TOPICS IN COMPUTABILITY, COMPLEXITY, CONSTRUCTIVITY & PROVABILITY

by

Michael Ralston

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

Spring 2014

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

TOPICS IN COMPUTABILITY, COMPLEXITY, CONSTRUCTIVITY & PROVABILITY

by

Michael Ralston

Approved: _____
Errol L. Lloyd, Ph.D.
Chair of the Department of Computer Science

Approved: _____
Babatunde Ogunnaike
Dean of the College of Engineering

Approved: _____
James G. Richards, Ph.D.
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: _____

James Royer, 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: _____

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

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. John Case, for all his assistance with this thesis; he helped me find interesting problems when my first line of research proved unfruitful, he has helped me immensely with my rhetoric, and last but not least he has provided me with the encouragement and motivation required to complete this work while also working at my place of employment.

I also want to thank the remaining members of my committee, Dr. David Saunders, Dr. Daniel Chester, and Dr. James S. Royer, for their time and feedback.

I futher want to thank Dr. Timo Kötzing, my advisor's just previous Ph.D. student, who has served as a role model and has provided assistance as well as the perspective of a fellow student.

I also want to thank the other researchers with whom I have had the honor of corresponding, especially Dr. Yohji Akama.

In addition, I want to thank my place of employment, IMVU, for giving me the time I've needed to complete this research.

And last, but certainly not least, I want to thank my mother for her unflagging support of my educational endeavors.

TABLE OF CONTENTS

ABSTRACT

The three content chapters of this doctoral dissertation involve *each* of the concepts Computability, Complexity, Constructivity & Provability from the title.

One of these chapters is devoted to showing that *un*verifiable programs need not be obfuscated. An application casts some doubts on an interesting 1980 argument of Putnam's. It is also shown that there is an acceptable programming system, of course with infinitely many universal simulating programs, presented so that it has exactly one verifiable such universal program, and there is another acceptable system presented so that it has *no* verifiable universal programs.

Another chapter was suggested by two functions in Rogers' book which are based on eventual, currently unknown patterns in the decimal expansion of π. One of them is classically and not at all constructively provably computable, and there is no known algorithm for it. For the other it is unknown whether it is computable. A problem is that advances in relevant knowledge about these now unknown patterns in π may destroy Rogers' examples. Presented in this chapter is a safer computable real to replace π so that the associated first function retains its classically provable computability, but has *un*provability of the correctness of any particular program for it. For the second Rogers' function π is replaced by a real with each bit linear-time computable in the *length* of its position, but with the associated second function provably *un*computable.

The last chapter features some computability results which are shown to provably require *non*-constructivity, e.g., that the program equivalence problem for acceptable programming systems is not computably enumerable (c.e.). Characterized is how to divide this example problem into non-trivial cases of disjoint index sets, where showing each of these index sets to be non-c.e. has a kind of unformity not found for the full

equivalence problem, and each set's being non-c.e. is of ostensibly lower degree of non-constructivity. Lastly, some related results are presented for natural run-time bounded programming systems — with run time bounds all the way down to linear-time. This chapter suggests a Reverse Mathematics project to minimize non-constructivity here and elsewhere.

Chapter 1

INTRODUCTION

This doctoral dissertation consists of four chapters together with a single bibliography. Chapter 1 is this introductory chapter. Chapters 2–4 are the content chapters. The three content chapters *each* involve the concepts of Computability, Complexity, Constructivity & Provability from the title.

We explain in the next section (Section 1.1) each of these concepts as it applies herein.

Then, in the next and last introductory section (Section 1.2) we discuss the three content chapters and briefly summarize the contents of each of them.

1.1 Background Concepts

In what follows, any residually unexplained notation or concept is from at least one of [44, 47, 35].

The Computability component is, of course, about that for which there are algorithms [44] (and that for which there are not).

In each content chapter the Complexity component involves *run-time complexity* at or above the level of linear-time computability (in the lengths of the inputs in binary) — all on efficiently described/coded, deterministic, multi-tape Turing machines (TMs) [47], essentially a *base* model for deterministic run time costs.

The Constructivity component involves various levels, and, for a generic computer science audience, we need to explain the concept and its herein relevant levels.

First we explain full-fledged constructivity with a general statement re its intent. The idea, beginning with the intuitionist[1] (constructivist) Brouwer, from about 1908 (see [7]), is that mathematical proofs at every layer and step should permit the explicit presentation/*computation* of *examples proved to exist*. This places some limitations (compared to classical mathematics) not only on the logical operator $\exists$, but also on the logical operator $\vee$. The problem with $\vee$ is nicely illustrated in [2] where a proof by cases (this case $\vee$ its opposite) permits *non*-constructivity. The classically valid principle of the Law of Excluded Middle (LEM) says that any proposition $\vee$ its opposite holds. In constructively permissible proofs by cases, to employ LEM, it must be *decidable* as to which case/disjunct holds.

By the way, constructive proofs *and even partially constructive proofs* are of interest for computer science — since they permit information to be computationally extracted from *such* proofs. In the case of the wonderful old AI project, Shakey the Robot, Shakey extracted, from proofs about its goals, its corresponding plans of actions to achieve those goals (see, [22, 20, 48]). Fortunately Shakey's proofs *just happened to be* constructive (perhaps because its world was so very finite) so its proof-mining was possible.[2] More modern computational scientists who are on about information extraction from proofs are very explicitly aware that that extraction requires at least some constructivity (or some way of easily introducing it). For example, [29] nicely employs such proof-mining to get numerical analysis convergence bounds.

It is useful to discuss an important *example* theorem from theoretical computer science whose original proof is *partially* constructive, but for which it became known that other parts of its proof that one would also like to be constructive provably can never be made so. The example is the famous, surprising Blum Speed-Up Theorem [3,

[1] Technically, Intuitionism is a brand of Constructivism somewhat more subjective in focus than general constructivism. We will not and need not explore herein the subjectivism of Brouwer's Intuitionism.

[2] The project suffered instead from the slow speed of finding the proofs [48].

15]. Informally it says that, for any (possibly large) computable factor h (e.g., super-exponential), there is a $\{0,1\}$-valued computable function f so that, for *any* program i for f, there is another program j for f which, on *all* inputs *excluding a finite exception set depending on* i, j is h*-faster* than i. The finite exception set *must* be there since, of course, run-times are *not* infinitely descending and cannot go below zero.

The constructive part of the original proof of the Speed-Up Theorem: an explicit program is displayed for defining f, and such a defining program *is* computable from any program for a pre-given h. That is nice since we are *not* stuck with a proof of f's existence without knowing an example of f. In [4] it is shown that, in general[3], *no* h-sped-up program j for f is *computable* from any pre-given i for f for which j is an h-speed-up of i. This unfortunate bit of *non*-constructivity cannot, then, be repaired by a better proof. Later in [49, 50] it is shown that, in general, there is *no* computable from i simultaneous *upper-bound* for both the corresponding sped-up js and the corresponding exceptional values — more irreparable *non*-constructivity. Much later, in [6], it is shown that, in general, there is *no* computable function of an initial program i (for f) that gives an upper-bound on just the associated exceptional values for a speed-up j (another irreparable *non*-constructivity); *but* that, if the upper-bounding function is taken as a function of the sped-up program j, then it *can* be chosen to be computable (a new, tiny bit of *constructivity* found). In a *fully* constructive proof of the Blum Speed-Up Theorem (which, by the above remarks, can*not* exist), the j and all the exceptional values would be exactly computable.

The sets (of non-negative integers) in the (Kleene-Mostowski) Arithmetical Hierarchy [44] are those which can be defined by a computably-decidable predicate with zero or more numerical quantifiers (over the elements of $\mathbb{N} \stackrel{\text{def}}{=} \{0,1,2,\ldots\}$) in front. These defining formulas can be algorithmically put into a kind of normal form [35, 44] in which the quantifiers, if any, alternate between $\exists$ and $\forall$. The more alternations, the more sets that are so definable [44] — whence the term, Hierarchy. The sets so

[3] In the context of Blum Speed-Up, 'in general' means: for all sufficiently large computable h.

definable are called *arithmetical*. The arithmetical sets are nicely *characterized* [44] as the sets semantically definable in so-called first-order arithmetic [35, 44], a theory with names for the non-negative integers, symbols for equality and the operations of successor, addition, and multiplication of them, and with quantifiers over them. The Peano Axioms [35, 44] can be formulated in this theory underpinned by first order logic (with standard equality axioms) [35]. All the theorems in an *elementary* number theory text book can be formulated and proved thereby.[4] We'll refer to this axiomatized theory as (first order) *Peano Arithmetic* (**PA** for short). Within this dissertation, the mathematics for which we'll consider *un*provability results is all *expressible* in the *language* of **PA**. There is an extension of this language which allows quantifiers *also* over the *sets* of non-negative numbers and which is needed for expressing other parts of mathematics (see, e.g., [51, 44]), but we have no need of the extra *expressibility* of such an extension. When we discuss the concept of Provability below, though, we'll have a need for such stronger theories, but only for their ability to *prove more theorems* which theorems are *still expressible in first order arithmetic*.

Brouwer's student, Heyting, formulated a variant of first order logic called *intuitionistic logic* which is just like first order logic except it is is missing unrestricted LEM; it captures thereby Brouwer's constructivist ideas in the context of logic [26]. *Heyting Arithmetic* (abbreviated: **HA**) is just **PA** but underpinned instead by intuitionistic logic (see also [52, 53]). It captures Brouwer's ideas about the mathematics expressible in it. We won't need to consider herein constructivism for mathematics not expressible in first order arithmetic.

There has been somewhat recent interest in adding some *arithmetically limited version* of LEM (and other principles) to **HA**. One, then, gets theories of strictly intermediate strength between **HA** and **PA** [38, 1, 25]. We give an example next, of some interest for this dissertation.

[4] But, by the first Gödel Incompleteness Theorem [21, 35], neither the Peano Axioms (nor any other computably decidable set of true axioms) suffice to prove *all* the first order expressible truths of the arithmetic of non-negative integers.

An important example for us: the Π^0_2 sets in the arithmetical hierarchy are just those that, for some computable predicate R, are of the form

$$\{w \mid (\forall u)(\exists v)R(u,v,w)\}. \tag{1.1}$$

Π^0_2-LEM, an example intermediate theory, is, then, the class of instances of LEM of the form

$$(\forall u)(\exists v)R(u,v,w) \ \vee \ \neg(\forall u)(\exists v)R(u,v,w), \tag{1.2}$$

where R is some computable predicate.[5]

More generally, Π^0_n is the class of arithmetical sets defined by a formula of the form, an alternation between n instances of $\forall$ & $\exists$ quantifying distinct variables *beginning* with $\forall$ and followed by a computable predicate. The $n = 2$ case is spelled out more concretely just above. Σ^0_n is the class of arithmetical sets defined by a formula of the form, an alternation between n instances of $\forall$ & $\exists$ quantifying distinct variables *beginning* with $\exists$ and followed by a computable predicate. $\Delta^0_n = \Pi^0_n \cap \Sigma^0_n$.

For each of these example classes of arithmetical sets $\mathcal{A}$, $\mathcal{A}$-LEM is the class of instances of LEM of the form

$$A \ \vee \ \neg A, \tag{1.3}$$

where A is some member of $\mathcal{A}$.

Let $\mathbf{HA}+\mathcal{A}$-LEM denote the result of adding the $\mathcal{A}$-LEM formulas to $\mathbf{HA}$.

Essentially from [1], we have that, for example, $\mathbf{HA}$, $\mathbf{HA}+\Pi^0_1$-LEM, $\mathbf{HA}+\Sigma^0_1$-LEM, $\mathbf{HA}+\Delta^0_2$-LEM, $\mathbf{HA}+\Pi^0_2$-LEM, $\mathbf{HA}+\Sigma^0_2$-LEM, $\mathbf{HA}+\Pi^0_3$-LEM, ... is a sequence of ever stronger (i.e., more non-constructive) theories.

One can put any formula of first order arithmetic into a *classically* equivalent normal form variant with all its quantifiers in front and alternating, *but* this equivalence, in some cases, is *not* provable in $\mathbf{HA}$. However, from [1], for $\mathcal{A}$, for example, one of the arithmetical classes defined above, ($\mathbf{HA}+\mathcal{A}$-LEM) *is* strong enough to *prove* the

[5] It turns out, *in the cases there are quantifiers in front of* R, we can take the Rs to be primitive recursive [44]. *Then*, for example, Π^0_2-LEM is a computably decidable set of sentences. In this way proof-checking in, for example, ($\mathbf{HA} + \Pi^0_2$-LEM) is algorithmic.

equivalence of the normal form variant of formulas classically equivalent to formulas defining the members of $\mathcal{A}$.

In *some* cases of these strictly intermediate theories [38], while objects proved to exist may not be computable, they are *limiting-computable*, computable by a total "mind-changing" algorithm which eventually stops changing its mind and then it lands on a correct answer (but there may be no algorithm to find when the mind-changes stop). In general, these strictly intermediate theories measure *degrees* of non-constructivity.

The Constructivity component of this dissertation ranges from being about the mere ability to *compute some* proved-to-exist examples; to full-fledged provably-correct in **HA**; to the latter with the addition of arithmetically-*limited*, LEM principles.

Next we briefly discuss the Provability concept as it occurs in the current dissertation. The Provability component is mostly about the existence (or non-existence) of formal proofs in algorithmically-decidably (computably) axiomatized, classical first order theories such as **PA** through (large cardinal axiom extensions of) Zermelo-Frankel Set Theory with the Axiom of Choice (**ZFC**)[6] [45, 27, 17, 28] *but* with (relevant) consequences *expressible and true in first order arithmetic*. As we've seen above, some of it is about provability in **HA** sometimes supplemented as mentioned just above with some limited non-constructive principles.

