions are not, in principle refutable by incorrect data point predictions, but *in principle*, they admit of being refuted by non-monotonicity (in the input data itself). This result also exhibits, then, some new subtleties re the application of Popper's Refutability or Falsifiability Principle: one cannot inductively infer $\mathcal{C}$ without being forced to accept a weaker refutability principle.[3]

Our second theorem (Theorem 1.4 also in Section 1.4 below) is like the first *except* that, for the associated class $\mathcal{C}$, some machine $\mathbf{M}$ *can* successfully infer Σ_1^0-definitions for each function in $\mathcal{C}$. Here one can infer $\mathcal{C}$ without being forced to accept a weakened refutability principle; however, the cost of this would be to lose the ability to infer also the global (shape) property of monotonicity. To retain that ability, one is again forced to accept a weakened refutability principle.

For us to rigorously state and prove our theorems it is necessary to provide mathematically precise definitions of concepts. We do this in Section 1.3 below, and, as noted above, this will include precisely defining the criteria of success of relevance to this chapter in Section 1.3.3 below.

1.3 Preliminaries

1.3.1 Notation

As above, N denotes the set of natural numbers, $\{0, 1, 2, 3, \ldots\}$. $i, j, k, m, n, p, s, t, w, x, y, z$ (with or without subscripts, superscripts, ...) range over N. $*$ denotes a non-member of N such that $(\forall n \in N)[n < * < \infty]$ ($*$ represents 'unbounded but finite'). We let a, b range over $N \cup \{*\}$.

[3] Quite different subtleties pertaining to Popper's Principle are discussed in [CS83], where it is noted, for example, that use of slightly incomplete predictive explanation, in some cases, provides more inferring power, but, in general, incomplete explanations are not refutable (see also a brief discussion of this example from [CS83] in [Odi99]).

$\emptyset$ denotes the empty set. $\in, \notin, \subseteq, \subset$ respectively denote 'is a member of', 'is not a member of', 'is a subset of', 'is a proper subset of'. $\uparrow$ denotes 'is undefined'. $\downarrow$ denotes 'is defined'.

For S, a subset of N, card(S) denotes the cardinality of S. So then, 'card$(S) \leq *$' means that card(S) is finite. max(S) and min(S) denote, respectively, the maximum and minimum of the set S, where $\max(\emptyset) = 0$ and $\min(\emptyset) = \infty$. D_x denotes the finite set with canonical index x [Rog67]. Also, for a finite set $S = \{x_0, \ldots, x_n\}$, we denote the canonical index of S by $\{\{x_0, \ldots, x_n\}\}$.

f, g, h (with or without subscripts, superscripts, ...) range over *total* functions with arguments and values from N. e, d (with or without subscripts, superscripts, ...), depending on the context, range over N or over total functions with arguments and values from N. η and ψ range over *partial* functions with arguments and values from N. $\eta(x)\downarrow$ denotes that $\eta(x)$ is defined; $\eta(x)\uparrow$ denotes that $\eta(x)$ is undefined. For η_1 and η_2 partial functions, $\eta_1(x) = \eta_2(x)$ is true iff (i) for some $y \in N$, $\eta_1(x)\downarrow = y$ and $\eta_2(x)\downarrow = y$, or (ii) $\eta_1(x)\uparrow$ and $\eta_2(x)\uparrow$. For $a \in (N \cup \{*\})$, if η_1 and η_2 are partial functions, then $\eta_1 =^a \eta_2$ means that card$(\{x \mid \eta_1(x) \neq \eta_2(x)\}) \leq a$; $\eta_1 =^\infty \eta_2$ means that card$(\{x \mid \eta_1(x) = \eta_2(x)\})$ is infinite; thus, $\eta_1 \neq^\infty \eta_2$ means that card$(\{x \mid \eta_1(x) = \eta_2(x)\})$ is finite. domain(η) and range(η) respectively denote the domain and range of the partial function η. We say that η is *monotone* $\Leftrightarrow$ $(\forall x, y \mid x < y)[\eta(x)\downarrow < \eta(y)\downarrow]$. Thus η is monotone if and only if η is a *strictly* increasing total function.

The set of all total computable functions of one variable is denoted by $\mathcal{R}$. $\mathcal{C}$, (with or without subscripts, superscripts, ...) ranges over subsets of $\mathcal{R}$.

A *pairing function* computably maps $N \times N$, 1–1 onto N [Rog67]. One can think of a pairing function as encoding pairs of natural numbers as natural numbers. In [RC94] is presented a very simple pairing function, $\langle \cdot, \cdot \rangle$, based on bit interlacing. We take our pairing function to be this one. It is computable and invertible in linear time in the lengths of its arguments. $\langle \cdot, \cdot \rangle$ is extended to map n-tuples of natural numbers 1–1 onto N in a usual way. For $n > 1$ and j such that $1 \leq j \leq n$, π_i^n's denote the corresponding inverses: i.e., $\pi_i^n(\langle x_1, \ldots, x_n \rangle) = x_i$.

The quantifier '$\forall^\infty$' means 'for all but finitely many'.

1.3.2 Useful Equivalents of Σ_1^0- and Σ_2^0-Definitions

We can consider the *acceptable programming systems* (synonym: *acceptable numberings*) [Rog58, Rog67, MY78, Roy87] as those programming systems for the partial computable functions which are intercompilable with naturally occurring general purpose programming formalisms such as Turing machine formalisms and the LISP programming language.[4]

φ^1 denotes a fixed *acceptable* programming system for the partial computable functions: $N \to N$, [Rog58, Rog67, MY78, Roy87]. φ_p^1 denotes the partial computable function computed by program p in the φ^1-system.

Φ denotes an arbitrary fixed Blum complexity measure for the φ^1-system [Blu67]. If φ^1 were based on Turing machines p, an example associated Blum complexity measure would define $\Phi_p(x)$ = the run time of Turing machine p on input x if p halts and undefined if p does not halt. Crucially, for every p, domain(φ^1) = domain(Φ), and it is algorithmically decidable, in p, x, t, whether or not $\Phi_p(x)\downarrow \leq t$ [Blu67]. We sometimes consider partial computable functions with $n > 1$ arguments in the φ^1 system. In such cases we implicitly assume that our n-ary pairing function is used to code the arguments, so, for example, $\varphi_p^1(x, y)$ stands for $\varphi_p^1(\langle x, y \rangle)$.

Next we describe the correspondences between, on the one hand, partial functions definable by Σ_1^0-formulas and those definable by ordinary φ^1-programs for

[4] Typically, for theoretical work, one works with *numerical* names for the programs in these systems — whence the term 'numbering.'

computing them, and, on the other hand, partial functions definable by Σ_2^0-formulas and those definable by so called mind changing programs (to be explained below).

Suppose $\psi = \varphi_p^1$. Then p is a φ^1-program for the partial computable function ψ. We clearly have, then, that

$$(\forall x, y)[\psi(x) = y \Leftrightarrow (\exists t)[\Phi_p(x) \leq t \ \wedge \ \varphi_p^1(x) = y]]. \tag{1.1}$$

The predicate after the '$(\exists t)$' on the right hand side of equation (1.1) is computable and a simple variant of Kleene's computable T-predicate [Kle52]. Hence, equation (1.1) witnesses that (the graph of) ψ is $\in \Sigma_1^0$.

Conversely, from [Rog67, Chapter 5], any partial function in Σ_1^0 is r.e.; hence, there is a φ^1-program for computing it.

In general Σ_1^0-definitions of partial functions are algorithmically intercompilable with φ^1-programs for those partial functions. Therefore, in general, Σ_1^0-definitions of partial functions form an acceptable system. Hence, we may informally treat ordinary programs defining partial functions as interchangeable with Σ_1^0-definitions for those partial functions. Furthermore, the results herein hold independently of which acceptable system is employed for the partial computable functions. Therefore, we can employ the φ^1-system in our formal results and their proofs without loss of generality.

Definition 1.1 ψ *is* partial lim-computable $\Leftrightarrow$

$$(\exists \textit{ computable } h)(\forall x)[\psi(x) = \lim_{t\to\infty} h(x,t)]. \tag{1.2}$$

Intuitively, in Definition 1.1, $h(x,t)$ is the output at discrete time t of a total mind changing algorithm for ψ (acting on input x). This mind changing algorithm changes its mind only finitely many times and converges to $\psi(x)$ whenever $\psi(x)$ is defined, and it changes in mind infinitely often whenever $\psi(x)$ is undefined.

Lemma 1.1 *There is a function* $\mathbf{L}$ *which is computable (in linear time in the lengths of its arguments) such that*

(a) $(\forall\psi)[\psi$ *is partial lim-computable* $\Leftrightarrow (\exists p)(\forall x)[\psi(x) = \lim_{t\to\infty}\mathbf{L}(p,x,t)]]$,

and

(b) $(\forall p \mid \varphi^1_p$ *is total*$)(\forall x)[\lim_{t\to\infty}\varphi^1_p(x,t) = \lim_{t\to\infty}\mathbf{L}(p,x,t)]$.

PROOF. This lemma straightforwardly follows from Corollary 3.12, Theorem 4.7(b), and Theorem 7.4 and its proof, all in [RC94]. □

Definition 1.2 $\varphi^2_p(x) \stackrel{\text{def}}{=} \lim_{t\to\infty}\mathbf{L}(p,x,t)]$.