1.2 Summary of Content Chapters

Much of the material described in Chapters 2 and 4 has already been presented at the respective conferences [11, 13] and is joint work with my dissertation advisor, John Case, with [13] also joint work with Yohji Akama from the Mathematical Institute, Tohoku University, Sendai, Japan. Much of that described in Chapter 3 appears in a conference proceedings [12] and is joint work with my dissertation advisor, John Case.

[6] **ZFC** essentially enables proof of all of standard, non-mathematical logic, text book mathematics.

In addition, each of the three content chapters has now been submitted as a full paper to a referred outlet — different outlets for different content chapters.

Notation useful just below is the linear-time computable and invertible pairing function $\langle \cdot, \cdot \rangle$ from [47]. This function maps all the pairs of elements of $\mathbb{N}$ 1-1, onto $\mathbb{N}$. We also employ this notation, based on iterating, $\langle \cdot, \cdot \rangle$, as in [47], to code also triples, quadruples, ... of elements of $\mathbb{N}$ 1-1, onto $\mathbb{N}$. Also useful just below is the concept of *acceptable programming systems*. These are characterized as those intercompilable with standard, natural programming systems such as full Turing machine systems. In the content chapters further below we freely use Church's lambda notation somewhat as in many dialects of Lisp, but we use the Greek letter instead of its English spelling. For example, $\lambda u, v.(u^2 + v)$ denotes the function which on input (u, v) returns the corresponding value of $(u^2 + v)$.

In several parts of this dissertation there are results to the effect that some inequalities hold for *small* constants. The proofs of these inequalities are seen to employ some cited inequalities from [47]. A careful analysis of the proofs of these latter inequalities would enable a tight upper bound calculation of the numerical value of these constants. It is clear from a casual inspection of these cited proofs in [47] that the associated constants would be close to low double digit numbers. The results referring to small constants are stronger for their interpretations that the size of them is so limited.

Next is our summary of the three content chapters.

Summary of Chapter 2: The *International Obfuscated C Code Contest* was a programming contest for the most creatively obfuscated C code. In many cases, the winning programmer did something simple in such an *obscure but succinct* way that it was hard for other (human) programmers to see how his/her code actually worked. By *contrast*, the interest of this chapter is in programs which are *relatively* succinct and fast; which, *in a sense*, are *easily* seen to be correct; but which can*not* be proved correct in any pre-assigned, computably axiomatized, powerful, true theory **T**. Specifically, it is shown that, for any deterministic, multi-tape TM program p, there will be an *easily*

seen equivalent such program q *almost (i.e., within small linear factors) as fast and succinct as* p, but this equivalence will *not* be provable in $\mathbf{T}$. My advisor's original motivation: in a 1980 book chapter, the philosopher Hilary Putnam essentially says that all the (semantically normal[7]) short-fast programs p for deciding membership in the set of well-formed formulas of the propositional calculus are similar and intrinsic, but my advisor believed one could create also (at least extensionally, semantically normal) short-fast programs q easily seen equivalent to such p but quite *dis*similar in that their equivalence wasn't provable (by pre-assigned, powerful, suitable, true $\mathbf{T}$). This is, then, realized as a special case of (the proof of) the just above result. Further and related questions and subtleties are considered and resolved. For example, it is shown there is an acceptable programming system, which, then, must have infinitely many universal simulating programs, but presented so that exactly one such universal program is *provably* correct, and there is another acceptable system presented so that there are *no* provably correct universal programs.

Summary of Chapter 3: On page 9 of Rogers' computability theory book [44] he presents two functions each based on eventual, currently unknown patterns in the decimal expansion of π. One of them is easily (classically) seen to be computable, but the proof is highly non-constructive and, conceptually interestingly, there is no *known* example algorithm for it. This function on any non-negative integer x is 1 if there is a *consecutive* sequence of 5s the decimal expansion of π of length *at least* x; else, it is 0. For the other, it is unknown as to whether it is computable. This function is like the first but with *at least* replaced by *exactly*. In the future, though, these unknown patterns in the decimal expansion of π may be sufficiently resolved, so that, for the one, we will know a particular algorithm for it, and/or, for the other whether it's computable. This chapter provides a "safer" computable real to replace π so that the associated one function retains its trivial computability but has unprovability of the correctness of any particular program for it. Re the other function, a real $\mathbf{r}$ to replace π is given

[7] Semantically normal means here that p's definition of grammaticality parallels the standard semantic definition of the corresponding truth-tables.

with each bit of this **r** uniformly linear-time computable in the *length* of its position and so that the Rogers' other function associated with **r** is provably uncomputable.

Summary of Chapter 4: Let φ be an acceptable programming system/numbering of the partial computable functions: $N = \{0, 1, 2, \ldots\} \rightarrow N$, where, for $p \in N$, φ_p is the partial computable function computed by program (number) p of the φ-system and $W_p = \text{domain}(\varphi_p)$; W_p is, then, the computably enumerable (c.e.) set accepted by p.

My advisor taught in a class a recursion theorem proof *employing a pair of cases* that, for *each* q, $\{x \mid \varphi_x = \varphi_q\}$ is not c.e. The particular (non-constructive) disjunction of cases employed was: $\text{domain}(\varphi_q)$ is infinite vs. finite. Some student asked why the proof involved an analysis by cases. The answer given straightaway to the student was that his teacher didn't know how else to do it. This chapter provides, among other things, a better answer: any proof that, for each q, $\{x \mid \varphi_x = \varphi_q\}$ is not c.e. *provably must* involve some such *non-constructiveness*. Furthermore, *completely characterized* are all the possible such pairs of cases that will work! The well-known completely-productive sets (abbr: c-productive) are the *effectively* non-c.e. sets. A set *sequence* $S_q, q = 0, 1, 2, \ldots$ is said to be *uniformly c-productive* iff there is a computable f so that, for all q, x, $f(q, x)$ is a counterexample to $S_q = W_x$. Certainly a completely constructive proof that each S_q is not c.e. would *entail* the S_qs forming a uniformly c-productive sequence. Relevantly shown is that the set *sequence* $\{x \mid \varphi_x = \varphi_q\}, q = 0, 1, 2, \ldots$ is *not* uniformly c-productive — and this even though, of course, the *set* $\{\langle q, x\rangle \mid \varphi_x = \varphi_q\}$ *is* c-productive. For some results we upper-bound them as to position in the Arithmetical Hierarchy of LEM needed. For another we also have a loose lower-bound. This begins a Reverse Mathematics (`en.wikipedia.org/wiki/Reverse_mathematics`) program to find the *minimum* non-constructivity needed for theorems. Results are also obtained for some similar problems, including for *run-time bounded programming systems*. Proof methods include recursion theorems.

Chapter 2

NON-OBFUSCATED YET UNPROVABLE PROGRAMS

2.1 Introduction

2.1.1 Background

The *International Obfuscated C Code Contest* (see the Wikepedia entry) was a programming contest for the most creatively obfuscated C code, held annually between 1984 and 1996, and thereafter in 1998, 2000, 2001, 2004, and 2006.

In many cases, the winning programmer did something simple in such an obscure but succinct way that it was hard for other (human) programmers to see how his/her code actually worked.

By *contrast*, the interest of this chapter is in programs which are, *in a sense*, *easily* seen to be correct, but which can*not* be proved correct in pre-assigned, computably axiomatized, powerful, true theories **T**. A point is that, then, *un*verifiable programs need *not* be obfuscated!

Our main theorem of this chapter (Theorem 1 in Section 2.2.1 below) entails: for *any* deterministic, multi-tape Turing Machine (TM) program p, there will be an *easily seen equivalent* such TM program q *almost (i.e., within small, linear factors) as fast and succinct as* p, but this equivalence will *not* be provable in **T**.

A point of the just mentioned, small, linear factors is that the *un*provability is *not* based on some huge (or at least non-linear) growth in run-time and/or program size in passing from p to q. In fact we'll see in the proof of the main theorem of this chapter that q will be like p except that q in effect encapsulates p in a top-level if-then-else with: 1. p being the else-part and 2. the succinct, linear-time testable if-condition being easily seen never to come true.

A motivated, concrete, special case will be presented (also in Section 2.2.1 below).

As will be seen, details regarding the main theorem of the present chapter suggest pleasantly *subtle* problems, and some are solved herein (in Section 2.2.2 below).

2.1.2 Mathematical Preliminaries

2.1.2.1 Complexity-Bounded Computability

Let φ^{TM} be the *efficiently* laid out and Gödel-numbered acceptable programming system (numbering) from [47, Chapter 3 & Errata] and which is based on deterministic *multi-tape Turing Machines* (with base two I/O). Its programs are named by all numbers in $\mathbb{N} = \{0, 1, 2, \ldots\}$. φ_p^{TM} is the partial computable function $\mathbb{N} \to \mathbb{N}$ computed by φ^{TM}-program (number) p. The numerical naming just mentioned does *not* feature prime powers and factorization, but, instead, is a *linear-time computable and invertible coding*. Let Φ^{TM} be the corresponding step-counting Blum Complexity Measure [3]. $(\varphi^{\mathrm{TM}}, \Phi^{\mathrm{TM}})$ is a *base* model for deterministic run time costs.

Within this chapter, we will use the linear-time computable and invertible pairing function $\langle \cdot, \cdot \rangle$ from [47]. This function maps all the *pairs* of elements of $\mathbb{N}$ 1-1, onto $\mathbb{N}$. We also employ this notation, based on iterating, $\langle \cdot, \cdot \rangle$, as in [47], to code also triples, quadruples, ... of elements of $\mathbb{N}$ 1-1, onto $\mathbb{N}$.

$\mathcal{L}$time is the class of functions: $\mathbb{N} \to \mathbb{N}$ each computable by some φ^{TM}-program running within a Φ^{TM}-time bound linear in the *length* of its base-two expressed argument. Of course by means of the iterated $\langle \cdot, \cdot \rangle$ function mentioned just above, we can and sometimes will speak of multi-argument functions as being (or not being) in $\mathcal{L}$time.

A $k \in \mathbb{N}$, k could be a numerically named program in φ^{TM} or just a datum. We let $|k|$ = the *length* of k, where k is written *in binary*. We can write this length as $(\lceil \log_2(k+1) \rceil)_+$, *where* $(\cdot)_+$ turns 0 into 1; else, leaves unchanged.[1]

[1] This formula can be derived as the minimal number of whole bits needed to store any one of the $k+1$ things 0 through k, *except* that the case of $k = 0$ needs only 0

Rogers [44] uses the terms 'converges' for computations which halt and provide output and 'diverges' for those that do not. Within this chapter we use the respective notations (due to Albert Meyer) $\downarrow$ and $\uparrow$ in place of these terms of Rogers.

From [47, Lemma 3.14], there are small positive $a \in \mathbb{N}$ and function if-then-else $\in \mathcal{L}$time st, for all $p_0, p_1, p_2, x \in \mathbb{N}$, $\varphi^{\mathrm{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) =$

$$\begin{cases} \varphi^{\mathrm{TM}}_{p_1}(x), & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow \neq 0; \\ \varphi^{\mathrm{TM}}_{p_2}(x), & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise;} \end{cases} \tag{2.1}$$

and $\Phi^{\mathrm{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) \leq$

$$\begin{cases} a \cdot (\Phi^{\mathrm{TM}}_{p_0}(x) + \Phi^{\mathrm{TM}}_{p_1}(x))_+, & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow \neq 0; \\ a \cdot (\Phi^{\mathrm{TM}}_{p_0}(x) + \Phi^{\mathrm{TM}}_{p_2}(x))_+, & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{2.2}$$

Essentially from (the $k, m = 1$ case of) [47, Theorem 4.8], we have the following constructive, *efficient*, and *parametrized* version of *Kleene's* 2nd (not Rogers') Recursion Theorem [44, Page 214].

There are small positive $b \in \mathbb{N}$ and function $\mathbf{krt} \in \mathcal{L}$time st, for all *parameter values* p, *tasks* r, *inputs* $x \in \mathbb{N}$:

$$\varphi^{\mathrm{TM}}_{\mathbf{krt}(\mathbf{p},\mathbf{r})}(x) = \varphi^{\mathrm{TM}}_r(\langle \mathbf{krt}(\mathbf{p},\mathbf{r}), \mathbf{p}, \mathbf{x}\rangle); \tag{2.3}$$

and $\Phi^{\mathrm{TM}}_{\mathbf{krt}(\mathbf{p},\mathbf{r})}(x) \leq$

$$b \cdot (|p| + |r| + |x| + \Phi^{\mathrm{TM}}_r(\langle \mathbf{krt}(\mathbf{p},\mathbf{r}), \mathbf{p}, \mathbf{x}\rangle)). \tag{2.4}$$

---

bits; however, a single 0 has length 1.

This and more general use of $(\cdot)_+$ *also* helps to deal with the fact that zero values can cause trouble for $\mathcal{O}$-notation. A problem comes with complexity bounds f of more than one argument. Jim Royer gave the example of two functions mapping pairs from $\mathbb{N}$, $f(m,n) = (m \cdot n)$ & $g(m,n) = (m+1) \cdot (n+1)$. Suppose, as *might* be expected, g is $\mathcal{O}(f)$. Then there are positive a, b such that, for each $m, n \in \mathbb{N}$, $g(m,n) \leq a \cdot f(m,n) + b$. Then we have, for *each* n, $n + 1 \leq a \cdot f(0,n) + b = b$, a contradiction. However, g *is* $\mathcal{O}((f)_+)$.

Above, in (2.3), on the left-hand side, the φ^{TM}-program $\mathbf{krt}(\mathbf{p},\mathbf{r})$ *has* p, r *stored inside*, and, on x, it: *makes a self-copy* (in linear-time), *forms* $y = \langle$*self-copy*$, p, x\rangle$ (in linear-time), and *runs task* r *on this* y. From (2.4) just above, for each p, r, any super-linear cost of running φ^{TM}-program $\mathbf{krt}(\mathbf{p},\mathbf{r})$ on its input is from the running of φ^{TM}-task r on its linear-time producible input.

2.1.2.2 Computably Axiomatized, Powerful, True Theories

Let $\mathbf{T}$ be a *computably axiomatized* first order (fo) theory extending fo Peano arithmetic (**PA**) [35, 44] — *but with numerals represented in base two to avoid size blow up from unary representation* (see [8, Page 29])[2] — *and which does not prove (standard model for* **PA** *[35]) falsehoods expressible in f.o. arithmetic.*

$\mathbf{T}$ could be, *for example*: fo Peano arithmetic (**PA**) itself, the two-sorted fo Peano arithmetic permitting quantifiers over numbers and sets of numbers [44, 51] (a second order arithmetic), Zermelo Frankel Set Theory with Choice (**ZFC**) [23], **ZFC** plus ones favorite large cardinal axiom [45, 27, 17, 28], etc.

---

[2] Let's suppose $\overline{0}$ is **PA**'s numeral for zero and that S is **PA**'s symbol for the successor function on $\mathbb{N}$. In effect, in, e.g., [35], the numeral $\overline{n}$ for natural number n is $S^{(n)}(\overline{0})$, where

$$S^{(n)} = \underbrace{S \circ \ldots \circ S}_{n\ S\text{s}}, \tag{2.5}$$

for iterated composition of Ss. This is a base *one* representation. Note that the length of *this* $\overline{n}$ is $\mathcal{O}(n)$ which is $\mathcal{O}$ of $2^{\text{the symbol length of } \overline{n}}$ — too high for feasible complexity. However, the symbol length for the binary representation of n grows only linearly with n — feasibly.
Based on [8, Page 29], within this chapter, by contrast with the just above, we can define our numeral $\overline{n}$ for $n \in \mathbb{N}$ thus. We suppose $\cdot$ is the symbol for **PA**'s multiplication over $\mathbb{N}$. We let: $\overline{2} = S \circ S(\overline{0})$; for $(n > 1)$, $n \in \mathbb{N}$,

$$\overline{2n} = (\overline{2} \cdot \overline{n}); \tag{2.6}$$

and, for $n \in \mathbb{N}$,

$$\overline{(2n)+1} = S(\overline{2n}). \tag{2.7}$$

Then, the length of $\overline{n}$ is in $\mathcal{O}$ of the symbol length of $\overline{n}$ — feasible.

If E is an expression such as 'the partial function computed by φ^{TM}-program # p is total' and which is expressible in **PA** and where p is a particular element of $\mathbb{N}$, we shall write $\ll \mathsf{E} \gg$ to denote a typically *naturally* corresponding, fixed standard cwff (closed well-formed formula) of **T** which (semantically) expresses E — and where p is expressed as the corresponding numeral in base two (as indicated above). We have that

> if E' is obtained from E by substituting a numerical value k, then $\ll \mathsf{E}' \gg$ can be algorithmically obtained from $\ll \mathsf{E} \gg$ in linear-time in $(|\ll \mathsf{E} \gg| + |k|)$.

By [47, Theorem 3.6 & Corollary 3.7] and their proofs, the running of a *carefully crafted, time-bounded, φ^{TM}-universal simulation* up through time t takes time a little worse than exponential in $|t|$. Early complexity theory, e.g., [5, 32, 31], provided delaying tricks to achieve polynomial time. From [47, Theorem 3.20] and its proof, the above mentioned carefully crafted, time-bounded universal simulation of any φ^{TM}-program can be *uniformly delayed* by a $\log \log$ factor on the time-bound to run *in* $\mathcal{L}$time.

The theorems of **T** form a *computably enumerable* set, so we can/do fix an automatic theorem prover for **T**. Let

$$\mathbf{T} \vdash_x \ll \mathsf{E} \gg \tag{2.8}$$

mean that *a delayed as above, linear-time computable, time-bounded simulation* of the fixed automatic theorem prover proves $\ll \mathsf{E} \gg$ from **T** *within* x steps — that's *linear-time in* $(|\ll \mathsf{E} \gg| + |x|)$.

Let D_x be the finite set ($\subseteq \mathbb{N}$) with canonical index x (see, e.g., [44]). x codes, for example, *both* how to list D_x *and* how to know when the listing is done. For the purposes of this chapter, we can and do restrict our canonical indexing of finite sets to those of sets cardinality ≤ 2. We do that in linear-time thus. Let 0 be the code of

$\emptyset$, and, for set $\{u, v\}$ (where u might or might not equal v), let the code be $\langle u, v\rangle + 1$. This coding is linear-time codable/decodable.[3]

2.2 Results

2.2.1 Main Result

Theorem 1. *There* exists $g \in \mathcal{L}\text{time}$ *and* small *positive* $c, d \in \mathbb{N}$ *such that, for* any p, $|D_{g(p)}| = 2$ *and* there is *a* $q \in D_{g(p)}$ *for which:*

$$\varphi_q^{\text{TM}} = \varphi_p^{\text{TM}}; \tag{2.9}$$

for all $x \in \mathbb{N}$,

$$\Phi_q^{\text{TM}}(x) \leq c \cdot (|p| + |x| + \Phi_p^{\text{TM}}(x)); \tag{2.10}$$

$$|q| \leq d \cdot |p|; \tag{2.11}$$

yet

$$\mathbf{T} \nvdash \ll \varphi_q^{\text{TM}} = \varphi_p^{\text{TM}} \gg. \tag{2.12}$$

Our proof below of Theorem 1, as will be seen, makes it *easily transparent* that $\varphi_q^{\text{TM}} = \varphi_p^{\text{TM}}$. Hence, q is *not obfuscated*, yet its correctness (at computing φ_p^{TM}), as will also be seen, is unprovable in $\mathbf{T}$. From the *time and program size complexity* content of the theorem, q is nicely *only slightly, linearly* more complex than p. Furthermore, our proof is what is called in [47, Page 131] *a rubber wall argument*: we set up a rubber wall, i.e., a *potential* contradiction off of which to bounce, so that, were the resultant construction to veer into satisfaction of an undesired condition (undesired here is the failure of (2.12) above), it bounces off the rubber wall (i.e., contradiction) toward our goal, here (2.12), instead.[4]

---

[3] As an aside: [10] canonically codes *any size* finite sets in cubic time & decodes them in linear-time.

[4] More discussion on identifying contradictions with walls, a.k.a. boundaries, can be found on [47, Page 131].

Proof of Theorem 1.

By *two* applications of *linear-time*: krt, if-then-else (these from Section 2.1.2.1 above), and $\lambda \mathsf{E}, x.(\mathbf{T} \vdash_x \mathsf{E})$ (this from Section 2.1.2.2 above), from *any* φ^{TM}-program p, one *can algorithmically find* in linear-time (in $|p|$), programs $e_{1,p}$ and $e_{2,p}$ behaving as follows.

For each x, $\varphi^{\mathrm{TM}}_{e_{1,p}}(x) =$

$$\begin{cases} \varphi^{\mathrm{TM}}_p(x) + 1, & \text{if } \mathbf{T} \vdash_x \\ & \ll \varphi^{\mathrm{TM}}_{e_{1,p}} = \varphi^{\mathrm{TM}}_p \gg; \\ \varphi^{\mathrm{TM}}_p(x), & \text{otherwise;} \end{cases} \tag{2.13}$$

and $\varphi^{\mathrm{TM}}_{e_{2,p}}(x) =$

$$\begin{cases} 0, & \text{if } \mathbf{T} \vdash_x \ll \varphi^{\mathrm{TM}}_{e_{2,p}} = \varphi^{\mathrm{TM}}_p \gg; \\ \varphi^{\mathrm{TM}}_p(x), & \text{otherwise.} \end{cases} \tag{2.14}$$

Let $g \in \mathcal{L}\mathrm{time}$ be such that, for each p, $D_{g(p)} = \{e_{1,p}, e_{2,p}\}$. We consider cases re p for the choice of the associated $q \in D_{g(p)}$.

Case (1). domain$(\varphi^{\mathrm{TM}}_p)$ ∞. Suppose for contradiction, for some x, $\mathbf{T} \vdash_x \ll \varphi^{\mathrm{TM}}_{e_{1,p}} = \varphi^{\mathrm{TM}}_p \gg$. Since, by assumption, $\mathbf{T}$ *does not prove false such sentences*, $\varphi^{\mathrm{TM}}_{e_{1,p}} = \varphi^{\mathrm{TM}}_p$, and by (2.13) above, for all $x' \geq x$, $\varphi^{\mathrm{TM}}_{e_{1,p}}(x')$ *also* $= \varphi^{\mathrm{TM}}_p(x') + 1$, but, since domain$(\varphi^{\mathrm{TM}}_p)$ ∞, we have a contradiction. Choose $q = e_{1,p}$. Then, trivially, again by (2.13), $\varphi^{\mathrm{TM}}_q = \varphi^{\mathrm{TM}}_p$, but $\mathbf{T}$ does not prove it.

Case (2). domain$(\varphi^{\mathrm{TM}}_p)$ finite. Suppose for contradiction, for some x, $\mathbf{T} \vdash_x \ll \varphi^{\mathrm{TM}}_{e_{2,p}} = \varphi^{\mathrm{TM}}_p \gg$. Since, by assumption, $\mathbf{T}$ *does not prove false such sentences*, $\varphi^{\mathrm{TM}}_{e_{2,p}} = \varphi^{\mathrm{TM}}_p$, and by (2.14) above, for all $x' \geq x$, $\varphi^{\mathrm{TM}}_{e_{2,p}}(x')$ also $= 0$, making domain$(\varphi^{\mathrm{TM}}_{e_{2,p}})\infty$, and, hence, domain$(\varphi^{\mathrm{TM}}_p)\infty$, a contradiction. Choose $q = e_{2,p}$. Then, trivially, again by (2.14), $\varphi^{\mathrm{TM}}_q = \varphi^{\mathrm{TM}}_p$, but $\mathbf{T}$ does not prove it.

In each case, by if-then-else and **krt** being linear-time (hence, at most linear growth) functions, $\lambda \mathsf{E}, x.(\mathbf{T} \vdash_x \mathsf{E}) \in \mathcal{L}\mathrm{time}$, and by the complexity upper bounds (2.2) and (2.4) (in Section 2.1.2.1 above) as well as the assertion (in Section 2.1.2.2 above) of

the linear-time (and, hence, linear size) cost of substituting numerals into formulas of **PA**, we have small positive c, d such that the theorem's time complexity bound (2.10) and it's program size bound (2.11) above each hold.

□ Theorem 1

Next is the promised, motivated, concrete example.

Putnam [40] notes that the typical inductive definitions of grammaticality (i.e., well-formedness) for propositional logic formulas parallel the typical definitions of truth (under any truth-value assignment to the propositional variables) for such formulas, and that the first kind of inductive definition provides a *short and feasible* decision program for grammaticality.[5] He goes on to say, though, that the other ways of providing *short and feasible* inductive definitions of such grammaticality which also parallel an inductive definition of truth are so similar as to constitute *intrinsic* grammars (and semantics). Let p be one of these *typical* short and fast decision procedures for propositional calculus grammaticality expressed naturally and directly as a φ^{TM}-program. Then by Theorem 1 above and its proof also above, there is an obviously semantically equivalent φ^{TM}-program q only slightly linearly more complex than p in size and run time (so it too is short and feasible); q also provides the same inductive definition of grammaticality as p which, then, parallels the truth definition like p does (after all the else part of q *is* p and the if-part of q never comes true); but the unprovability (in pre-assigned **T**) of the semantic equivalence of q with p makes q a bit peculiar as an *intrinsic* grammar for propositional logic, providing a basis to doubt Putnam's assertion. However, we do note that *intensionally* [44] q is a bit unlike p — since it performs an always false (quick) test p doesn't.

2.2.2 Further Results

It's interesting to ask: can the condition $|D_{g(p)}| = 2$ in Theorem 1 be improved to $|D_{g(p)}| = 1$? If so, it makes sense to replace a singleton set, $\{q\}$, by just q and

[5] In computer science these inductive definitions would be called recursive.

use $g(p) = q$ (not the code of $\{q\}$). Anyhow, the answer to the question is, No (see Theorem 2 below). Before we present and prove this theorem, it is useful to have for its proof the unsurprising lemma (Lemma 1) just below.[6]

Lemma 1. *If* $\varphi_p^{\mathrm{TM}}(x)\!\downarrow = y$, *then*

$$\mathbf{PA} \vdash \ll \varphi_p^{\mathrm{TM}}(x)\!\downarrow = y \gg . \tag{2.15}$$

Proof of Lemma 1. The relation, in p, x, y, t, that holds iff $\varphi_p^{\mathrm{TM}}(x)\!\downarrow = y$ within t steps, where the steps are measured by the natural Φ^{TM}, is trivially computable (a.k.a. recursive) [3].

Suppose $\varphi_p^{\mathrm{TM}}(x)\!\downarrow = y$. Then there is some t such that $\varphi_p^{\mathrm{TM}}(x)\!\downarrow = y$ within t steps. By Gödel's Lemma [21, 35] that recursive relations are numeralwise provably-representable in, e.g., **PA**, $\mathbf{PA} \vdash \ll \varphi_p^{\mathrm{TM}}(x)\!\downarrow = y$ within t steps $\gg$. By existential generalization *inside* **PA**, we have $\mathbf{PA} \vdash \ll (\exists t)[\varphi_p^{\mathrm{TM}}(x)\!\downarrow = y$ within t steps$] \gg$. Hence, $\mathbf{PA} \vdash \ll \varphi_p^{\mathrm{TM}}(x)\!\downarrow = y \gg$. □ Lemma 1

The next theorem implies that, in Theorem 1 above, the condition $|D_{g(p)}| = 2$ can*not* be improved to $|D_{g(p)}| = 1$ (or equivalent as discussed above). The proof of this next theorem (Theorem 2) provides positive cases re proving true program properties in **PA**.