As we noted earlier, we sometimes speak of p as a φ^1-program for the partial computable function φ^1_p. We shall sometimes say that p is (also) a *lim-program*, a *mind changing program*, or a φ^2*-program* for computing φ^2_p.

Clearly, by Lemma 1.1(b), we have

Lemma 1.2 *Suppose* φ^1_p *is total. Then*

$$(\forall x)[\varphi^2_p(x) = \lim_{t\to\infty}\varphi^1_p(x,t)]. \tag{1.3}$$

We have that

Proposition 1.1 $(\forall\psi)[\psi \in \Sigma_2^0 \Leftrightarrow (\exists p)[\psi = \varphi_p^2]]$.

PROOF. $(\Leftarrow)$: Suppose $\psi = \varphi_p^2$. Then, from Definition 1.2 just above, $(\forall x)[\psi(x) = \lim_{t\to\infty}\mathbf{L}(p,x,t)]$. This holds $\Leftrightarrow$

$$(\exists t)(\forall t')[t' \geq t \Rightarrow y = \mathbf{L}(p,x,t')]. \tag{1.4}$$

Since equation (1.4) is of the form a numerical existential quantifier followed by a numerical universal quantifier followed by a computable (in linear time in the lengths of its arguments) predicate, (the graph of) ψ is $\in \Sigma_2^0$.[5]

$(\Rightarrow)$: Suppose $\psi \in \Sigma_2^0$. The proof of $((3) \Rightarrow (1))$ for Theorem 1 in [Gol65][6] can, then, be straightforwardly modified to prove that ψ is partial lim-computable. Hence, by Lemma 1.1(a) and Definition 1.2, $(\exists p)[\psi = \varphi_p^2]$. ☐

Thanks to Proposition 1.1, the system of φ^2-programs essentially provides a system of Σ_2^0-definitions for the class of partial functions in Σ_2^0. It is also, for this work, a technically useful such system. From [Rog67], the partial functions in Σ_2^0 are those partial computable in an oracle for the halting problem. φ^2 is, then, in effect, a relativized system. It is also an *acceptable* relativized system essentially in the sense of [LMF76]. The results in this chapter hold independently of which acceptable systems are employed, and there is algorithmic intercompilability between such systems. Hence, for Σ_2^0, we lose no generality working with φ^2.

[5] Actually, a different and linear-time computable variant of Kleene's T-predicate can be used instead as the predicate on the right hand side of equation (1.1) above [RC94]. Of course, for the present chapter, mere computability is all that is needed regarding each of the linear-time computable predicates we have mentioned.

[6] See also [Put65] for related results and [Sha71] for connections to the work of E. Post.

For partial functions ψ definable by either Σ_2^0-formulas or lim-programs, in this chapter, we are exclusively interested in the cases where ψ is nonetheless partial computable.[7]

1.3.3 Inductive Inference Criteria of Success

If η is a partial function, and $n \in N$ is such that, for all $x < n$, $\eta(n)\downarrow$, then $\eta[n]$ denotes the finite initial segment $((0, \eta(0)), (1, \eta(1)), \ldots, (n-1, \eta(n-1)))$. Let $\text{INIT} = \{f[n] \mid f \in \mathcal{R} \text{ and } n \in N\}$.

As noted above, initially, an inductive inference machine is allowed to output ?'s to indicate that it has not decided on its first program output yet, but once it outputs some program, it is not allowed to output ?'s again.

We say that $\mathbf{M}(f)$ *converges to* p (written $\mathbf{M}(f)\downarrow = p$) $\Leftrightarrow$ $(\exists p)(\forall^\infty n)[\mathbf{M}(f[n]) = p]$; $\mathbf{M}(f)$ is undefined if no such p exists (written $\mathbf{M}(f)\uparrow$).

Next, we rigorously define the success criteria needed. We recall that $a, b \in N \cup \{*\}$.

Definition 1.3 (Gold [Gol67], Blum–Blum [BB75], Case–Smith [CS83])

(a) $\mathbf{M}$ $\mathbf{Ex}_b^a$–identifies f *(written:* $f \in \mathbf{Ex}_b^a(\mathbf{M})$*)* $\Leftrightarrow$ $[(\exists p \mid \varphi_p^1 =^a f)[\mathbf{M}(f)\downarrow = p] \wedge \ \text{card}(\{n \mid ? \neq \mathbf{M}(f[n]) \neq \mathbf{M}(f[n+1])\}) \leq b]$.

(b) $\mathbf{Ex}_b^a = \{\mathcal{C} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{C} \subseteq \mathbf{Ex}_b^a(\mathbf{M})]\}$.

We mostly write $\mathbf{Ex}_*^0$ as $\mathbf{Ex}$ and $\mathbf{Ex}_*^a$ as $\mathbf{Ex}^a$. $\mathbf{Ex}_0^0$ is also known as $\mathbf{Fin}$. As noted in [JORS99, Page 262], $\mathbf{Fin}$ is also called *finite identification* or *one-shot inference*, and was first studied in [Gol67] and [TB70].

Intuitively, a machine $\mathbf{M}$ that $\mathbf{Ex}_b^a$–identifies f, when fed the graph of f, may make no more than b 'mind changes' while converging to a final program that may

[7] Suppose $i \in \{1, 2\}$. Suppose p is a φ^i-program for a *total* function f. Since, blocks of like quantifiers can be coalesced into one quantifier of that type [Rog67], and, since for all x, y, $(x, y) \notin f \Leftrightarrow (\exists z)[f(x) = z \ \wedge \ y \neq z]$, (the graph of) f is also in Π_i^0. We make no use of this in the present chapter.

have no more than a 'anomalies' with respect to f. (See [CS83, BF74] for more about the notion of 'mind changes', and [BB75, CS83, OSW86, JORS99] for the notion of 'anomalies' in machine inductive inference.)

"**Ex**" stands for *explanatory.* Case and Smith [CS83] introduced another set of inductive inference criteria, called $\mathbf{Bc}^a$, which we define next. "**Bc**" stands for *behaviorally correct.* Barzdin [Bar74] essentially introduced the notion $\mathbf{Bc}^0$.

Definition 1.4 (Case–Smith [CS83])

(a) $\mathbf{M}$ $\mathbf{Bc}^a$*-identifies a computable function* f *(written:* $f \in \mathbf{Bc}^a(\mathbf{M})$) $\Leftrightarrow$ $(\forall^\infty n)[\varphi^1_{\mathbf{M}(f[n])} =^a f]$.

(b) $\mathbf{Bc}^a = \{\mathcal{C} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{C} \subseteq \mathbf{Bc}^a(\mathbf{M})]\}$.

Thus, unlike for the $\mathbf{Ex}^a_b$ inductive inference criteria, convergence to a single final program is *optional*, and not a requirement, for successful $\mathbf{Bc}^a$-identification. However, in order for a machine $\mathbf{M}$ to successfully $\mathbf{Bc}^a$-identify a function f, $\mathbf{M}$, on the graph of f, must eventually output programs that each compute f with no more than a anomalies.

Theorem 1.1 below gives some results about $\mathbf{Ex}^a_b$ and $\mathbf{Bc}^a$. criteria.

Theorem 1.1 *For all* $m, n \in N$

(a) $\mathbf{Ex}^{m+1}_0 - \mathbf{Ex}^m \neq \emptyset$,

(b) $\mathbf{Ex}^0_{n+1} - \mathbf{Ex}^*_n \neq \emptyset$,

(c) $\mathbf{Ex}^*_0 - \bigcup_{m \in N} \mathbf{Ex}^m \neq \emptyset$,

(d) $\mathbf{Ex}^0_* - \bigcup_{n \in N} \mathbf{Ex}^*_n \neq \emptyset$,

(e) $\mathbf{Ex}^* \subset \mathbf{Bc}$,

(f) $\mathbf{Bc}^m \subset \mathbf{Bc}^{m+1}$,

(g) $\bigcup_{m \in N} \mathbf{Bc}^m \subset \mathbf{Bc}^*$, *and*

(h) $\mathcal{R} \in \mathbf{Bc}^*$.

Parts (a), (b), (c), (d), (f) and (g) are due to Case and Smith [CS83]. John Steel first observed that $\mathbf{Ex}^* \subseteq \mathbf{Bc}$ and the diagonalization in part (e) is due to Harrington and Case [CS83]. Part (h) is due to Harrington [CS83]. Blum and Blum [BB75] first showed that $\mathbf{Ex} \subset \mathbf{Ex}^*$. Barzdin [Bar74] first showed that $\mathbf{Ex} \subset \mathbf{Bc}$.

We recall that $a, b \in N \cup \{*\}$. The following partially generalizes a definition in [CJS92]. The mind-change parameter, b, of the inductive inference criteria defined below, is new.

Definition 1.5

(a) $\mathbf{M}$ $\mathbf{LimEx}_b^a$-*identifies* f (written: $f \in \mathbf{LimEx}_b^a(\mathbf{M})) \Leftrightarrow [(\exists p \mid \varphi_p^2 =^a f)[\mathbf{M}(f)\downarrow = p] \ \wedge \ \mathrm{card}(\{n \mid ? \neq \mathbf{M}(f[n]) \neq \mathbf{M}(f[n+1])\}) \leq b]$.

(b) $\mathbf{LimEx}_b^a = \{\mathcal{C} \subseteq \mathcal{R} \mid (\exists \mathbf{M})[\mathcal{C} \subseteq \mathbf{LimEx}_b^a(\mathbf{M})]\}$.

In this chapter, we will mostly be interested in $\mathbf{LimEx}_0^0$, which we call **LimFin**.

The following are some results from [CJS92] that are relevant to this chapter

Theorem 1.2 *For all m,*

(a) $\mathbf{Ex} \subset \mathbf{LimEx}$

(b) $\mathbf{LimEx} - \mathbf{Bc}^m \neq \emptyset$,

(c) $\mathbf{Bc} - \mathbf{LimEx}^* \neq \emptyset$.

We note that the proofs of Parts (a) and (b) of Theorem 1.2, as presented in [CJS92], show that, in fact, $\mathbf{Fin} \subset \mathbf{LimFin}$ and for all m, $\mathbf{LimFin} - \mathbf{Bc}^m \neq \emptyset$.

1.4 Results

As mentioned earlier, we significantly extend results in [CJS92] and [BCJS94]. In these papers, results are presented for extensions of first order arithmetic. We proceed informally, as in [CJS92] and [BCJS94], with respect to expressing propositions in first order arithmetic, as outlined next.

If E is an expression such as '$\Phi_i \leq t$', or 'φ_i^1 is total', we shall write $\ll \mathsf{E} \gg$ to denote a naturally corresponding, fixed standard wff (well-formed formula) of first order arithmetic [Men86] which (semantically) expresses E. We need and assume that

> if E' is obtained from E by changing some numerical values, then $\ll \mathsf{E}' \gg$ can be *algorithmically* obtained from those changed numerical values and $\ll \mathsf{E} \gg$.

It is understood that, if E contains references to particular partial functions defined by particular programs or lim-programs, then these particular programs or lim-programs are referred to for them. It is well known that wffs extensionally equivalent (with respect to standard models) may not be intensionally or provably equivalent [Fef60]. In what follows, when we use the $\ll \mathsf{E} \gg$ notation, it will always be for propositions that are easily seen to be (semantically) expressible in first order arithmetic. '$\vdash$' denotes the provability relation.

Next is our first theorem of this chapter. It amalgamates, in a new way, separate and disjoint results from [CJS92, BCJS94]. It also adds the completely new Clause 3b which is absolutely crucial for the theorem's interpretation for philosophy of science.[8]

Theorem 1.3 *Suppose* $\mathbf{T}$ *is a computably axiomatizable first order theory which extends Peano Arithmetic* ($\mathbf{PA}$) *and in which one cannot prove sentences that are false in the standard model* ([Men97]). *Then, there is a class of monotone* computable *functions* $\mathcal{C}$ *such that*

[8] Clause 3c first appeared in a result from [BCJS94] and was based on a question of Grigori Schwartz. It's function, in the present context, is to provide an additional, conceptually subtle, and perhaps somewhat surprising extra condition.

1. $\mathcal{C} \notin \cup_{k \in N} \mathbf{Bc}^k$,

2. $(\forall f \in \mathcal{C})(\forall p \mid \varphi^1_p =^\infty f)[\mathbf{T} \nvdash \ll \varphi^1_p$ *is monotone* $\gg]$,[9] *and*

3. *there exists a machine* $\mathbf{M}$, *such that* $\mathcal{C} \subseteq \mathbf{LimFin}(\mathbf{M})$, *and, for every* $f \in \mathcal{C}$, *for every* e *such that* $\mathbf{M}(f) = e$,

   (a) $\mathbf{PA} \vdash \ll \varphi^2_e$ *is monotone* $\gg$,

   (b) $(\forall x, y)[\mathbf{T} \nvdash \ll \varphi^2_e(x) = y \gg]$[10], *and*

   (c) $\mathbf{PA} \vdash \ll \varphi^2_e$ *is computable* $\gg$.

This theorem, then, shows some strong tradeoffs between inferring Σ^0_1- vs. Σ^0_2-definitions. For $\mathcal{C}$, one can employ the latter, but *not* the former for successful inference *and* to prove monotonicity. But, by the important Clause 3b, one cannot predict, from the latter and $\mathbf{T}$, for the sake of Popper's Refutability Criterion for science, any data points at all in the graphs of the φ^2_e's!

We note that we cannot strengthen Theorem 1.3 by using, instead of Clause 1, $\mathcal{C} \notin \mathbf{Bc}^*$, since by Theorem 1.1, $\mathcal{R} \in \mathbf{Bc}^*$. In any case, from the analysis in [CCJ01], the $\mathbf{Bc}^*$ criterion is not as useful (as the other inductive inference criteria introduced in this chapter), from the standpoint of scientific inference.

PROOF OF THEOREM 1.3.

Suppose the hypotheses. Fix an automatic theorem prover for $\mathbf{T}$. In what follows, any reference to proving some formula in $\mathbf{T}$ within so many steps refers to steps in the execution of this automatic theorem prover. For a formula $\ll \mathsf{E} \gg$, we write '$\mathbf{T} \vdash_{\leq t} \ll \mathsf{E} \gg$' to denote '$\mathbf{T}$ proves $\ll \mathsf{E} \gg$ in less than or equal to t steps

---

9 *Therefore,* $(\forall f \in \mathcal{C})(\forall p \mid \varphi^1_p =^\infty f$ *and* φ^1_p *is monotone* $)[\mathbf{T} \nvdash \ll \varphi^1_p$ *is monotone* $\gg]$. *This is, perhaps, surprisingly strong.*

10 *Therefore,* $(\forall x, y \mid \varphi^2_e(x) = y)[\mathbf{T} \nvdash \ll \varphi^2_e(x) = y \gg]$.

(of the above fixed theorem prover)'. We write '$\mathbf{T} \vdash_{=t} \ll \mathsf{E} \gg$' to denote '$\mathbf{T}$ proves $\ll \mathsf{E} \gg$ in exactly t steps (of the above fixed theorem prover)'.

Definition 1.6 *Suppose $g \in \mathcal{R}$. Then f is* the g–extension of $\eta \Leftrightarrow$

$$f(x) = \begin{cases} \eta(x), & \textit{if } x \in \mathrm{domain}(\eta); \\ g(x), & \textit{otherwise.} \end{cases}$$

We note that for any $g, h \in \mathcal{R}$, the g–extension of h is h itself. Also, for any η, such that domain(η) is a finite initial segment of N, for any $g \in \mathcal{R}$, the g–extension of η is a computable function.

Let $\mathbf{M}_0, \mathbf{M}_1, \ldots$ be a recursively enumerable sequence of total computable inductive inference machines such that $(\forall \mathcal{C} \in \cup_{k \in N} \mathbf{Bc}^k)(\exists k, n)[\mathcal{C} \subseteq \mathbf{Bc}^k(\mathbf{M}_n)]$. From [JORS99, §4.2.5, pages 68-69], such a sequence of machines exists.

N.B. 1.1 *In this proof, wherever we use φ without any superscript, we mean φ^1.*

We next define a mapping V from N to canonical indices of finite sets.

begin procedure for defining V (on input x).

if $(\forall i \leq x)[(\mathbf{T} \vdash_{\leq x} \ll \varphi_i \text{ is monotone} \gg) \Rightarrow (\exists t)(\forall w \leq x)[\Phi_i(w) \leq t]]$

then

$$V(x) = \{\{\varphi_i(x) \mid i \leq x \ \wedge \ \mathbf{T} \vdash_{\leq x} \ll \varphi_i \text{ is monotone} \gg\}\}$$

else

$V(x)\uparrow$

endif

end procedure for defining V.

N.B. 1.2 *For convenience, we later abuse notation and treat V as a mapping from N to finite sets; specifically, for each x, we write $V(x)$ when we really mean $D_{V(x)}$.*

We will let p_V be the φ^1-program that the defining procedure for V compiles into.[11] Therefore,

N.B. 1.3 *When we refer to V in the context of* **PA** *or extensions thereof, it should be understood that V is represented by procedure p_V.*

We note that by our hypothesis that **T** does not prove anything that is false in the standard model, the **if** clause in the procedure defining V is always true. Therefore, V is a computable mapping from N to canonical indices. However, that argument can*not* be even expressed in **PA** (by Tarski's Theorem [Men97]). The reason we define V using the specific conditions above is so that we *can* get the Lemma 1.3 just below.

Lemma 1.3 $\mathbf{PA} \vdash \ll \mathrm{domain}(V)$ *is N or is a finite initial segment of N* $\gg$.

Proof. In **PA**, we can prove (non-constructively) that either domain(V) is N or not. Hence, we can, then, do the following (non-constructive) case analysis *within* **PA**. If domain(V) is N, then we are done.

So suppose (in **PA**) domain(V) is not N. Then, we can prove in **PA** that there exists an x, such that, $V(x)\uparrow$ and for all $x' < x$, $V(x')\downarrow$.

Next, we can prove within **PA** that for all $x' < x$, $V(x')\downarrow \wedge (\exists i \leq x)(\forall t)(\exists w \leq x)[\Phi_i(w) > t]$. Thus, we can also prove within **PA** that, for any $y \geq x$, $(\exists i \leq y)(\forall t)(\exists w \leq y)[\Phi_i(w) > t]$.

Hence, we can prove within **PA** that domain(V) is a finite initial segment of N. $\square$

Also, for all x, $\mathrm{card}(V(x)) \leq x + 1$.

---

[11] Every ordinary program can be compiled into an equivalent φ^1-program — since φ^1 is an acceptable system [Rog67].