Theorem 2. *It is* not *the case that there exists computable* g *such that, for any* p, *for* $q = g(p)$,

$$\mathbf{T} \nvdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg . \tag{2.16}$$

Proof of Theorem 2. Suppose for contradiction otherwise.

Suppose d is a φ^{TM}-program for g, i.e., suppose $\varphi_d^{\mathrm{TM}} = g$.

Of course $\lambda p, x.[\varphi^{\mathrm{TM}}_{\varphi_d^{\mathrm{TM}}(p)}(x)]$ is partially computable, and, importantly, this is constructively provable in **PA**. Later in this proof we explain why we want this constructivity; for now, we sketch how we know the constructive provability in **PA**.

[6] We bother to prove it since we do not know a citation for its proof.

For example, one step in showing the constructivity is to explicitly construct a φ^{TM} universal program u so that its detailed correctness is (trivially, albeit tediously) constructively provable in **PA**. In particular,

$$\mathbf{PA} \vdash \ll (\forall p, x)[\varphi_u^{\mathrm{TM}}(p, x) = \varphi_p^{\mathrm{TM}}(x)] \gg. \tag{2.17}$$

For φ^{TM}, the construction of a relatively efficient, but *time-bounded variant* of such a u is outlined in the proof of [47, Theorem 3.6]. This construction could be altered to remove the time-boundedness and just get a suitable u.

Another step would be to spell out a φ^{TM}-program c for a computable function comp for computing a φ^{TM}-program for the composition of the (partial) functions computed by its φ^{TM}-program arguments as in [47, Lemma 3.10] and its proof — where, again, **PA** constructively proves correctness (including comp $= \varphi_c^{\mathrm{TM}}$ is total).

Relevance of u and c: clearly we have,

$$\varphi^{\mathrm{TM}}_{\varphi_d^{\mathrm{TM}}(p)}(x) = \varphi_u^{\mathrm{TM}}(\varphi_d^{\mathrm{TM}}(p), x), \tag{2.18}$$

and the right-hand side of (2.18) just above is a composition and, then, can be further expanded employing c.[7] With u and c, then, we can explicitly compute a φ^{TM}-program for $\lambda p, x\,.[\varphi^{\mathrm{TM}}_{\varphi_d^{\mathrm{TM}}(p)}(x)]$ and constructively prove it correct in **PA**.

So, then, by Constructive Kleene's Second Recursion Theorem *but without the parameter p as above in Section 2.1.2.1*, we have a (self-referential) p_0 such that, for any x,

$$\varphi_{p_0}^{\mathrm{TM}}(x) = \varphi^{\mathrm{TM}}_{\varphi_d^{\mathrm{TM}}(p_0)}(x). \tag{2.19}$$

Below we'll refer to this parameter-free version of the above Constructive Kleene Theorem as **KRT**. Then we have a φ^{TM}-program k for the above function **krt** *again with parameter p completely omitted*, and, with this k representing in the language of **PA** this modified version of the function **krt**, **KRT** is completely constructively provable in **PA**.

---

[7] Further below in Section 2.2.2, we'll consider, among other things, some programming systems with provability subtleties re universality and/or composition.

Hence, by our remarks above about obtaining and constructively proving correct a program for $\lambda p, x.[\varphi^{\mathrm{TM}}_{\varphi^{\mathrm{TM}}_d(p)}(x)]$, we can obtain a p_0 as in (2.19) and constructively prove it correct in **PA**, we have in particular,

$$\mathbf{PA} \vdash \ll \varphi^{\mathrm{TM}}_{p_0} = \varphi^{\mathrm{TM}}_{\varphi^{\mathrm{TM}}_d(p_0)} \gg. \tag{2.20}$$

Now, here's why we care about the constructivity in **PA**. Existence proofs in **PA** in general do *not* deliver a numerical value such as p_0 above for proved existence statements, but constructive proofs do [53, 52], and we need this numerical value from **PA**'s constructive proof of the employed special case of the Kleene Theorem to know (2.20) just above.

However, we don't know enough about g (and d) to know whether we can prove g's totality in **PA** — including by representing g as φ^{TM}_d; fortunately, we won't need that.

We do know (at least outside **PA**) that g is total (since it's a consequence of g's assumed computability). Hence, we know (at least outside **PA**) that $g(p_0)\downarrow$. Since, from above, d is a φ^{TM}-program for g, we have that $\varphi^{\mathrm{TM}}_d(p_0)\downarrow$ = to some explicit numerical value q_0. Therefore, from Lemma 1 above,

$$\mathbf{PA} \vdash \ll \varphi^{\mathrm{TM}}_d(p_0)\downarrow = q_0 \gg . \tag{2.21}$$

Hence, by substitution of equals for equals and reflexivity of equals *inside* **PA**, (2.20), and (2.21),

$$\mathbf{PA} \vdash \ll \varphi^{\mathrm{TM}}_{q_0} = \varphi^{\mathrm{TM}}_{p_0} \gg, \tag{2.22}$$

a contradiction to our beginning assumption — since **T** extends **PA**.

□ Theorem 2

So far we have considered the natural, deterministic complexity theory relevant, acceptable system, φ^{TM}. After we got Theorem 2 just above showing a condition in Theorem 1 further above (in Section 2.2.1) couldn't be improved, we wondered if there were some (possibly not quite so natural but, perhaps, still acceptable) systems ψ for

which we don't have the just above Theorem 2. We initially obtained the first part of the next theorem (Theorem 3) which provides such a ψ, *but* we didn't, then, know whether our ψ was acceptable. We subsequently obtained Theorem 3's furthermore clause providing our ψ's acceptability *together with a surprise we didn't expect*. We explain the surprise after the statement of Theorem 3 and before its proof.

Theorem 3. *There is a programming system ψ and a computable g such that, for all p, $\psi_{g(p)} = \psi_p$, yet, for $q = g(p)$, $\mathbf{T} \nvdash \ll \psi_q = \psi_p \gg$.*

Furthermore, ψ is acceptable, and, surprisingly,

$$(\forall p)[\psi_p = \varphi^{\mathrm{TM}}_p]. \tag{2.23}$$

How can (2.23) be true — in the light of the rest of the just above theorem (Theorem 3)? It seems to contradict Theorem 2 further above. The answer is that, in the proof just below of the just above theorem (Theorem 3), the needed ψ is, in effect, *defined by* an unusual φ^{TM}-program e in (2.26, 2.27) below, and, *in the language of* **PA**, for formulating (un)provability *about* ψ in **T**, ψ is, of course, *represented by its defining* e.[8] φ^{TM} itself, on the other hand, can be and is understood to be naturally (not unusually) represented in the language of **PA**.[9]

To aid us in some proofs below, including that of the above Theorem 3, we present the following lemma (Lemma 2), where the recursion theorem part of its proof is from H. Friedman [19].

Lemma 2 (**T**-Provable Padding-Once)**.** *Suppose φ is any acceptable programming system such that* **T** *proves φ's acceptability.*

Then, there is a total computable function g such that for any p, $g(p) \neq p$, but $\varphi_{g(p)} = \varphi_p$.[10]

---

[8] An original source for unusual representations in arithmetic (as is our e) is [18].

[9] See the informal discussion about the notation $\ll \mathsf{E} \gg$ in Section 2.1.2.2 above, where, in effect, the particular example $\ll \varphi^{\mathrm{TM}}_p$ is total $\gg$ is employed.

[10] Of course, the more general, constructive *infinite* padding holds [33], but we do not need that for the results of this chapter.

Furthermore, this padding-once result is, then, expressible and provable in $\mathbf{T}$.

PROOF OF LEMMA 2. Assume the hypothesis, i.e., that $\mathbf{T}$ proves φ's acceptability.

Then $\mathbf{T}$ proves Kleene's S-m-n Theorem, so we obtain that $\mathbf{T}$ proves the Parameterized Second Kleene Recursion Theorem (as above in Section 2.1.2.1, but with witnessing functions not necessarily in $\mathcal{L}$time, and we don't need in *this* result, to have $\mathbf{T}$ "calculate" numerals for the self-referential programs).

Then, from this Kleene Theorem, we have a computable function f such that, for each p, x,

$$\varphi_{f(p)}(x) = \begin{cases} \varphi_p(x), & \text{if } f(p) \neq p; \\ \varphi_{p+1}(x), & \text{if } f(p) = p. \end{cases} \tag{2.24}$$

Then, let g be defined as follows.

$$g(p) = \begin{cases} f(p), & \text{if } f(p) \neq p; \\ p+1, & \text{if } f(p) = p. \end{cases} \tag{2.25}$$

We consider two cases.

Case one: $f(p) \neq p$. Then, from (2.24), $\varphi_{f(p)} = \varphi_p$, and, from (2.25), $g(p) = f(p) \neq p$.

Case two: $f(p) = p$. Then, from (2.24), $\varphi_{f(p)} = \varphi_{p+1}$, which, by Case two, $= \varphi_p$. From (2.25), $g(p) = p + 1 \neq p$.

By this case-analysis, g satisfies Padding-Once. The above is so simple as to be provable in $\mathbf{T}$ — as needed. □ LEMMA 2

PROOF OF THEOREM 3. By Kleene's second recursion theorem (again without parameter), there is a (self-referential) φ^{TM}-program e and an associated ψ both such that, for each p, x,

$$\psi_p(x) \stackrel{\text{def}}{=} \varphi^{\text{TM}}_e(\langle p, x \rangle), \text{ which } = \tag{2.26}$$

$$\begin{cases} p, & \text{if } \mathbf{T} \vdash_x \\ & \ll (\exists q, r \mid q \neq r)[\psi_q = \psi_r] \gg; \\ \varphi_p^{\mathrm{TM}}(x), & \text{otherwise.} \end{cases} \tag{2.27}$$

N.B. The mentions of ψ in (2.27) just above with variable subscripts q, r should be understood, employing ψ's definition (2.26) above, to be $\varphi_e^{\mathrm{TM}}(\langle q, \cdot \rangle), \varphi_e^{\mathrm{TM}}(\langle r, \cdot \rangle)$, respectively.

Claim 1. $\mathbf{T} \nvdash \ll (\exists q, r \mid q \neq r)[\psi_q = \psi_r] \gg$.

Proof of Claim 1. Suppose for contradiction otherwise.

Then there exists an x_0 such that $\mathbf{T} \vdash_{x_0} \ll (\exists q, r \mid q \neq r)[\psi_q = \psi_r] \gg$. However, then, by (2.26, 2.27) above, we have $(\forall p)[\psi_p(x_0)\downarrow = p]$, and thus $(\forall p, q \mid p \neq q)[\psi_p(x_0) \neq \psi_q(x_0)]$; therefore, $\mathbf{T}$ has proven a sentence of first order arithmetic which is false in the standard model, a contradiction. □ Claim 1

Claim 2. $(\forall p)[\psi_p = \varphi_p^{\mathrm{TM}}]$*; hence,* ψ *is acceptable.*

Proof of Claim 2. By Claim 1, the first clause in (2.27) above is false for each p, x. Therefore, by (2.26, 2.27) above, $(\forall p, x)[\varphi_e^{\mathrm{TM}}(\langle p, x \rangle) = \varphi_p^{\mathrm{TM}}(x)]$; hence, $(\forall p)[\psi_p = \varphi_p^{\mathrm{TM}}]$ — making ψ acceptable too. □ Claim 2

Claim 3. *There is a computable* g *such that, for all* p, $g(p) \neq p$, $\psi_{g(p)} = \psi_p$, *and, for* $q = g(p)$, $\mathbf{T} \nvdash \ll \psi_q = \psi_p \gg$.

Proof of Claim 3. The acceptability of φ^{TM} is provable in **PA**, hence, in $\mathbf{T}$. By Lemma 2, there exists a computable g such that $(\forall p)[g(p) \neq p \ \wedge \ \varphi_{g(p)}^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}]$. By Claim 2, $\psi = \varphi^{\mathrm{TM}}$; thus, for this *same* g, $(\forall p)[\psi_{g(p)} = \psi_p]$.

Suppose arbitrary p is given. Let $q = g(p)$ Suppose for contradiction, $\mathbf{T} \vdash \ll \psi_q = \psi_p \gg$. Clearly by Gödel's Lemma (employed in the proof of Lemma 1 above),

$\mathbf{PA} \vdash \ll q \neq p \gg$. Then, by this and existential generalization in $\mathbf{T}$, $\mathbf{T} \vdash \ll (\exists q, r \mid q \neq r)[\psi_q = \psi_r] \gg$, a contradiction to Claim 1 above. □ Claim 3

□ Theorem 3

For the next three corollaries (Corollaries 1, 2, and 3), the mentioned ψ is that from Theorem 3 and its proof, including (2.26, 2.27) above.

Corollary 1. $\psi = \varphi^{\mathrm{TM}}$, *but* $\mathbf{T} \nvdash \ll \psi = \varphi^{\mathrm{TM}} \gg$.

Proof of Corollary 1. $\psi = \varphi^{\mathrm{TM}}$ is from Theorem 3 above. Suppose for contradiction $\mathbf{T} \vdash \ll \psi = \varphi^{\mathrm{TM}} \gg$.

Then, from this and the proof of Theorem 2 above, one obtains a Theorem 2 *but* with ψ replacing φ^{TM}. This contradicts Theorem 3 above (which is also about ψ).

□ Corollary 1

To understand the corollary (Corollary 2) and its proof just below, it may be useful to review the roles of φ^{TM}-programs u, c, k in the proof of Theorem 2 above. This corollary says there can be *no* analog of *all three* of these programs for ψ (in place of φ^{TM}).