We define a mapping S from $(N \times N)$ to canonical indices as follows. For all e, x,

$$S(e,0) \stackrel{\text{def}}{=} \{\{e, e+1, e+2\}\};$$
$$\text{for } x > 0, S(e,x) \stackrel{\text{def}}{=} \{\{x' \mid \max(S(e,x-1)) < x' \leq \max(S(e,x-1)) + x + 3\}\}.$$

N.B. 1.4 *For convenience, we later abuse notation and treat S as a mapping from $(N \times N)$ to finite sets; specifically, for each e, x, we write $S(e,x)$ when we really mean $D_{S(e,x)}$.*

Also, a note similar to N.B. 1.3 holds for S too. We note that

$$\text{For all } e, x, \text{ card}(S(e,x)) = x + 3. \tag{1.5}$$

Also,

$$\text{For all } e, x, \max(S(e,x)) < \min(S(e,x+1)). \tag{1.6}$$

For convenience we employ an implicit application the Operator Recursion Theorem [Cas74, Cas94] to get the existence of computable *functions* e_0, d, e_1, and e_2 such that for each n, z, the partial functions $\varphi_{e_0(n)}, \varphi_{d(n,z)}, \varphi_{e_1(n)}$ and $\varphi_{e_2(n)}$ are defined as follows.[12]

For all x,

$$\varphi_{e_0(n)}(x) \stackrel{\text{def}}{=} \begin{cases} \min(S(e_2(n), x) \setminus V(x)), & \text{(i) if } V(x)\downarrow; \\ \uparrow, & \text{(ii) otherwise.} \end{cases} \tag{1.7}$$

Since V is computable, we note that for each n, $\varphi_{e_0(n)}$ is (total) computable. Clause (ii) of equation (1.7) is there only to help with later reasoning within **PA**. We also note that for all x, $\varphi_{e_0(n)}(x) \notin V(x)$.

$\varphi_{d(n,z)} \stackrel{\text{def}}{=}$ the $\varphi_{e_0(n)}$–extension of $\varphi_{e_1(n)}[z]$, if for all $z' < z$, $\varphi_{e_1(n)}(z')\downarrow$; $\stackrel{\text{def}}{=}$ the everywhere undefined function, otherwise.

[12] The Operator Recursion Theorem allows for multiple self and other reference between the parameterized programs.

The $\varphi_{d(n,z)}$'s that we use in the definition of $\varphi_{e_1(n)}$'s below will actually turn out to be (total) computable. Also, we note that, for such $\varphi_{d(n,z)}$'s, for all n, x and z, such that $x \geq z$, we get that $\varphi_{d(n,z)}(x) = \varphi_{e_0(n)}(x)$.

$\varphi_{e_1(n)}$ is defined in stages as follows. Let $\varphi_{e_1(n)}(0) = e_2(n)$. Let x_s denote the least x such that $\varphi_{e_1(n)}(x)$ has not been defined before stage s. Go to stage 0.

Begin Stage s.

1. Search for $m, y_1, y_2, \ldots, y_{s+1}$ such that[13]
   - (a) $y_{s+1} > y_s > \ldots > y_1 \geq m \geq x_s$,
   - (b) for all $x \leq y_{s+1}, \varphi_{d(n,x_s)}(x)\downarrow$,
   - (c) $\mathbf{M}_n(\varphi_{d(n,x_s)}[m])\downarrow = p$ (say),
   - (d) for each i, such that $1 \leq i \leq s+1$, $\varphi_p(y_i)\downarrow$, and
   - (e) for all $x \leq y_{s+1}$, $V(x)$ is defined.
2. If and when such y_i's and m are found, let:
   - 2.1 $\varphi_{e_1(n)}(y_i) =$ the least $z \in [S(e_2(n), y_i) \backslash (\{\varphi_p(y_i)\} \cup V(y_i))]$, for $1 \leq i \leq s+1$.
   - 2.2 $\varphi_{e_1(n)}(x) = \varphi_{d(n,x_s)}(x)$, for $x_s \leq x \leq y_{s+1}$ and $x \notin \{y_1, y_2, \ldots, y_{s+1}\}$.
3. Go to stage $s+1$.

End Stage s.

[13] Parts 1(b) and 1(e) are only there to help with later reasoning in **PA**.

Finally, we define $\varphi_{e_2(n)}$'s as follows. For all t, x,

$$\varphi_{e_2(n)}(x,t) \stackrel{\text{def}}{=} \begin{cases} \min(S(e_2(n),x) \setminus \{y_0\}), & \text{(i) if } (\exists w \leq x)(\exists y \leq t)[\mathbf{T} \vdash_{\leq t} \\ & \ll \varphi^2_{e_2(n)}(w) = y \gg] \text{ and } y_0 \text{ is} \\ & \text{the least such } y \text{ found for the} \\ & \text{least such } w \text{ found;} \\ \min(S(e_2(n),x)), & \text{(ii) if } (\forall w \leq x)(\forall y \leq t)[\mathbf{T} \not\vdash_{\leq t} \\ & \ll \varphi^2_{e_2(n)}(w) = y \gg] \wedge \\ & \Phi_{e_1(n)}(x) > t \ \wedge\ \Phi_{e_0(n)}(x) > t; \\ \varphi_{e_1(n)}(x), & \text{(iii) if } (\forall w \leq x)(\forall y \leq t)[\mathbf{T} \not\vdash_{\leq t} \\ & \ll \varphi^2_{e_2(n)}(w) = y \gg] \wedge \\ & \Phi_{e_1(n)}(x) \leq t; \\ \varphi_{e_0(n)}(x), & \text{(iv) otherwise.} \end{cases} \tag{1.8}$$

We note that all $\varphi_{e_2(n)}$'s are total. Therefore, from Lemma 1.2, we get

$$\text{For all } n, x, \varphi^2_{e_2(n)}(x) = \lim_{t\to\infty} \varphi_{e_2(n)}(x,t). \tag{1.9}$$

We note that for each n, $\varphi_{e_1(n)}$ is partial computable and has domain either N or a finite initial segment of N. Let f_n = the $\varphi_{e_0(n)}$–extension of $\varphi_{e_1(n)}$. Let $\mathcal{C} = \{f_n \mid n \in N\}$. Clearly, $\mathcal{C} \subseteq \mathcal{R}$.

The rest of the proof involves ensuring, via a series of claims, that the many pieces of Theorem 1.3 hold.

Claim 1.1 $\mathcal{C} \notin \cup_{k\in N}\mathbf{Bc}^k$.

Proof. Suppose by way of contradiction that there is a machine $\mathbf{M}_n$ such that $\mathcal{C} \subseteq \cup_{k\in N}\mathbf{Bc}^k(\mathbf{M}_n)$. We consider $f = f_n \in \mathcal{C}$. We recall that f_n is the $\varphi_{e_0(n)}$–extension of $\varphi_{e_1(n)}$. From the construction of $\varphi_{e_1(n)}$, only the following two cases are possible.

Case 1: All stages are entered and subsequently terminate.

In this case, $\varphi_{e_1(n)}$ is total and therefore $f = \varphi_{e_1(n)}$. Also, in the definition of $\varphi_{e_1(n)}$, step 1 succeeds in each stage. Thus, by the diagonalization in step 2, for all k, for infinitely many m, $\varphi_{\mathbf{M}_n(f[m])} \neq^k f$. Therefore, $f_n \notin \cup_{k\in N}\mathbf{Bc}^k(\mathbf{M}_n)$, a contradictions.

Case 2: Some stage s is entered and never subsequently terminates.

Clearly, domain($\varphi_{e_1(n)}$) is a finite initial segment of N. Now, for all but finitely many m, for all but finitely many x, $\varphi_{\mathbf{M}_n(f[m])}(x)\uparrow$ (otherwise, the search in step 1 would succeed). Therefore, $f_n \notin \cup_{k\in N}\mathbf{Bc}^k(\mathbf{M}_n)$, a contradiction.

From the above cases, it follows that $\mathcal{C} \notin \cup_{k\in N}\mathbf{Bc}^k$. □

Claim 1.2 *For all $f \in \mathcal{C}$, for all p such that $\varphi_p =^\infty f$, $\mathbf{T} \not\vdash \ll \varphi^1_p$ is monotone $\gg$.*

PROOF. Suppose by way of contradiction that $(\exists f \in \mathcal{C})(\exists p \mid \varphi_p =^\infty f)[\mathbf{T} \vdash \ll \varphi^1_p$ is monotone $\gg]$.

Let t be such that $\mathbf{T} \vdash_{=t} \ll \varphi^1_p$ is monotone $\gg$. Let n be such that $f = f_n$, the $\varphi_{e_0(n)}$–extension of $\varphi_{e_1(n)}$.

We now consider the same cases as in Claim 1.1, but this time in order to show that for all x, $f(x) \notin V(x)$.

Case 1: All stages are entered and subsequently terminate.

In this case, f is identical to $\varphi_{e_1(n)}$. Also, for all x, $\varphi_{e_1(n)}(x)$ gets defined in step 2.1 or step 2.2 of the construction. Clearly, if $\varphi_{e_1(n)}(x)$ get defined in step 2.1, the value assigned to $\varphi_{e_1(n)}(x) \notin V(x)$. If, on the other hand, $\varphi_{e_1(n)}(x)$ gets defined in step 2.2, $\varphi_{e_1(n)}(x) = \varphi_{d(n,x_s)}(x)$. But, since $x \geq x_s$, $\varphi_{d(n,x_s)}(x) = \varphi_{e_0(n)}(x) \notin V(x)$. Hence, for all x, $f(x) \notin V(x)$.