Corollary 2. *There are* no u, c, k *such that* simultaneously:

$$\mathbf{T} \vdash \ll u \textit{ is a witness to universality in } \psi \gg, \tag{2.28}$$

$$\mathbf{T} \vdash \ll c \textit{ is a witness to composition in } \psi \gg, \textit{\&} \tag{2.29}$$

$$\mathbf{T} \vdash \ll k \textit{ is a witness to } \mathbf{KRT} \textit{ in } \psi \gg . \tag{2.30}$$

Proof of Corollary 2. Suppose for contradiction otherwise. Then, enough is provable in $\mathbf{T}$ *about* ψ to make Theorem 2 above *also* provable for ψ — in place of φ^{TM}. This contradicts Theorem 3 about ψ. □ Corollary 2

Re Corollary 2 just above, it is well known that, from [33, 34], Kleene's S-m-n can be constructed out of a program c for composition and, then, Kleene's proof of **KRT** can be done from S-m-n; so, it might appear that (2.30) just above could be eliminated. This is actually open. The reason is that, while each of these just mentioned constructions requires some easily existing auxiliary ψ-programs, for Corollary 2 we'd ostensibly also need these auxiliary ψ-programs to be **T**-provably correct.[11] The **T**-provable correctness is the hard part.

Corollary 3. $\mathbf{T} \nvdash \ll \psi$ *is acceptable*$\gg$.

Proof of Corollary 3. Assume for contradiction otherwise. Then, by Lemma 2 above, $\mathbf{T} \vdash \ll (\exists p)[\psi_p \text{ is total } \wedge\ (\forall q)(\exists r = \psi_p(q) \mid r \neq q)[\psi_q = \psi_r] \gg$.

From this we have, $\mathbf{T} \vdash \ll (\exists q, r \mid q \neq r)[\psi_q = \psi_r] \gg$, a contradiction to Claim 1 above. □ Corollary 3

The next and last two theorems of the present chapter (Theorems 4 and 5) provide two more acceptable programming systems, η, θ, respectively, each defined (as was ψ above) by respective, unusual φ^{TM}-programs. The first of these theorems (Theorem 4) provides a surprise, part positive, part negative, regarding proving *in* **T** that *universality* holds for η. The contrast between these last two theorems of the prsent chapter is also interesting. Of course, since each of η, θ is acceptable, universality holds for each of them (at least outside **T**).

Below, for (partial) function ξ, $\rho(\xi)$ denotes the *range* of ξ.

Theorem 4. *There exists an acceptable programming system* η *and an* e *such that* $\mathbf{PA} \vdash \ll e$ *is universal for* $\eta \gg$*, yet, surprisingly, for each* p*,*

$$\text{If } \mathbf{T} \vdash \ll p \text{ is universal for } \eta \gg, \text{ then } p = e. \tag{2.32}$$

[11] Machtey and Young's construction [33, 34] of an S-m-n function out of a composition function, for example, employs auxiliary ψ-programs q_0, q_1 such that

$$\psi_{q_0} = \lambda z\,.\langle 0, z\rangle; \text{ and } \psi_{q_1} = \lambda\langle y, z\rangle\,.\langle y+1, z\rangle. \tag{2.31}$$

Marcoux's more efficient solution [34] employs three such auxiliary ψ-programs.

Of course, in η, there are infinitely many universal programs, but exactly one *provably so in* $\mathbf{T}$.

Proof of Theorem 4. Kleene Second Recursion Theorem provides a φ^{TM}-program e and an associated η both such that, for each p, x,

$$\eta_p(x) \stackrel{\text{def}}{=} \varphi_e^{\mathrm{TM}}(\langle p, x\rangle), \text{ which } = \tag{2.33}$$

$$\begin{cases} p, & \text{if } [p \neq e \ \wedge \ \mathbf{T} \vdash_x \\ & \ll (\exists q, r \mid r \neq q)[\eta_q = \eta_r] \gg]; \\ \varphi_p^{\mathrm{TM}}(x), & \text{otherwise.} \end{cases} \tag{2.34}$$

Of course, since **KRT** for φ^{TM} is constructively provable in **PA**, we can get the numeral for e inside **PA** as well as the universally quantified equation just above for the value of $\varphi_e^{\mathrm{TM}}(\langle p, x\rangle)$ by cases.

Claim 4. $\mathbf{T} \nvdash \ll (\exists q, r \mid q \neq r)[\eta_q = \eta_r]$

Proof of Claim 4. Assume for contradiction that $\mathbf{T} \vdash \ll (\exists q, r \mid q \neq r)[\eta_q = \eta_r] \gg$. Let x_0 be the minimum number of steps in any such proof. Also, since $\mathbf{T}$ does not prove false things like the just above,

$$(\exists q, r \mid q \neq r)[\eta_q = \eta_r]. \tag{2.35}$$

Then, for $f(p) = \varphi_e^{\mathrm{TM}}(\langle p, x_0\rangle)$, $\rho(f) \supseteq (\mathbb{N} - \{e\})$. Clearly, $\eta_e = \varphi_e^{\mathrm{TM}}$, and η_e has infinite range. Furthermore, $(\forall p \neq e)(\forall x \geq x_0)[\eta_p(x) = p]$; therefore, $(\forall p \neq e)[\eta_p$ has finite range], and, thus, there is no η-program whose code number is not e whose computed function is equal to η_e. Furthermore, $(\forall p, q \mid p \neq e \ \wedge \ p \neq q \ \wedge \ q \neq e)[\eta_p(x_0) = p \ \wedge \ \eta_q(x_0) = q]$, thus there are no two distinct programs that compute the same function, a contradiction to (2.35). □ Claim 4

Claim 5. $(\forall p, x)[\eta_p(x) = \varphi_p^{\mathrm{TM}}(x)]$*; hence, η is acceptable.*

Proof of Claim 5. By Claim 4, the if clause of (2.34) is always false, hence, by (2.33, 2.34), $(\forall p, x)[\eta_p(x) = \varphi_e^{\mathrm{TM}}(\langle p, x\rangle) = \varphi_p^{\mathrm{TM}}(x)]$. □ Claim 5

Claim 6. *There does not exist p, q such that $p \neq q$, and $\mathbf{T} \vdash \ll \eta_p = \eta_q \gg$.*

Proof of Claim 6. Suppose for contradiction otherwise. Then by existential generalization in $\mathbf{T}$, we obtain a contradiction to Claim 4. □ Claim 6

Claim 7. $\mathbf{PA} \vdash \ll e$ *is universal for* $\eta \gg$.

Proof of Claim 7. We need *not* prove in **PA** that η is a programming system for the 1-argument partial computable functions. Instead, it suffices for us to argue only that

$$\mathbf{PA} \vdash \ \ll (\forall p, x)[\eta_e(< p, x >) = \eta_p(x)] \gg. \tag{2.36}$$

Then, from (2.33) above, the definition of η by e in the φ^{TM}-system, applied to each side of (2.36), it, then, suffices to show that $\mathbf{PA} \vdash$

$$\ll (\forall p, x)[\varphi_e^{\mathrm{TM}}(\langle e, \langle p, x\rangle\rangle) = \varphi_e^{\mathrm{TM}}(\langle p, x\rangle)] \gg. \tag{2.37}$$

By the otherwise case of (2.33, 2.34) above, applied to $\ll \varphi_e^{\mathrm{TM}}(\langle e, \langle p, x\rangle\rangle) \gg$, where p, x are variables (not numerals), we get its provable in **PA** value to be $\ll \varphi_e^{\mathrm{TM}}(\langle p, x\rangle) \gg$ — again with p, x variables. This together with universal generalization *inside* **PA** on the variables p, x, verifies *in* **PA** the sufficient (2.37) just above. □ Claim 7

Claim 8. *For all $p \neq e$, $\mathbf{T} \nvdash \ll p$ is universal in $\eta \gg$.*

Proof of Claim 8. Immediate from Claims 6 and 7. □ Claim 8

□ Theorem 4

Theorem 5. *There exists an acceptable programming system θ such that, for each u,*

$$\mathbf{T} \nvdash \ll u \text{ is universal in } \theta \gg . \tag{2.38}$$

Of course, in θ, there are infinitely many universal programs, but none *are provably so in* $\mathbf{T}$.

Proof of Theorem 5. Kleene's Recursion Theorem provides a φ^{TM}-program e and an associated θ both such that, for each p, x,

$$\theta_p(x) \stackrel{\text{def}}{=} \varphi_e^{\text{TM}}(\langle p, x\rangle) \text{ which } = \tag{2.39}$$

$$\begin{cases} p, & \text{if } \mathbf{T} \vdash_x \\ & \ll (\exists u)[u \text{ is universal in } \theta] \gg; \\ \varphi_p^{\text{TM}}(x), & \text{otherwise.} \end{cases} \tag{2.40}$$

Assume for contradiction that $\mathbf{T} \vdash \ll (\exists u)[u \text{ is universal in } \theta] \gg$. Then, since $\mathbf{T}$ does not prove false things of this sort, universality holds in θ.

Let x_0 be the smallest number of steps in any proof as is assumed just above to exist.

Then, since $(\forall p, x \mid x \geq x_0)[\theta_p(x) = p]$, we have, $(\forall p)[|\rho(\theta_p)| \leq 1 + x_0]$. By contrast, $\rho(\theta) = \mathbb{N}$. Then there is no p such that $\rho(\theta_p) = \rho(\theta)$; therefore, there cannot be any universal programs for θ, a contradiction.

Therefore, $\mathbf{T} \nvdash \ll (\exists u)[u \text{ is universal in } \theta] \gg$, and, thus, $(\forall p, x)[\theta_p(x) = \varphi_p^{\text{TM}}(x)]$. This makes θ acceptable.

Furthermore, it is not the case that $(\exists u)[\mathbf{T} \vdash \ll u \text{ is universal in } \theta \gg]$, since, if $\mathbf{T}$ proved such a thing, it would immediately follow from Existential Generalization *in* $\mathbf{T}$ that $\ll (\exists u)[u \text{ is universal in } \theta] \gg$ is provable in $\mathbf{T}$, which has already been shown not to be provable by $\mathbf{T}$. □ Theorem 5

We expect that analogs of Theorems 4 and 5 just above can be obtained for other properties besides universality, but we did not pursue this herein.

Chapter 3

BEYOND ROGERS' NON-CONSTRUCTIVELY COMPUTABLE FUNCTION

3.1 Introduction

Rogers [44, Page 9] defines functions f and, then, g — each based on eventual patterns in the decimal expansion of π.[1] We'll discuss each of these in turn (in Sections 3.1.1 and 3.1.2), but opposite the order in which Rogers discusses them.[2]

3.1.1 Our f and Variants

As will be seen, the second of these Rogers' functions, which we're calling f, is clearly computable *but not constructively so* — *and* there is currently apparently *no known* way to compute it.

For each $x \in \mathbb{N} = \{0, 1, 2, \ldots\}$, let $f(x) \stackrel{\text{def}}{=} 1$, if $\exists$ at least x consecutive 5's in π's decimal expansion; 0, otherwise.

Clearly, f is either constantly 1 — in case (i) π's decimal expansion has arbitrary long, finite runs of consecutive 5's — *or* f steps exactly once from 1 to 0 — (ii) otherwise. In each possibility f is trivially computable: in case (i), employing Church's Lambda Notation [44], $f = \lambda x.(1)$; in case (ii), if n is the maximum length of any consecutive sequences of 5s in the decimal expansion of π, $f = \lambda x.(1 \text{ if } x \leq n; 0 \text{ otherwise})$. Hence, in any case, f is trivially computable.

It's *un*known *which* of cases (i) and (ii) just above holds, *and*, if (ii) became known, it might still be unknown the exact size of a largest consecutive run of 5's in

---

[1] Brouwer first similarly employed patterns in π's digits, e.g., [7].

[2] To preserve alphabetical order in *our* discussion, we'll call his g, f, and his f, g.

π's decimal expansion. Then we still wouldn't know exactly where f steps from 1 to 0 — and we still wouldn't know how to compute f.

This Rogers' example very nicely illustrates the (classical) conceptual difference between, *there is an algorithm for* f and *the human race knows an algorithm for* f: the former is true, but the latter is currently false. *However*, in the future, the above *un*knowns about the decimal expansion of π may be suitably resolved, so that, then, Rogers' example will no longer illustrate this just above pleasant conceptual difference.[3]

As will be seen, our Theorems 7 and 8 below provide *"safer"* replacements for f — so as to preserve an interesting version of the nice (classical) conceptual difference above.

Suppose **T** is any fixed, *computably axiomatized* extension of first order Peano Arithmetic (**PA**) [35, 44]. E.g., **T** might be: **PA** itself, the also first order variant with variables for both natural numbers and for sets thereof and with induction expressed for sets (e.g., from [44]), or **ZFC**. We *also need* that (certain of) **T**'s theorems which are *expressible in* **PA** are *true* (essentially those featured in the proofs of Theorems 7 and 8 below). **T** being computably axiomatized makes its set of theorems computably enumerable, so **T** plays the role of an *algorithmic extractor of (relevant) true information*.

Except for π, from now on, we'll restrict our attention to reals $\mathbf{r}$ in the interval $[0, 1]$ represented as an infinite expansion in *binary* (not in base 10).[4]

We say that such a real $\mathbf{r} = .r_0r_1r_2\ldots$, where the r_js are its successive binary digits, is *computable* iff the function $\lambda j.(r_j)$ is computable.

---

[3] A future proof as to *which* function is f may still be non-constructive, but, importantly for the present chapter, re the nice conceptual difference above, we would have the *less* interesting (and more normal) situation that both *there is an algorithm for* f and *the human race knows an algorithm for* f are true — of course, classically.

[4] There can be two expansions for a single real, e.g., $01000\cdots$ vs. $001111\cdots$, but this causes us no problem herein.

For a real $\mathbf{r}$, for each $x \in \mathbb{N}$, we let

$$f_{\mathbf{r}}(x) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \exists \textit{ at least } x \textit{ consecutive } 1\text{'s in } \mathbf{r}\text{'s } \textit{binary} \text{ expansion;} \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

Clearly, $f_{\mathbf{r}}$, just as is f, is trivially computable — even if $\mathbf{r}$ is not computable. Again, even for *computable* $\mathbf{r}$, it may, in some cases, be hard to know that some particular program q *is* a program for $f_{\mathbf{r}}$.

We can and do consider the *acceptable programming systems* (synonym: *acceptable numberings*) [43, 44, 33, 41, 42, 46] as those programming systems for the 1-argument partial computable functions: $\mathbb{N} \to \mathbb{N}$ which are intercompilable with naturally occurring general purpose programming formalisms such as a full Turing machine formalism or a LISP programming language. Typically, for theoretical work, one works with *numerical* names for the programs in these systems — whence the term 'numbering.' Let φ be any fixed acceptable programming system. φ_p denotes the 1-argument partial computable function: $\mathbb{N} \to \mathbb{N}$ and computed by program (number) p in the φ-system. We write W_p for the domain of φ_p, i.e., the c.e. set accepted by φ-program p. Below $\downarrow$ means converges or is defined, and $\uparrow$ means diverges or is undefined.

If E is an expression such as 'φ_p is total', where p is a particular element of $\mathbb{N}$, we shall write $\ll \mathsf{E} \gg$ to denote a *naturally* corresponding, fixed standard cwff (closed well-formed formula) of **PA** which (semantically) expresses E. We 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 well known that cwffs extensionally equivalent may not be intensionally or even provably equivalent [18]. In what follows, when we use the $\ll \mathsf{E} \gg$ notation, it will always be for E that are easily and *naturally* (semantically) expressible in **PA** as $\ll \mathsf{E} \gg$.

'$\vdash$' denotes the provability relation, and '$\nvdash$' is its negation.

Our theorems 7 and 8 below each imply that there are *computable* reals $\mathbf{r}$ such that, for *any* p, q which compute $\mathbf{r}$ and $f_{\mathbf{r}}$, respectively,

$$\mathbf{T} \nvdash \ll q \text{ computes } f_{\varphi_p} \gg . \tag{3.2}$$

Of course, the word *any* just above is important, since, then, the desired result is not the result of only pathological choices of p, q for computing $\mathbf{r}$ and $f_{\mathbf{r}}$, respectively.

There is, though, some residual subtlety left in what is and is not actually allowed to contribute to the unprovability in (3.2). First what isn't subtle (then, what is): suppose, as above, p, q which compute $\mathbf{r}$ and $f_{\mathbf{r}}$, respectively. Then, trivially, φ_q's totality is provable if we already had a proof that $\varphi_q = f_{\varphi_p}$, so including φ_q's totality as part of a translation of q computes f_{φ_p} does not contribute to its *un*provability. What is subtle: φ_p's totality. Were it included in a translation of what we want to say is unprovable, then it might contribute to the overall unprovability. We'd like to minimize this contribution — to make the theorems just mentioned above stronger. A bit below, then, we'll consider *two* translations (Translations 1 and 2). Provability of the first one does *not* entail φ_p is total, but, instead, by Claim 14 below, it entails only W_p is infinite. Furthermore, Translation 1, which follows, is quite sensible.

Translation 1. $\ll (\forall x)[\varphi_q(x)\downarrow \leq 1 \;\;\&\;\; [\varphi_q(x) = 1 \implies (\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 1]] \;\&\; [\varphi_q(x) = 0 \implies (\forall y)(\exists z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 0]]] \gg$.

We'll abbreviate this first translation as $\ll q \text{ computes } f_{\varphi_p} \gg_{\mathbf{1}}$, so that, for Theorems 7 and 8 below, (3.2) above becomes

$$\mathbf{T} \nvdash \ll q \text{ computes } f_{\varphi_p} \gg_{\mathbf{1}} . \tag{3.3}$$

The second translation follows. It also sensible but weaker than the first. Its provability does not also entail that W_p is infinite.

Translation 2. $\ll (\forall x)[\varphi_q(x)\downarrow \leq 1 \;\;\&\;\; [\varphi_q(x) = 1 \implies (\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 1]] \;\&\; [\varphi_q(x) = 0 \implies \neg(\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 1]]] \gg$.

We'll abbreviate this second translation as $\ll q$ computes $f_{\varphi_p} \gg_{\mathbf{2}}$

As we will see, a stronger analog of our Theorem 7 also holds (for Translation 2 instead of Translation 1), namely, Theorem 9. However, by contrast, an interesting constructive opposite of Theorem 8 holds for Translation 2, namely Theorem 10.

Importantly, in the light of (3.3) above, the following is perhaps surprising. For Theorem 7, $f_{\mathbf{r}}$ is made $= \lambda x.(1)$. The same happens for the stronger Theorem 9. For Theorem 8, for each $n > 0$, a corresponding $f_{\mathbf{r}}$ is made $= \lambda x.(1$ if $x \leq n; 0$ otherwise). The solution to the *apparent* paradoxes, one for each of these three theorems, is that **T** does not and cannot know (i.e., prove) that theorem's information about *which* function $f_{\mathbf{r}}$ is!

3.1.2 Our g and Variants

The first of Rogers' functions [44, Page 9], which we're calling g, is defined thus. For each $x \in \mathbb{N}$, let $g(x) \stackrel{\text{def}}{=} 1$, if $\exists$ *exactly* x consecutive 5's in π's decimal expansion; 0, otherwise.

Rogers [44, Page 9] points out that it is *un*known whether g just above is computable. Of course in the future enough may become known about the distributions of runs of consecutive 5s in the decimal expansion of π for whether g is computable to be resolved. Again we'll consider only reals $\mathbf{r}$ (except for π), as in Section 3.1.1 above, which are restricted to the interval $[0, 1]$ and represented as an infinite expansion in *binary* (not in base 10).

For a such a real $\mathbf{r}$, for each $x \in \mathbb{N}$, we let

$$g_{\mathbf{r}}(x) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \exists \textit{ exactly } x \textit{ consecutive } 1\text{'s in } \mathbf{r}\text{'s } \textit{binary} \text{ expansion;} \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

For clarity as to what is meant by 'exactly' just above, we define a *consecutive run of exactly* x $1s$ to be one that is bounded on both sides by a 0.

Trivially, there are many examples, such as $\mathbf{r} = .00000\ldots$, so that $g_{\mathbf{r}}$ *is* computable. My advisor has given the next theorem (Theorem 6) as a course exercise with hints.

Theorem 6. *There is a primitive recursive* $\mathbf{r}$ *such that* $g_{\mathbf{r}}$ *is not computable.*

We'll omit the hints since we describe next a considerable improvement.

Our last theorem of the present chapter (Theorem 11) is about reals $\mathbf{r} = \lambda j\,.(r_j)$ computable in linear-time in $|j|$, the *length of* j (and *not* requiring prior computing of r_i, for any $i < j$); this theorem says that, for each computably enumerable set A, there is real $\mathbf{r} = \lambda j\,.(r_j)$ computable in linear-time in $|j|$ such that $g_{\mathbf{r}}$ is the characteristic function of $(A \cup \{0\})$; if we choose $A = K$, where K is the diagonal halting problem set from [44], then $g_{\mathbf{r}}$ is not computable.[5]

3.2 Preliminaries

The material in this section (Section 3.2) is largely important for our machine-*dependent*, *natural*-complexity theorem (Theorem 11) and its proof.

φ^{TM} is the specific fixed acceptable programming system from [47, Chapter 3] for the partial computable functions: $\mathbb{N} \to \mathbb{N}$; it is a system based on deterministic, *multi-tape* Turing machines (TMs). In this system the φ^{TM}-programs are *efficiently* given numerical names or codes. This efficient numerical coding guarantees that many simple operations run in linear-time in the *length* of input.

Φ^{TM} denotes the natural TM step counting complexity measure (also in [47, Chapter 3]) and associated with φ^{TM}. In the present chapter, we employ a number of complexity bound results from [47, Chapters 3 & 4] regarding $\Phi^{\text{TM}}, \varphi^{\text{TM}}$. These results will be clearly referenced as we use them.

For the purposes of this chapter, *linear-time computable* means computable in linear-time in the *length* of inputs — of course as measured by Φ^{TM} in the φ^{TM} system.

We let $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the fixed, 1-1, onto, *linear-time computable* pairing function from [47, Section 2.3]. Its respective left and right inverses, π_1 and π_2, are also *linear-time computable* [47, Section 2.3]. $\langle \cdot, \cdot \rangle$ enables us to restrict our attention to one-argument partial computable functions and still handle, with iterated coding

[5] In [24] there is a nice, unrelated conjecture about reals $\mathbf{r} = \lambda j\,.(r_j)$ computable in linear-time in j.

by $\langle \cdot, \cdot \rangle$, multiple argument cases. The linear-time computable functions π_1 and π_2 are employed below in the proof of the present chapter's last theorem (Theorem 11).

We employ the convenient discrete log function from [47, Page 22]: for each $x \in \mathbb{N}$, $\log(x) \stackrel{\text{def}}{=} (\lfloor \log_2(x) \rfloor, \text{ if } x > 1; 1, \text{ if } x \leq 1)$.

We let $\Phi^{\text{SlowedDownTM}}$ be the special, *slowed down* step-counting measure associated with the acceptable φ^{TM}-system from [47, Theorem 3.20]. In the proof of [47, Theorem 3.20], for the case of $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$, $\Phi^{\text{SlowedDownTM}}$ is obtained from the standard Φ^{TM} measure associated with φ^{TM} — in part by a standard log-factors slow down trick. $\Phi^{\text{SlowedDownTM}}$ has the nice property (among others) that the predicate

$$T \stackrel{\text{def}}{=} \lambda p, x, t.(1, \text{ if } \Phi_p^{\text{SlowedDownTM}}(x) \leq t; 0, \text{ otherwise}) \tag{3.5}$$

is *linear-time computable*!

In the proof of Claim 25 below (part of the proof of the present chapter's last theorem, Theorem 11), we'll make use of the following lemma from [47] concerning a very efficiently implemented *conditional* (i.e., *if-then-else*) control structure for $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$. In particular we'll employ the remark immediately following this lemma (Remark 1).

Lemma 3 (Lemma 3.14, [47])**.** *There is a $k > 0$ and a linear-time computable function* cond *such that, for all $a, b, c, x \in \mathbb{N}$,*

$$\varphi^{\text{TM}}_{\text{cond}(\langle a,b,c \rangle)}(x) = \begin{cases} \varphi^{\text{TM}}_b(x), & \text{if } \varphi^{\text{TM}}_a(x)\downarrow > 0; \\ \varphi^{\text{TM}}_c(x), & \text{if } \varphi^{\text{TM}}_a(x)\downarrow = 0; \\ \uparrow, & \text{otherwise}; \end{cases} \tag{3.6}$$

and

$$\Phi^{\text{TM}}_{\text{cond}(\langle a,b,c \rangle)}(x) \leq k \cdot \begin{cases} (\Phi^{\text{TM}}_a + \Phi^{\text{TM}}_b(x)) + 1, & \text{if } \varphi^{\text{TM}}_a(x)\downarrow > 0; \\ (\Phi^{\text{TM}}_a + \Phi^{\text{TM}}_c(x)) + 1, & \text{if } \varphi^{\text{TM}}_a(x)\downarrow = 0; \\ \uparrow, & \text{otherwise}. \end{cases} \tag{3.7}$$

Remark 1. *Note that from (3.7) just above, that, if φ^{TM}-programs a, b, c each run in time linear in input length, then so does φ^{TM}-program* $\text{cond}(\langle a, b, c \rangle)$.

3.3 Results

The notations, basic assumptions, and some employed technical observations are from the Introduction and Preliminaries (Sections 3.1 and 3.2) above. Recall from there that, for *any* choice of real $\mathbf{r}$, $f_{\mathbf{r}}$ is *always computable*.

3.3.1 Results About $f_{\mathbf{r}}$s

In what follows $u \dot{-} v \stackrel{\text{def}}{=}$

$$\begin{cases} 0, & \text{if } u < v; \\ u - v, & \text{if } u \geq v. \end{cases} \tag{3.8}$$

Theorem 7. *There is a computable real* $\mathbf{r}$ *such that, for any* p, q *which compute* $\mathbf{r}$ *and* $f_{\mathbf{r}}$*, respectively,* $\mathbf{T} \nvdash \ll q$ *computes* $f_{\varphi_p} \gg_{\mathbf{1}}$*. Furthermore,* $f_{\mathbf{r}} = \lambda x\,.(1)$*.*

Proof of Theorem 7. As above in Section 3.1, our expansion of $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$ is: $\mathbf{T} \vdash \ll (\forall x)[\varphi_q(x)\downarrow \leq 1 \;\;\&\;\; [\varphi_q(x) = 1 \implies (\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 1]] \;\&\; [\varphi_q(x) = 0 \implies (\forall y)(\exists z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 0]]] \gg_{\mathbf{1}}$.

Define program list as follows. list runs a theorem prover for $\mathbf{T}$ and algorithmically lists, in some order, all the (p, q) such that $\mathbf{T} \vdash\!\ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$.

The global variable seq has value at any point a finite sequence of bits with known canonical index, and its initial value is empty. We reconceptualize this initial value equivalently as an infinite sequence of known-to-be-undefined values.

Define no-input subroutine longest_run (with global variable seq) as follows, and let longest_run with no arguments stand for the value returned by a call to this subroutine: longest_run scans seq to find the longest subsequence of contiguous 1s in seq and returns the length of that subsequence.

Define subroutine outputs_ones (with global variable seq) as follows. On input x, outputs_ones scans seq for the first x contiguous known-undefined values in seq and defines them all to be 1.

Define subroutine setup_seq (with global variable seq) as follows.

On input (p, q), setup_seq runs q on all values between 0 and longest_run inclusive. If φ_q returns 0 on any of these values, setup_seq returns and does nothing.