Case 2: Some stage s is entered and never subsequently terminates.

Then, f is identical to $\varphi_{d(n,x_s)}$, which is the $\varphi_{e_0(n)}$–extension of $\varphi_{e_1(n)}[x_s]$. Also, for all $x \in$ domain($\varphi_{e_1(n)}$), as in Case 1, $f(x) \notin V(x)$. For all $x \notin$ domain($\varphi_{e_1(n)}$), $f(x) = \varphi_{d(n,x_s)}(x) = \varphi_{e_0(n)}(x) \notin V(x)$. Hence, for all x, $f(x) \notin V(x)$.

Therefore, for all x, $f(x) \notin V(x)$. Also, φ_p is total. We observe, however, that for all $x \geq \max(\{p,t\})$, $\varphi_p(x) \in V(x)$. Thus, for all but finitely many x, $\varphi_p(x) \neq f(x)$, a contradiction. $\square$

Claim 1.3 *For all n, x, y, $\mathbf{T} \not\vdash \ll \varphi^2_{e_2(n)}(x) = y \gg$.*

Proof. Suppose by way of contradiction that n, x and y are such that $\mathbf{T} \vdash \ll \varphi^2_{e_2(n)}(x) = y \gg$. Furthermore, for this n, let w be the least number less than or equal to x for which there exists a y_0 such that $\mathbf{T} \vdash \ll \varphi^2_{e_2(n)}(w) = y_0 \gg$. By our assumption that $\mathbf{T}$ does not prove anything that is false, for n and w as above, y_0 is unique. Let t be such that $\mathbf{T} \vdash_{=t} \ll \varphi^2_{e_2(n)}(w) = y_0 \gg$.

Therefore, from equation (1.8), for all $t' \geq \max(\{t, y_0\})$, $\varphi_{e_2(n)}(w, t') = \min(\{S(e_2(n), x) \setminus \{y_0\}\})$. Hence $\varphi^2_{e_2(n)}(w) \neq y_0$, a contradiction. $\square$

Claim 1.4 *For all n, $\varphi^2_{e_2(n)} = f_n$.*

Proof. Consider any n. By Claim 1.3, Clause (i) of equation (1.8) cannot apply for any x, t. Therefore, clearly, using equations (1.5) and (1.9), for $x \in \mathrm{domain}(\varphi_{e_1(n)})$, $\varphi^2_{e_2(n)}(x) = \varphi_{e_1(n)}(x)$. Since $\varphi_{e_0(n)}$ is computable, for $x \notin \mathrm{domain}(\varphi_{e_1(n)})$, $\varphi^2_{e_2(n)}(x) = \varphi_{e_0(n)}(x)$. Since $\mathrm{domain}(\varphi_{e_1(n)})$ can only be either N or a finite initial segment of N, the claim follows. $\square$

Claim 1.5 *There exists a machine $\mathbf{M}$ such that $\mathcal{C} \subseteq \mathbf{LimFin}(\mathbf{M})$.*

Proof. We recall that an inductive inference machine is allowed to output ?'s initially, before, if ever, it decides to output a program (number in N).

For all $f \in \mathcal{R}$, $n \in N$,

$$\mathbf{M}(f[n]) = \begin{cases} ?, & \text{if } n = 0; \\ f(0), & \text{otherwise.} \end{cases}$$

Suppose $f = f_n \in \mathcal{C}$. We recall that f_n is the $\varphi_{e_0(n)}$–extension of $\varphi_{e_1(n)}$. From the definition of $\varphi_{e_1(n)}$, $f(0) = e_2(n)$. By Claim 1.4, $\varphi^2_{e_2(n)} = f$. So, $\varphi^2_{\mathbf{M}(f)} = f$. Also, $\mathrm{card}(\{n \mid ? \neq \mathbf{M}(f[n]) \neq \mathbf{M}(f[n+1])\}) = 0$. Therefore, $\mathcal{C} \subseteq \mathbf{LimFin}(\mathbf{M})$.

□

M from the proof of Claim 1.5 just above is the one we use to prove Clause 3 of this theorem. We note that Claim 1.3, then, applies to the φ^2-programs that **M** converges to on functions in $\mathcal{C}$. Similar observations are true for Claims 1.6 and 1.8 below. We need the following Fact for proving Claims 1.6 and 1.8 below.

Fact 1.1

$$\mathbf{PA} \vdash \ll S \textit{ is computable} \gg \tag{1.10}$$

$$\mathbf{PA} \vdash \ll (\forall e, x)[\mathrm{card}(S(e,x)) = x + 3] \gg \tag{1.11}$$

$$\mathbf{PA} \vdash \ll (\forall e, x)[\max(S(e,x)) < \min(S(e, x+1))] \gg \tag{1.12}$$

$$\mathbf{PA} \vdash \ll (\forall x)[x \in \mathrm{domain}(V) \Rightarrow \mathrm{card}(V(x)) \leq x + 1] \gg \tag{1.13}$$

$$\mathbf{PA} \vdash \ll (\forall n)[\mathrm{domain}(\varphi_{e_0(n)}) \textit{ is } N \textit{ or is a finite initial segment of } N] \gg \tag{1.14}$$

$$\mathbf{PA} \vdash \ll (\forall n, x)[x \in \mathrm{domain}(\varphi_{e_0(n)}) \Rightarrow \varphi_{e_0(n)} \in S(e_2(n), x) \gg \tag{1.15}$$

$$\mathbf{PA} \vdash \ll (\forall n, x, z)[x \in \mathrm{domain}(\varphi_{d(n,z)}) \ \wedge \ x \geq z \Rightarrow \varphi_{d(n,z)}(x) = \varphi_{e_0(n)}(x) \gg \tag{1.16}$$

$$\mathbf{PA} \vdash \ll (\forall n)[\mathrm{domain}(\varphi_{e_1(n)}) \textit{ is } N \textit{ or is a finite initial segment of } N] \gg \tag{1.17}$$

Proof.

Each of the assertions in Fact 1.1 can be proved using techniques similar to those used in the proof of Lemma 1.3. We omit the details. □

Claim 1.6 *For all n, $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ is monotone $\gg$.*[14]

Proof. We will first show that for all n, $\mathbf{PA} \vdash \ll (\forall x)[\varphi^2_{e_2(n)}(x) \in S(e_2(n), x)] \gg$. Then, the claim follows from equation (1.12).

Consider any n. Given any x, we consider the following (non-constructive) case analysis based on the definition of $\varphi_{e_2(n)}$ in equation (1.8). The case analysis below can be done within $\mathbf{PA}$, but we omit the details of how this may be done within $\mathbf{PA}$.

Case 1: $(\exists t)(\exists w \leq x)(\exists y \leq t)[\mathbf{T} \vdash_{\leq t} \ll \varphi^2_{e_2(n)}(w) = y \gg]$.

Let w be the least value for which there exists y such that $\mathbf{T} \vdash \ll \varphi^2_{e_2(n)}(w) = y \gg$. Also, let y_0 be the least value such that $\mathbf{T} \vdash \ll \varphi^2_{e_2(n)}(w) = y \gg$. Let t be the least value such that $\mathbf{T} \vdash_{\leq t} \ll \varphi_{e_2(n)}(w) = y_0 \gg$.

Therefore, from Clause (i) of equation (1.8), for all $t' \geq \max(\{w, t\})$, $\varphi_{e_2(n)}(x, t') = \min(S(e_2(n), x) \setminus \{y_0\})$, where y_0 is as found just above. Using equations (1.10) and (1.11), we can show that $\varphi^2_{e_2(n)}(x){\downarrow} \in S(e_2(n), x)$.

Case 2: Not Case 1 and $(\forall t)[\Phi_{e_1(n)}(x) > t]$ and $(\forall t)[\Phi_{e_0(n)}(x) > t]$.

Clearly, in this case, for all t, $\varphi_{e_2(n)}(x, t) = \min(S(e_2(n), x))$. Hence, $\varphi^2_{e_2(n)}(x){\downarrow} \in S(e_2(n), x)$.

Case 3: Not Case 1 and $(\exists t)[\Phi_{e_1(n)}(x) \leq t]$.

Therefore, for all $t' \geq t$, $\varphi_{e_1(n)}(x){\downarrow}$ and $\varphi_{e_2(n)}(x, t') = \varphi_{e_1(n)}(x)$. Hence, $\varphi^2_{e_2(n)}(x){\downarrow} = \varphi_{e_1(n)}(x)$. Also, from the definition of $\varphi_{e_1(n)}$, if $\varphi_{e_1(n)}(x){\downarrow}$, then it is assigned a value in step 2.1 or step 2.2 of some stage s in the construction of $\varphi_{e_1(n)}$. If it is assigned a value in step 2.1, then clearly, $\varphi_{e_1(n)}(x) \in S(e_2(n), x)$. If, on the other hand, it is assigned a value in step 2.2, then $\varphi_{e_1(n)}(x){\downarrow} = \varphi_{d(n,x_s)}(x)$. But, since $x \geq x_s$, using equations (1.16) and (1.15), we get $\varphi_{d(n,x_s)}(x) = \varphi_{e_0(n)}(x)$, and, $\varphi_{e_0(n)}(x) \in S(e_2(n), x)$. Therefore, $\varphi^2_{e_2(n)}(x){\downarrow} \in S(e_2(n), x)$.