Otherwise, let x be longest_run $+ 1$, and setup_seq runs q on x. If it returns a 0, then setup_seq calls outputs_ones on x and returns.

If $\varphi_q(x)$ returns a non-zero value, then setup_seq dovetails an enumeration of the values of φ_p, and compares them against the *corresponding* entries of seq. If any entries are defined as something different from φ_p's values, it returns and does nothing. If it finds a value v such that $\varphi_p(v)\downarrow$ and seq(v) is still known-undefined, it defines seq(v) to be $1 \dot{-} \varphi_p(v)$ and returns.

If none of the previously-listed stopping conditions are met, or if setup_seq runs a program that goes undefined, it runs forever.

Define program define_real (with global variable seq) as follows.

On input x, define_real does the following:

Set i to 0.

For $x + 1$ iterations, do:

Run another step of list, and if, by the end of that step, it outputs some (p, q), pass (p, q) to setup_seq.[6]

If seq(i) is known-undefined after that, define it equal to 0.

Increment i by 1.

Output the value of seq(x).

Claim 9. *If* setup_seq *is passed a* (p, q) *such that* $\mathbf{T} \vdash \ll q$ *computes* $f_{\varphi_p} \gg_{\mathbf{1}}$*, then it will terminate.*

Proof of Claim 9. Since $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, and $\mathbf{T}$ does not prove false things (of this sort), it follows that $(\forall x)[\varphi_q(x)\downarrow \leq 1]$. Thus, running q on a

[6] list can output either nothing or exactly one ordered pair on a given step.

finite set of values must terminate. As only finitely many elements of seq can be defined during a call to setup_seq, all calls to outputs_ones must terminate. The only remaining case is that $(\forall x \mid x \leq \text{longest\_run} + 1)[\varphi_q(x) = 1]$, from which it follows that $(\exists y)(\forall z \mid y \leq z < y + \text{longest\_run} + 1)[\varphi_p(z)\downarrow = 1]$, which provides a sequence of consecutive 1s that is longer than the longest sequence of consecutive 1s in seq, which means there is at least one value of φ_p which is defined equal to 1 whose corresponding value of seq is not defined to be equal to 1, and thus setup_seq will terminate in this case as well. □ Claim 9

Claim 10. *For any p, q, if $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, then passing (p, q) to* setup_seq *results in a* seq *such that the set of reals compatible with the defined values of* seq *does not include φ_p.*

Proof of Claim 10. Case one: for some x less than or equal to the value of longest_run, $\varphi_q(x) = 0$, in which case seq is already incompatible with φ_p as seq has a run of 1s which are longer than the longest such run in φ_p.

Case two: $\varphi_q(\text{longest\_run} + 1) = 0$, in which case setup_seq calls outputs_ones and makes seq incompatible with φ_p for the same reason as case one.

Case three: $(\forall x \mid x \leq \text{longest\_run} + 1)[\varphi_q(x) = 1]$. In this case, setup_seq will search for a value of φ_p whose corresponding entry in seq either doesn't match or is known-undefined and will be set to not match. By Claim 9, such a value will be found, after which seq will not be compatible with φ_p. □ Claim 10

Claim 11. *The program* define_real *does, in fact, define a real.*

Proof of Claim 11. outputs_ones clearly terminates, by Claim 9 and the definition of list, all setup_seq calls will terminate, and, on any input x to define_real, list is only run for a finite number of steps. Thus, define_real will always output a value from seq. The loop in define_real includes a step that ensures that the element of seq output by define_real will be defined, and all times when elements of seq are assigned values, they

are assigned a value of either 0 or 1. So, define_real computes a total $\{0,1\}$-valued function, a real. □ Claim 11

Claim 12. *For any p, q, if $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, then $\varphi_p \neq \varphi_{\text{define\_real}}$.*

Proof of Claim 12. By Claim 11, $\varphi_{\text{define\_real}}$ defines a real, and thus, if φ_p does not define a real, they are not equal. By Claim 10 and the fact that define_real's output is always compatible with seq, if φ_p does define a real, the real it defines is not equal to $\varphi_{\text{define\_real}}$. Therefore, $\varphi_p \neq \varphi_{\text{define\_real}}$. □ Claim 12

Claim 13. $f_{\varphi_{\text{define\_real}}} = \lambda x.(1)$.

Proof of Claim 13. For all positive n, by provable in **PA** (hence, in **T**) padding for the φ-system, there's an infinite set of ps & qs such that q computes $\lambda x.(1$ if $x \leq n$; 0 otherwise) *and* $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg$, so outputs_ones is called infinitely often. For such calls, it provides a sequence of consecutive 1s longer than prior such.

□ Claim 13

By Claims 11, 12, and 13, the statement of the theorem follows with $\mathbf{r} = \varphi_{\text{define\_real}}$. □ Theorem 7

In general, $\lambda x.(1)$ is only one possibility for $f_{\mathbf{r}}$, so the question naturally arises: does there exist computable real $\mathbf{r}$ such that $(\forall p, q)[\varphi_p = \mathbf{r} \implies \mathbf{T} \nvdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, where $f_{\mathbf{r}} \neq \lambda x.(1)$?

For *all* $f_{\mathbf{r}}$ of the form $\lambda x.(1$ if $x \leq n; 0$ otherwise), except for the case where $n = 0$, the answer is, *Yes*. For $f_{\mathbf{r}} = \lambda x.(1$ if $x = 0; 0$ otherwise), there is precisely one such real, $\lambda x.(0)$, and there exists p, q such that $\varphi_p = \lambda x.(0)$ & $\mathbf{PA} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$.

Theorem 8. *For every $n > 0$, there is a computable real $\mathbf{r}$ such that, for any p, q which compute $\mathbf{r}$ and $f_{\mathbf{r}}$, respectively, $\mathbf{T} \nvdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$. Furthermore, $f_{\mathbf{r}} = \lambda x.(1$ if $x \leq n; 0$ otherwise).*

PROOF OF THEOREM 8. As above in Section 3.1, our expansion of $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$ is: $\mathbf{T} \vdash \ll (\forall x)[\varphi_q(x)\downarrow \leq 1 \;\;\&\;\; [\varphi_q(x) = 1 \implies (\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 1]] \;\&\; [\varphi_q(x) = 0 \implies (\forall y)(\exists z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 0]]] \gg$.

Define program list as follows. list runs a theorem prover for $\mathbf{T}$ and algorithmically lists, in some order, all the (p, q) such that $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$.

The global variable seq has value at any point a finite sequence of bits with known canonical index, and its initial value is a sequence whose first n values are all 1s, followed by a 0, and all other values are empty. We reconceptualize this initial value equivalently as a finite sequence followed by an infinite sequence of known-to-be-undefined values.

Define subroutine setup_seq (with global variable seq) as follows.

On input (p, q), setup_seq dovetails execution of p on all values x, until, if ever, a first x is found satisfying the following condition:

$\varphi_p(x)$ is defined, the position in seq corresponding to x is known undefined, and the positions in seq immediately before and immediately after that position are either known undefined or defined equal to 0.

If such an x is found, then define the position in seq corresponding to x to be $1 \dot{-} \varphi_p(x)$, and replace in seq any known undefineds immediately before or after that position with 0. After updating seq, setup_seq returns.

If there are no values for which the condition holds, setup_seq will run forever.

Define program define_real (with global variable seq) as follows.

On input x, define_real does the following:

Set i to 0.

For $x + 1$ iterations, do:

Run another step of list, and if, by the end of that step, it outputs

some (p, q), pass (p, q) to setup_seq.[7]

If seq(i) is known-undefined after that, define it equal to 0.

Increment i by 1.

Output the value of seq(x).

Claim 14. *For any (p, q) such that $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, it follows that W_p is infinite.*

Proof of Claim 14. Since $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, and $\mathbf{T}$ does not prove false things (of this sort), it follows that $(\forall x)[\varphi_q(x)\downarrow \leq 1]$, $\varphi_q(x) = 1 \implies (\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 1]$, and $\varphi_q(x) = 0 \implies (\forall y)(\exists z \mid y \leq z < y + x)[\varphi_p(z)\downarrow = 0]$. Thus, one of the following two cases about q holds:

Case one: $\varphi_q = \lambda x.(1)$. Then, φ_p does not have a longest run of 1s; the claim immediately follows for this case.

Case two: $\varphi_q = \lambda x.(1$ if $x \leq m; 0$ otherwise), for some m. Then, $\varphi_q(m+1)\downarrow = 0$; by that and the third conjunction in Translation 1 above, it follows that $(\forall y)(\exists z \mid y \leq z < y + m + 1)[\varphi_p(z)\downarrow = 0]$, from which the claim follows. □ Claim 14

Claim 15. *If* setup_seq *is passed a (p, q) such that $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, then it will terminate.*

Proof of Claim 15. Since $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, by Claim 14, it follows that W_p is infinite. As only finitely many elements of seq can be defined during a call to setup_seq, there will be *some* value for which the condition in setup_seq will be satisfied; therefore, setup_seq will terminate. □ Claim 15

Claim 16. *For any p, q, if $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, then passing (p, q) to* setup_seq *results in a* seq *such that the set of reals compatible with the defined values of* seq *does not include φ_p.*

---

[7] list can output either nothing or exactly one ordered pair on a given step.

Proof of Claim 16. By Claim 15, setup_seq will terminate. When it does, it will have found an empty location in the just prior seq corresponding to some value x such that it can set that position's value equal to $1 \dot{-} \varphi_p(x)$; therefore, the resultant seq will not be compatible with φ_p. □ Claim 16

Claim 17. *The program* define_real *does, in fact, define a real.*

Proof of Claim 17. By Claim 15 and the definition of list, all setup_seq calls will terminate, and, on any input x to define_real, list is only run for a finite number of steps. Thus, define_real will always output a value from seq. The loop in define_real includes a step that ensures that the element of seq output by define_real will be defined, and all times when elements of seq are assigned values, they are assigned a value of either 0 or 1. So, define_real computes a total $\{0,1\}$-valued function, a real. □ Claim 17

Claim 18. *For any p, q, if $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg_{\mathbf{1}}$, then $\varphi_p \neq \varphi_{\text{define\_real}}$.*

Proof of Claim 18. By Claim 17, $\varphi_{\text{define\_real}}$ defines a real, and thus, if φ_p does not define a real, they are not equal. By Claim 16 and the fact that define_real's output is always compatible with seq, if φ_p does define a real, the real it defines is not equal to $\varphi_{\text{define\_real}}$. Therefore, $\varphi_p \neq \varphi_{\text{define\_real}}$. □ Claim 18

Claim 19. $f_{\varphi_{\text{define\_real}}} = \lambda x\,.(1 \text{ if } x \leq n; 0 \text{ otherwise})$.

Proof of Claim 19. seq is initialized with a run of exactly n 1s followed by a 0. Whenever setup_seq is executed, it creates a run in seq of consecutive 1s of length at most 1, with 0s on each side. If a value of seq is ever defined otherwise, it is defined by define_real to be 0; therefore, $\varphi_{\text{define\_real}}$ contains a run of exactly n 1s and does not contain any longer runs of 1s - which is merely another way to state the claim. □ Claim 19

By Claims 17, 18, and 19, the statement of the theorem follows with $\mathbf{r} = \varphi_{\text{define\_real}}$. □ Theorem 8

Now we begin to examine in detail what happens with Theorems 7 and 8 above, when we employ the weaker Translation 2 instead of Translation 1. By essentially the same proof, Theorem 7 merely becomes the stronger

Theorem 9. *There is a computable real* $\mathbf{r}$ *such that, for any* p, q *which compute* $\mathbf{r}$ *and* $f_{\mathbf{r}}$*, respectively,* $\mathbf{T} \not\vdash \ll q$ *computes* $f_{\varphi_p} \gg_{\mathbf{2}}$*. Furthermore,* $f_{\mathbf{r}} = \lambda x.(1)$*.*

As for Theorem 8 above, as noted above, under Translation 2, instead of Translation 1, it becomes a constructive opposite. It is useful to rewrite Translation 2 as its simple equivalent as follows.

Translation 3. $\ll (\forall x)[\varphi_q(x)\!\downarrow \leq 1 \ \ \& \ \ [\varphi_q(x) = 1 \implies (\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z)\!\downarrow = 1]] \ \ \& \ \ [\varphi_q(x) = 0 \implies (\forall y)(\exists z \mid y \leq z < y+x)[\varphi_p(z)\!\uparrow \vee \varphi_p(z)\!\downarrow \neq 1]]] \gg$.

Of course, we'll continue to abbreviate this reworded, equivalent second translation as $\ll q$ computes $f_{\varphi_p} \gg_{\mathbf{2}}$.

Theorem 10. *There exist constructively computable functions* p *and* q *such that, for any program* r *and integer* n *for which* $f_{\varphi_r} = \lambda x.(1$ *if* $x \leq n; 0$ *otherwise*$)$*,* $\mathbf{PA} \vdash \ll q(r, n)$ *computes* $f_{\varphi_{p(r,n)}} \gg_{\mathbf{2}}$*, and* $\varphi_{p(r,n)} = \varphi_r$*.*

Proof of Theorem 10. By Kleene's S-m-n theorem [44], constructively provable in **PA**, there is a constructively computable function p such that, for each x, r, n,

$$\varphi_{p(r,n)}(x) = \begin{cases} \varphi_r(x), & \text{if } (\forall y \mid x \dot{-} n \leq y \leq x)(\exists z \mid y \leq z \leq y + n)[\varphi_r(z)\!\downarrow \neq 1 \ \vee \ \Phi_r(z) > \Phi_r(x)]; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{3.9}$$

By Kleene's S-m-n theorem [44], again constructively provable in **PA**, there is a constructively computable function q such that, for each x, r, n,

$$\varphi_{q(r,n)}(x) = \begin{cases} 1, & \text{if } x \leq n; \\ 0, & \text{otherwise.} \end{cases} \tag{3.10}$$

Claim 20. *For all* n*,* $\mathbf{PA} \vdash \ll q(r, n) = \lambda x.(1$ *if* $x \leq n; 0$ *otherwise*$) \gg_{\mathbf{2}}$*.*

Proof of Claim 20. Thanks to the constructivity, **PA** proves the claim.