[14] *In fact, we can prove that $\mathbf{PA} \vdash \ll (\forall n)[\varphi^2_{e_2(n)}$ is monotone$] \gg$.*

Case 4: Not Case 1 and $(\forall t)[\Phi_{e_1(n)}(x) > t]$ and $(\exists t)[\Phi_{e_0(n)}(x) \leq t]$.

In this case, for all $t' \geq t$, $\varphi_{e_2(n)}(x, t') = \varphi_{e_0(n)}(x)$. Also, from equation (1.15), $\varphi_{e_0(n)}(x) \in S(e_2(n), x)$. Therefore, $\varphi^2_{e_2(n)}(x)\downarrow \in S(e_2(n), x)$.

By the above case analysis, $\varphi^2_{e_2(n)}(x) \in S(e_2(n), x)$. Therefore, by equation (1.12), $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ is monotone $\gg$. $\square$

As a corollary to Claim 1.6, we have

Claim 1.7 *For all n, $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ is total $\gg$.*

Claim 1.8 *For all n, $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ is computable $\gg$.*[15]

Proof. Given $n \in N$, we consider the following (non-constructive) case analysis based on the definition of $\varphi_{e_2(n)}$. Using the definition of $\varphi_{e_2(n)}$, equations (1.14) and (1.17), the following (non-constructive) case analysis can be done within $\mathbf{PA}$. Once again, we omit the details of how this may be done within $\mathbf{PA}$.

Case 1: $(\exists x)(\exists t)(\exists w \leq x)(\exists y \leq t)[\mathbf{T} \vdash_{\leq t} \ll \varphi^2_{e_2(n)}(w) = y \gg]$.

Using an analysis similar to that in Case 1 of Claim 1.6, we can prove that there exists y_0, such that for all $x' \geq x$, $\varphi^2_{e_2(n)}(x') = \min(S(e_2(n), x') \setminus \{y_0\})$. Then, using equations (1.10) and (1.11), and Claim 1.7, we can see that $\varphi^2_{e_2(n)}$ is computable.

Case 2: Not Case 1 and domain$(\varphi_{e_1(n)})$, domain$(\varphi_{e_0(n)})$ are finite initial segments of N.

In this case, for all but finitely many x, for all t, Clause (ii) of equation (1.8) holds for $\varphi_{e_2(n)}(x, t)$. Hence, for all but finitely many x, $\varphi^2_{e_2(n)}(x) = \min(S(e_2(n), x))$. Again, using equations (1.10) and (1.11), and Claim 1.7, we can see that $\varphi^2_{e_2(n)}$ is computable.

---

[15] *We can prove that $\mathbf{PA} \vdash \ll (\forall n)[\varphi^2_{e_2(n)}$ is computable$] \gg$.*

Case 3: Not Case 1 and domain($\varphi_{e_1(n)}$) is N.

We can show that for every x, for all but finitely many t, Clause (iii) of equation (1.8) holds. Hence, we can show that $\varphi^2_{e_2(n)} = \varphi_{e_1(n)}$. Also, from the definition of $\varphi_{e_1(n)}$, we can show that $\varphi_{e_1(n)}$ is computable. Therefore, $\varphi^2_{e_2(n)}$ is computable.

Case 4: Not Case 1 and domain($\varphi_{e_1(n)}$) is a finite initial segment of N and domain($\varphi_{e_0(n)}$) is N.

In this case, there exists an x such that for all $x' \geq x$, $\varphi^2_{e_2(n)}(x') = \varphi_{e_0(n)}(x)$. Also, from the definition of $\varphi_{e_0(n)}$, we can show that $\varphi_{e_0(n)}$ is computable. Hence, $\varphi^2_{e_2(n)}$ is computable.

Thus, by the above case analysis, we conclude that $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ is computable $\gg$. $\square$

From Claims 1.1–1.8, the theorem follows. $\square$ FOR THEOREM 1.3

Theorem 1.4 *Suppose* $\mathbf{T}$ *is an axiomatizable first order theory which extends Peano Arithmetic* ($\mathbf{PA}$) *and in which one cannot prove sentences that are false in the standard model. Then, there is a class of monotone* computable *functions* $\mathcal{C}$ *such that*

1. $\mathcal{C} \in \mathbf{Fin}$,

2. $(\forall f \in \mathcal{C})(\forall p \mid \varphi^1_p =^\infty f)[\mathbf{T} \nvdash \ll \varphi^1_p$ *is monotone* $\gg]$,[16] *and*

3. *there exists a machine* $\mathbf{M}$, *such that* $\mathcal{C} \subseteq \mathbf{LimFin}(\mathbf{M})$, *and, for every* $f \in \mathcal{C}$, *for every* e *such that* $\mathbf{M}(f) = e$,

   (a) $\mathbf{PA} \vdash \ll \varphi^2_e$ *is monotone* $\gg$,

[16] *Therefore,* $(\forall f \in \mathcal{C})(\forall p \mid \varphi^1_p =^\infty f$ *and* φ^1_p *is monotone* $)[\mathbf{T} \nvdash \ll \varphi^1_p$ *is monotone* $\gg]$.

(b) $(\forall x, y)[\mathbf{T} \not\vdash \ll \varphi_e^2(x) = y \gg]$[17], *and*

(c) $\mathbf{PA} \vdash \ll \varphi_e^2$ *is computable* $\gg$.

Theorem 1.4 shows that there exists a class of function $\mathcal{C}$, such that, one can infer *both* Σ_1^0- and Σ_2^0-definitions for each function in $\mathcal{C}$; however, for such definitions inferred, one can prove monotonicity *only* for the Σ_2^0-definitions and predict data points *only* for the Σ_1^0-definitions. As noted above, here one *can* infer $\mathcal{C}$ without being forced to accept a weakened refutability principle; however, the cost of this would be to lose the ability to infer also the global (shape) property of monotonicity. To retain that ability, one is again forced to accept a weakened refutability principle.

Proof of theorem 1.4.

The proof of this Theorem is quite similar in many respects to the proof of Theorem 1.3. However, it is much clearer to present it than to say only where the two proofs differ.

Suppose the hypotheses. Fix an automatic theorem prover for $\mathbf{T}$. In what follows, any reference to proving some formula in $\mathbf{T}$ within so many steps refers to steps in the execution of this automatic theorem prover.

In this proof also, wherever we use φ without any superscript, we mean φ^1. Let V and S be defined as in the proof of Theorem 1.3.

Therefore, we still have[18]

$$\text{For all } e, x,\ \mathrm{card}(S(e,x)) = x + 3. \tag{1.18}$$

Also,

$$\text{For all } e, x,\ \max(S(e,x)) < \min(S(e,x+1)). \tag{1.19}$$

[17] *Therefore,* $(\forall x, y \mid \varphi_e^2(x) = y)[\mathbf{T} \not\vdash \ll \varphi_e^2(x) = y \gg]$.

[18] For this proof, we note that is sufficient that $\mathrm{card}(S(e,x)) = x + 2$.

By the implicit use of a Kleene Parametric Double Recursion Theorem [Rog67], there exist computable functions e_0 and e_2 such that for each n, the partial functions $\varphi_{e_0(n)}$ and $\varphi_{e_2(n)}$ are defined as follows.

Firstly, for all n, let $e_n \stackrel{\text{def}}{=} \langle e_0(n), e_2(n) \rangle$.

For all x,

$$\varphi_{e_0(n)}(x) \stackrel{\text{def}}{=} \begin{cases} \min(S(e_n, x) \setminus V(x)), & \text{(i) if } V(x)\downarrow; \\ \uparrow, & \text{(ii) otherwise.} \end{cases} \tag{1.20}$$

Since V is computable, we note that for each n, $\varphi_{e_0(n)}$ is (total) computable. Clause (ii) of equation (1.20) is there only to help with later reasoning in **PA**. We also note that for all x, $\varphi_{e_0(n)}(x) \notin V(x)$.

For all t, x,

$$\varphi_{e_2(n)}(x,t) \stackrel{\text{def}}{=} \begin{cases} \min(S(e_n, x) \setminus \{y_0\}), & \text{(i) if } (\exists w \leq x)(\exists y \leq t)[\mathbf{T} \vdash_{\leq t} \\ & \ll \varphi^2_{e_2(n)}(w) = y \gg] \text{ and } y_0 \text{ is} \\ & \text{the least such } y \text{ found for the} \\ & \text{least such } w \text{ found;} \\ \min(S(e_n, x)), & \text{(ii) if } (\forall w \leq x)(\forall y \leq t)[\mathbf{T} \nvdash_{\leq t} \\ & \ll \varphi^2_{e_2(n)}(w) = y \gg] \wedge \\ & \Phi_{e_0(n)}(x) > t; \\ \varphi_{e_0(n)}(x), & \text{(iii) otherwise.} \end{cases} \tag{1.21}$$

We note that all $\varphi_{e_2(n)}$'s are total. Therefore, from Lemma 1.2, we get

$$\text{For all } n, x, \varphi^2_{e_2(n)}(x) = \lim_{t\to\infty} \varphi_{e_2(n)}(x,t). \tag{1.22}$$

Let $f_n = \varphi_{e_0(n)}$. Let $\mathcal{C} = \{f_n \mid n \in N\}$. Clearly, $\mathcal{C} \subseteq \mathcal{R}$.

The rest of the proof involves ensuring, via a series of claims, that the many pieces of Theorem 1.4 hold.

Claim 1.9 *There exists a machine* $\mathbf{M}'$ *such that* $\mathcal{C} \subseteq \mathbf{Fin}(\mathbf{M}')$.

PROOF.

We recall that an inductive inference machine is allowed to output ?'s initially, before, if ever, it decides to output a program (number in N).

For all $f \in \mathcal{R}$, $n \in N$,

$$\mathbf{M}'(f[n]) = \begin{cases} ?, & \text{if } n = 0; \\ \pi_1^2(f(0)), & \text{otherwise.} \end{cases}$$

Suppose $f = f_n \in \mathcal{C}$. From the definition of $\varphi_{e_0(n)}$, $f(0) = \langle e_0(n), e_2(n) \rangle$. So, $\varphi_{\mathbf{M}'(f)} = f$. Also, $\text{card}(\{n \mid ? \neq \mathbf{M}'(f[n]) \neq \mathbf{M}'(f[n+1])\}) = 0$. Therefore, $\mathcal{C} \subseteq \mathbf{Fin}(\mathbf{M}')$. $\square$

Claim 1.10 *For all* $f \in \mathcal{C}$, *for all* p *such that* $\varphi_p =^\infty f$, $\mathbf{T} \nvdash \ll \varphi_p^1$ *is monotone* $\gg$.

PROOF. Suppose by way of contradiction that $(\exists f \in \mathcal{C})(\exists p \mid \varphi_p =^\infty f)[\mathbf{T} \vdash \ll \varphi_p^1 \text{ is monotone} \gg]$.

Let t be such that $\mathbf{T} \vdash_{=t} \ll \varphi_p^1$ is monotone $\gg$. Let n be such that $f = f_n$, which, by definition, is $\varphi_{e_0(n)}$.

We note that for all x, $f(x) \notin V(x)$. Also, φ_p is total. We observe, however, that for all $x \geq \max(\{p, t\})$, $\varphi_p(x) \in V(x)$. Thus, for all but finitely many x, $\varphi_p(x) \neq f(x)$, a contradiction. $\square$

Claim 1.11 *For all* n, x, y, $\mathbf{T} \nvdash \ll \varphi_{e_2(n)}^2(x) = y \gg$.

PROOF. Suppose by way of contradiction that n, x and y are such that $\mathbf{T} \vdash \ll \varphi_{e_2(n)}^2(x) = y \gg$. Furthermore, for this n, let w be the least number less than or equal to x for which there exists a y_0 such that $\mathbf{T} \vdash \ll \varphi_{e_2(n)}^2(w) = y_0 \gg$. By our assumption that $\mathbf{T}$ does not prove anything that is false, for n and w as above, y_0 is unique. Let t be such that $\mathbf{T} \vdash_{=t} \ll \varphi_{e_2(n)}^2(w) = y_0 \gg$.

Therefore, from equation (1.21), for all $t' \geq \max(\{t, y_0\})$, $\varphi_{e_2(n)}(w, t') = \min(\{S(e_n, x) \setminus \{y_0\}\})$. Hence $\varphi^2_{e_2(n)}(w) \neq y_0$, a contradiction. □

Claim 1.12 *For all n, $\varphi^2_{e_2(n)} = f_n$.*

PROOF. Consider any n. By Claim 1.11, Clause (i) of equation (1.21) cannot apply for any x, t. Therefore, clearly, using equations (1.18) and (1.22), since $\varphi_{e_0(n)}$ is total, $\varphi^2_{e_2(n)}(x) = \varphi_{e_0(n)}(x)$. Hence, the claim follows. □

Claim 1.13 *There is a machine* $\mathbf{M}$ *such that* $\mathcal{C} \subseteq \mathbf{LimFin}(\mathbf{M})$.

PROOF.

For all $f \in \mathcal{R}$, $n \in N$,

$$\mathbf{M}(f[n]) = \begin{cases} ?, & \text{if } n = 0; \\ \pi_2^2(f(0)), & \text{otherwise.} \end{cases}$$

Suppose $f = f_n \in \mathcal{C}$. From the definition of $\varphi_{e_0(n)}$, $f(0) = \langle e_0(n), e_2(n) \rangle$. By claim 1.12, $\varphi^2_{e_2(n)} = f$. So, $\varphi^2_{\mathbf{M}(f)} = f$. Also, $\mathrm{card}(\{n \mid ? \neq \mathbf{M}(f[n]) \neq \mathbf{M}(f[n+1])\}) = 0$. Therefore, $\mathcal{C} \subseteq \mathbf{LimFin}(\mathbf{M})$. □

$\mathbf{M}$ from the proof of Claim 1.13 just above is the one we use to prove Clause 3. We note that Claim 1.11, then, applies to the φ^2-programs that $\mathbf{M}$ converges to on functions in $\mathcal{C}$. Similar observations are true for Claims 1.14 and 1.16 below. We need the following Fact for proving Claims 1.14 and 1.16 below.

Fact 1.2

$$\mathbf{PA} \vdash \ll S \textit{ is computable} \gg \tag{1.23}$$

$$\mathbf{PA} \vdash \ll (\forall e, x)[\mathrm{card}(S(e, x)) = x + 3] \gg \tag{1.24}$$

$$\mathbf{PA} \vdash \ll (\forall e, x)[\max(S(e, x)) < \min(S(e, x + 1))] \gg \tag{1.25}$$

$$\mathbf{PA} \vdash \ll \text{domain}(V) \textit{ is } N \textit{ or is a finite initial segment of } N \gg \tag{1.26}$$

$$\mathbf{PA} \vdash \ll (\forall x)[x \in \text{domain}(V) \Rightarrow \text{card}(V(x)) \leq x+1] \gg \tag{1.27}$$

$$\mathbf{PA} \vdash \ll (\forall n)[\text{domain}(\varphi_{e_0(n)}) \textit{ is } N \textit{ or is a finite initial segment of } N] \gg \tag{1.28}$$

$$\mathbf{PA} \vdash \ll (\forall n, x)[x \in \text{domain}(\varphi_{e_0(n)}) \Rightarrow \varphi_{e_0(n)} \in S(e_n, x) \gg \tag{1.29}$$

PROOF.

Each of the assertions in Fact 1.2 can be proved using techniques similar to those used in the proof of Lemma 1.3. We omit the details. $\square$

Claim 1.14 *For all n, $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ is monotone $\gg$.*[19]

PROOF. We will first show that for all n, $\mathbf{PA} \vdash \ll (\forall x)[\varphi^2_{e_2(n)}(x) \in S(e_n, x)] \gg$. Then, the claim follows from equation (1.25).

Consider any n. Given any x, we consider the following (non-constructive) case analysis based on the definition of $\varphi_{e_2(n)}$ in equation (1.21). The case analysis below can be done within **PA**, but we omit the details of how this may be done within **PA**.

Case 1: $(\exists t)(\exists w \leq x)(\exists y \leq t)[\mathbf{T} \vdash_{\leq t} \ll \varphi^2_{e_2(n)}(w) = y \gg]$.

Let w be the least value for which there exists y such that $\mathbf{T} \vdash \ll \varphi^2_{e_2(n)}(w) = y \gg$. Also, let y_0 be the least value such that $\mathbf{T} \vdash \ll \varphi^2_{e_2(n)}(w) = y \gg$. Let t be the least value such that $\mathbf{T} \vdash_{\leq t} \ll \varphi_{e_2(n)}(w) = y_0 \gg$.

Therefore, from Clause (i) of equation (1.21), for all $t' \geq \max(\{w, t\})$, $\varphi_{e_2(n)}(x, t') = \min(S(e_n, x) \setminus \{y_0\})$, where y_0 is as found just above. Using equations (1.23) and (1.24), we can show that $\varphi^2_{e_2(n)}(x){\downarrow} \in S(e_n, x)$.

[19] *In fact, we can prove that* $\mathbf{PA} \vdash \ll (\forall n)[\varphi^2_{e_2(n)}$ *is monotone*$] \gg$.

Case 2: Not Case 1 and $(\forall t)[\Phi_{e_0(n)}(x) > t]$.

Clearly, in this case, for all t, $\varphi_{e_2(n)}(x,t) = \min(S(e_n,x))$. Hence, $\varphi^2_{e_2(n)}(x)\downarrow \in S(e_n,x)$.

Case 3: Not Case 1 and $(\exists t)[\Phi_{e_0(n)}(x) \leq t]$.

In this case, for all $t' \geq t$, $\varphi_{e_2(n)}(x,t') = \varphi_{e_0(n)}(x)$. Also, from equation (1.29), $\varphi_{e_0(n)}(x) \in S(e_n,x)$. Therefore, $\varphi^2_{e_2(n)}(x)\downarrow \in S(e_n,x)$.

By the above case analysis, $\varphi^2_{e_2(n)}(x) \in S(e_n,x)$. Therefore, by equation (1.25), $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ is monotone $\gg$. $\square$

As a corollary to Claim 1.14, we have

Claim 1.15 *For all n,* $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ *is total* $\gg$.

Claim 1.16 *For all n,* $\mathbf{PA} \vdash \ll \varphi^2_{e_2(n)}$ *is computable* $\gg$.[20]

Proof. Given $n \in N$, we consider the following (non-constructive) case analysis based on the definition of $\varphi_{e_2(n)}$. Using the definition of $\varphi_{e_2(n)}$, equations (1.26), and (1.28), the following (non-constructive) case analysis can be done within $\mathbf{PA}$. Once again, we omit the details.

Case 1: $(\exists x)(\exists t)(\exists w \leq x)(\exists y \leq t)[\mathbf{T} \vdash_{\leq t} \ll \varphi^2_{e_2(n)}(w) = y \gg]$.

Using the analysis similar to that in Case 1 of Claim 1.14, we can prove that there exists y_0, such that for all $x' \geq x$, $\varphi^2_{e_2(n)}(x') = \min(S(e_n,x') \setminus \{y_0\})$. Then, using equations (1.23) and (1.24), and Claim 1.15, we see that $\varphi^2_{e_2(n)}$ is computable.

Case 2: Not Case 1 and domain$(\varphi_{e_0(n)})$ is a finite initial segment of N.

In this case, for all but finitely many x, for all t, Clause (ii) of equation (1.21) holds for $\varphi_{e_2(n)}(x,t)$. Hence, for all but finitely many x, $\varphi^2_{e_2(n)}(x) = \min(S(e_n,x))$. Again, using equations (1.23) and (1.24), and Claim 1.15, we see that $\varphi^2_{e_2(n)}$ is computable.

[20] *Again, we can prove that* $\mathbf{PA} \vdash \ll (\forall n)[\varphi^2_{e_2(n)}$ *is computable*$] \gg$.

Case 3: Not Case 1 and domain($\varphi_{e_0(n)}$) is N.

We can show that for every x, for all but finitely many t, Clause (iii) of equation (1.21) holds. Hence, we can show that $\varphi^2_{e_2(n)} = \varphi_{e_0(n)}$. Also, from the definition of $\varphi_{e_0(n)}$, we can show that $\varphi_{e_0(n)}$ is computable. Therefore, $\varphi^2_{e_2(n)}$ is computable.

Thus, by the above case analysis, we conclude that **PA** $\vdash$ $\ll \varphi^2_{e_2(n)}$ is computable $\gg$. $\square$

From Claims 1.9–1.16, the theorem follows. $\square$ FOR THEOREM 1.4

1.5 Open Questions

1. Is Theorem 1.3 (or 1.4) true, for example, for Σ^0_2- vs. Σ^0_3-Definitions, etc., while keeping the class of functions $\mathcal{C}$ *computable*?

2. Is it possible to characterize or generalize the properties such as totality, monotonicity, etc. for which Theorem 1.3 (or 1.4) is true?

3. Can we add to Theorem 1.3 (or 1.4) something like:
   'For all machines $\mathbf{M}'$ witnessing that $\mathcal{C} \in$ **LimFin**, certain interesting subcases of Clause 3 hold, but with $\mathbf{M}$ replaced by $\mathbf{M}'$'?

4. Is Theorem 1.3 (or 1.4) true if we replace **PA** by Heyting Arithmetic (**HA** [Kle52]) in Clause 3a only? Of course in Clause 3c we can*not* replace **PA** by **HA**. For, otherwise, we get a contradiction to Clause 3b: informally, using the existence property [Kle45, Kle52] (see also [Fri75]) for **HA**, one can get, in essence, an ordinary program (in arithmetic as a numeral) provably equivalent in **HA** to φ^2_e; from this one can easily prove in **HA** (hence in **PA**) all the data points in φ^2_e, thus contradicting Clause (3b).

BIBLIOGRAPHY

[AS83] D. Angluin and C. Smith. A survey of inductive inference: Theory and methods. *Computing Surveys*, 15:237–289, 1983.

[Bar74] J. M. Barzdin. Two theorems on the limiting synthesis of functions. In *Theory of Algorithms and Programs*, volume 88, pages 82–88. Latvian State University, Riga, 1974.

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

[BCJS94] G. Baliga, J. Case, S. Jain, and M. Suraj. Machine learning of higher order programs. *Journal of Symbolic Logic*, 59(2):486–500, 1994.

[BF74] J. Bārzdiņš and R. Freivalds. Prediction and limiting synthesis of recursively enumerable classes of functions. *Latvijas Valsts Univ. Zimatm. Raksti*, 210:101–111, 1974.

[Blu67] M. Blum. A machine independent theory of the complexity of recursive functions. *Journal of the ACM*, 14:322–336, 1967.

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

[Cas94] J. Case. Infinitary self-reference in learning theory. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:3–16, 1994.

[CCJ01] J. Case, K. Chen, and S. Jain. Costs of general purpose learning. *Theoretical Computer Science*, 259:455–473, 2001.

[CJS92] J. Case, S. Jain, and A. Sharma. On learning limiting programs. *International Journal of Foundations of Computer Science*, 3(1):93–115, 1992.

[CS83] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.

[CS99] J. Case and M. Suraj. Inductive inference of Σ_1^0-vs. Σ_2^0-definitions for computable functions. In *Proceedings of the International Conference on Mathematical Logic*, 1999. Conference in Novosibirsk, Russia.

[dMSS56] K. deLeeuw, E. Moore, C. Shannon, and N. Shapiro. Computability by probabilistic machines. *Automata Studies, Annals of Math. Studies*, 34:183–212, 1956.

[Fef60] S. Feferman. Arithmetization of metamathematics in a general setting. *Fundamenta Mathematicae*, 49:35–92, 1960.

[Fri75] H. Friedman. The disjunction property implies the numerical existence property. *Proc. Nat. Acad. Sci. U.S.A.*, 72(8):2877–2878, 1975.

[Ful85] M. Fulk. *A Study of Inductive Inference Machines*. PhD thesis, SUNY at Buffalo, 1985.

[Gil72] J. Gill. *Probabilistic Turing Machines and Complexity of Computation*. PhD thesis, University of California, Berkeley, 1972.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6:675–695, 1977.

[Gly85] C. Glymour. Inductive inference in the limit. *Erkenntnis*, 22:23–31, 1985.

[Gol65] E. Gold. Limiting recursion. *Journal of Symbolic Logic*, 30:28–48, 1965.

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

[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.

[Kel96] K. Kelly. *The Logic of Reliable Inquiry*. Oxford University Press, Oxford, 1996.

[KG89] K. Kelly and C. Glymour. Convergence to the truth and nothing but the truth. *Philosophy of Science*, 56:185–220, 1989.

[KG90] K. Kelly and C. Glymour. Theory discovery from data with mixed quantifiers. *Journal of Philosophical Logic*, 19:1–33, 1990.

[Kle45] S. C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[Kle52] S. C. Kleene. *Introduction to metamathematics.* D. Van Nostrand Co., Inc., New York, N. Y., 1952.

[KS95] K. Kelly and O. Schulte. The computable testability of theories with uncomputable predictions. *Erkenntnis*, 43:29–66, 1995.

[KSJ97] K. Kelly, O. Schulte, and C. Juhl. Learning theory and philosophy of science. *Philosophy of Science*, 64:245–267, 1997.

[KW80] R. Klette and R. Wiehagen. Research in the theory of inductive inference by GDR mathematicians – A survey. *Information Sciences*, 22:149–169, 1980.

[LMF76] N. Lynch, A. Meyer, and M. Fischer. Relativization of the theory of computational complexity. *Transactions of the American Mathematical Society*, 220:243–287, 1976.

[Men86] E. Mendelson. *Introduction to Mathematical Logic.* Brooks-Cole, San Francisco, third edition, 1986.

[Men97] E. Mendelson. *Introduction to Mathematical Logic.* Chapman & Hall, London, fourth edition, 1997.

[MY78] M. Machtey and P. Young. *An Introduction to the General Theory of Algorithms.* North–Holland, New York, 1978.

[Odi99] P. Odifreddi. *Classical Recursion Theory*, volume II. Elsivier, Amsterdam, 1999.

[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.

[Put63] H. Putnam. Probability and confirmation. *Voice of America, Forum on Philosophy of Science*, 10, 1963. Also appears as [Put75].

[Put65] H. Putnam. Trial and error predicates and the solution to a problem of Mostowski. *Journal of Symbolic Logic*, 30:49–57, 1965.

[Put75] H. Putnam. Probability and confirmation. In *Mathematics, Matter, and Method.* Cambridge University Press, 1975.

[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, MIT Press, 1987.

[Roy87] J. Royer. *A Connotational Theory of Program Structure*. Lecture Notes in Computer Science 273. Springer Verlag, 1987.

[Sch99] O. Schulte. Means-ends epistemology. *British Journal for the Philosophy of Science*, 50:1–31, 1999.

[Sha71] N. Shapiro. Review of "Limiting recursion" by E.M. Gold and "Trial and error predicates and the solution to a problem of Mostowski" by H. Putnam. *Journal of Symbolic Logic*, 36:342, 1971.

[Sur01] Mandayam Suraj. *Theory of Computationally Learned Definitions*. PhD thesis, Department of Computer & Information Sciences, University of Delaware, 2001.

[TB70] B. Trakhtenbrot and J. Bārzdiņš. *Konetschnyje Awtomaty (Powedenie i Sintez)*. Nauka, Moskwa, 1970. in Russian. English Translation: Finite Automata–Behavior and Synthesis, *Fundamental Studies in Computer Science 1*, North Holland, Amsterdam, 1975.