□ Claim 20

Claim 21. *For all r, n such that $f_{\varphi_r} = \lambda x.(1$ if $x \leq n; 0$ otherwise$)$, $\varphi_{p(r,n)} = \varphi_r$.*

Proof of Claim 21. By assumption about r and n, for all x, because the longest run of 1s in φ_r is of length no more than n, it is the case that $(\forall y)(\exists z \mid y \leq z < y+n+1)[\varphi_r(z)\uparrow \vee \varphi_r(z)\downarrow \neq 1]$; by restricting the range of y and replacing $z < y+n+1$ with $z \leq y+n$, it follows that $(\forall y \mid x \dot{-} n \leq y \leq x)(\exists z \mid y \leq z \leq y+n)[\varphi_r(z)\downarrow \neq 1 \vee \varphi_r(z)\uparrow]$.

Furthermore, for each x, one of the following three cases holds:

Case one: $\varphi_r(x)\uparrow$. In this case, both clauses of (3.9) have the same result; $\varphi_{p(r,n)}(x)\uparrow$; thus, $\varphi_{p(r,n)}(x) = \varphi_r(x)$.

Case two: $\varphi_r(x)\downarrow \neq 1$. In this case, $(\forall y \mid x \dot{-} n \leq y \leq x)$, $z = x$ satisfies $(\exists z \mid y \leq z \leq y+n)[\varphi_r(z)\downarrow \neq 1]$; thus, the first clause of (3.9) holds. Therefore, $\varphi_{p(r,n)}(x) = \varphi_r(x)$ in this case.

Case three: $\varphi_r(x)\downarrow = 1$. Since, in this case, $\Phi_r(x)$ has a finite value, it follows from the last formula in the first paragraph of this proof, that $(\forall y \mid x \dot{-} n \leq y \leq x)(\exists z \mid y \leq z \leq y+n)[\varphi_r(z)\downarrow \neq 1 \vee \Phi_r(z) > \Phi_r(x)]$; thus, the first clause of (3.9) holds again. Therefore, $\varphi_{p(r,n)}(x) = \varphi_r(x)$ in this case as well.

The statement of the claim follows immediately. □ Claim 21

Claim 22. *For all r, n such that $f_{\varphi_r} = \lambda x.(1$ if $x \leq n; 0$ otherwise$)$, **PA** $\vdash \ll (\exists y)(\forall z \mid y \leq z < y+n)[\varphi_{p(r,n)}(z)\downarrow = 1] \gg$, i.e., **PA** proves that $\varphi_{p(r,n)}$ has a run of at least n 1s.*

Proof of Claim 22. By Claim 21 and the definition of f_{φ_r}, it follows that $\varphi_{p(r,n)}$ has a run of n consecutive 1s; from this, there exists *some* y, t such that $(\forall z \mid y \leq z < y+n)[\varphi_{p(r,n)}(z)\downarrow = 1$within t steps$]$. By Gödel's Lemma [21, 35] that computable (or recursive) relations are numeralwise, provably-representable in, e.g., **PA**, for *that* y, t, **PA** $\vdash \ll (\forall z \mid y \leq z < y+n)[\varphi_{p(r,n)}(z)\downarrow = 1$ within t steps$] \gg$. By two applications of

existential generalization in **PA**, **PA** $\vdash \ll (\exists y)(\forall z \mid y \leq z < y+n)[\varphi_{p(r,n)}(z)\downarrow = 1] \gg$; thus **PA** $\vdash \ll (\exists y)(\forall z \mid y \leq z < y+n)[\varphi_{p(r,n)}(z)\downarrow = 1] \gg$. □ Claim 22

Claim 23. *For each* r, n, **PA** $\vdash \ll (\forall y)(\exists z \mid y \leq z \leq y+n)[\varphi_{p(r,n)}(z)\uparrow \ \vee \ \varphi_{p(r,n)}(z)\downarrow \neq 1] \gg$, *i.e.,* **PA** *proves that* $\varphi_{p(r,n)}$ *does not have a run of more than* n 1*s*

Proof of Claim 23.

Fix r, n.

We first reason that $(\forall y)(\exists z \mid y \leq z \leq y+n)[\varphi_{p(r,n)}(z)\uparrow \ \vee \ \varphi_{p(r,n)}(z)\downarrow \neq 1]$.

Assume by way of contradiction that $(\exists y)(\forall z \mid y \leq z \leq y+n)[\varphi_{p(r,n)}(z)\downarrow = 1]$; fix some such y. Fix an x such that $y \leq x \leq y+n$ and $\Phi_r(x)$ maximal in that range.

From the fact that $\varphi_{p(r,n)}(x)\downarrow$, it follows that the first clause of (3.9) holds; i.e., $(\forall y \mid x \dot{-} n \leq y \leq x)(\exists z \mid y \leq z \leq y+n)[\varphi_r(z)\downarrow \neq 1 \ \vee \ \Phi_r(z) > \Phi_r(x)]$. By universal specification, it follows that for y as previously fixed, $(\exists z \mid y \leq z \leq y+n)[\varphi_r(z)\downarrow \neq 1 \ \vee \ \Phi_r(z) > \Phi_r(x)]$. Fix such a z.

Case one: $\varphi_r(z)\downarrow \neq 1$. This is a contradiction to our assumption for contradiction, in conjunction with the fixed y.

Case two: $\Phi_r(z) > \Phi_r(x)$. This is a contradiction to $\Phi_r(x)$ being maximal in the range.

Therefore, $\neg(\exists y)(\forall z \mid y \leq z \leq y+n)[\varphi_{p(r,n)}(z)\downarrow = 1]$, which is equivalent to $(\forall y)(\exists z \mid y \leq z \leq y+n)[\varphi_{p(r,n)}(z)\uparrow \ \vee \ \varphi_{p(r,n)}(z)\downarrow \neq 1]$. This completes our above announced first part.

Next we observe that the above reasoning can be carried out in **PA**, which is underpinned first order logic, and which, in turn, allows reasoning with non-constructive cases.

Hence, **PA** $\vdash \ll (\forall y)(\exists z \mid y \leq z \leq y+n)[\varphi_{p(r,n)}(z)\uparrow \ \vee \ \varphi_{p(r,n)}(z)\downarrow \neq 1] \gg$.

□ Claim 23

Claim 24. *For all* r, n *such that* $f_{\varphi_r} = \lambda x.(1$ *if* $x \leq n; 0$ *otherwise*), **PA** $\vdash \ll q(r,n)$ *computes* $f_{\varphi_{p(r,n)}} \gg_{\mathbf{2}}$

Proof of Claim 24. This follows from Claims 20, 22, and 23. □ Claim 24

The theorem follows immediately from Claims 24 and 21. □ Theorem 10

3.3.2 Results About $g_{\mathbf{r}}$s

Next we present our strong, promised theorem regarding $g_{\mathbf{r}}$s.

Theorem 11. *For* any *computably enumerable (c.e.) set A containing 0, there exists a linear-time computable real $\mathbf{r}$ such that $g_{\mathbf{r}}$ is the characteristic function of A.*[8]

Proof of Theorem 11. Since set A is c.e., there exists a φ^{TM}-program a which half-decides A, terminating after possibly arbitrarily much computation for inputs in A, and going undefined for inputs not in A.

We informally define φ^{TM}-program r, which computes the corresponding, desired real $\mathbf{r}$, as follows. It is to be understood, that this r makes *implicit* use of the efficient conditional control structure from Lemma 3 and Remark 1 above. In (3.11) just below the T predicate is the one defined in (3.5) above. For each $x \in \mathbb{N}$, let

$$\varphi_r^{\text{TM}}(x) = \begin{cases} 0, & \text{if } (x < 4 \vee \\ & T(a, \pi_1(\log\log(x)), \pi_2(\log\log(x))) = 0 \vee \\ & x = 2^{\log(x)} \vee \\ & x > \pi_1(\log\log(x)) + 2^{\log(x)}); \\ 1, & \text{otherwise.} \end{cases} \tag{3.11}$$

Claim 25. *φ^{TM}-program r runs in time linear in the length of its input.*

Proof of Claim 25. Clearly, comparing x to a constant can be done in constant time. Computing $\log(x)$ can be done in time linear in the length of x [47, Lemma

[8] For technical reasons, it is difficult, in the proof just below, to produce a real that does not contain two consecutive 0s. A real $\mathbf{r}$ *with* two consecutive 0s always satisfies $g_{\mathbf{r}}(0) = 1$.

3.2(k)], as can computing $2^{\log(x)}$.[9] As noted in Section 3.2 above, calculation of each of π_1 and π_2 take time linear in the length of their input. Then, from (3.5) and line following (3.5) (above), $T(a, \pi_1(\log\log(x)), \pi_2(\log\log(x)))$ can be computed in time linear in the length of $\log\log(x)$. Adding two numbers can be done in time linear in the length of the longer, and comparing the result to another number can once again be done in time linear in the length of the longer number [47, Lemma 3.2(f, g)]. By [47, Lemma 3.18], linear-time computable predicates are closed under Boolean operations. Thus, from r's implicit employment of cond from Lemma 3 above, and by Remark 1 above, r itself executes in time linear in the length of its input x. □ Claim 25

Claim 26. *φ^{TM}-program r defines a real.*

Proof of Claim 26. From Claim 25 it follows that r is total. Since r returns only values in the range $\{0, 1\}$, it follows that it defines a real. □ Claim 26

Claim 27. *For all x such that $\varphi_r^{\mathrm{TM}}(x) = 1$, it is the case that for all y such that $2^{\log(x)} < y \leq x$, $\varphi_r^{\mathrm{TM}}(y) = 1$.*

Proof of Claim 27. Let x be any value such that $\varphi_r^{\mathrm{TM}}(x) = 1$. Clearly, x is not a power of 2, $T(a, \pi_1(\log\log(x)), \pi_2(\log\log(x))) = 1$, and $x \leq \pi_1(\log\log(x)) + 2^{\log(x)}$. If $x - 1$ is not a power of 2, then it follows that $\log(x) = \log(x - 1)$. From these facts, it follows that $T(a, \pi_1(\log\log(x - 1)), \pi_2(\log\log(x - 1)) = 1$ and $x - 1 \leq \pi_1(\log\log(x - 1)) + 2^{\log(x-1)}$, and thus it follows that $\varphi_r^{\mathrm{TM}}(x - 1) = 1$, if and only if $x - 1$ is not a power of 2. By downwards induction on x, the claim follows. □ Claim 27

Claim 28. *For all x such that $\varphi_r^{\mathrm{TM}}(x) = 1$, it is the case that for all y such that $x \leq y \leq \pi_1(\log\log(x)) + 2^{\log(x)}$, $\varphi_r^{\mathrm{TM}}(y) = 1$, and also that $\varphi_r^{\mathrm{TM}}(1 + \pi_1(\log\log(x)) + 2^{\log(x)}) = 0$.*

[9] An algorithm for this latter, with I/O in binary number representation, is to scan x (as usual left to right), and, on the output tape *from right to left*, writing a 0 onto an output tape for each symbol in x, then replacing the last-written 0 with a 1 when the end of x is reached. The φ^{TM}-system is based on dyadic I/O [47], but, by [47, Lemma 3.2(b)] passing between binary and dyadic is linear-time computable.

Proof of Claim 28. Let x be any value such that $\varphi_r^{\mathrm{TM}}(x) = 1$. Clearly, x is not a power of 2, $T(a, \pi_1(\log\log(x)), \pi_2(\log\log(x))) = 1$, and $x \leq \pi_1(\log\log(x)) + 2^{\log(x)}$. If $x+1$ is a power of 2, then $\varphi_r^{\mathrm{TM}}(x+1) = 0$. If $x+1$ is not a power of 2, then it follows that $\log(x) = \log(x+1)$, from which it follows that $T(a, \pi_1(\log\log(x+1)), \pi_2(\log\log(x+1))) = 1$, and thus $\varphi_r^{\mathrm{TM}}(x+1) = 1$ if and only if $x+1$ is not a power of 2 and $x+1 \leq \pi_1(\log\log(x+1)) + 2^{\log(x+1)}$. By induction on x and the fact that $2^{\log(x)} + \pi_1(\log\log(x)) < 2^{\log(x)+1}$, the claim follows. □ Claim 28

Claim 29. *For all x such that $\varphi_r^{\mathrm{TM}}(x) = 1$, there is a run of exactly $\pi_1(\log\log(x))$ 1s in the real $\mathbf{r}$, and the position x is in one such run.*

Proof of Claim 29. From Claims 27 and 28, as well as the fact that, for all x, $\varphi_r^{\mathrm{TM}}(2^{\log(x)}) = 0$, it follows that, for all x such that $\varphi_r^{\mathrm{TM}}(x) = 1$, there is a run of 1s from $2^{\log(x)} + 1$ to $\pi_1(\log\log(x)) + 2^{\log(x)}$, bounded on both sides by 0s. The claim follows immediately by use of basic algebra and arithmetic. □ Claim 29

Claim 30. *For all $y > 0$ and for all t, if $\varphi_a^{\mathrm{TM}}(y)$ terminates within t steps, then $\varphi_r^{\mathrm{TM}}(2^{2^{<y,t>}} + 1) = 1$.*

Proof of Claim 30. The claim follows directly from the definitions of a and r. □ Claim 30

Claim 31. *$g_{\mathbf{r}}$ is the characteristic function of A.*

Proof of Claim 31. By Claim 26 and the definition of $g_{\mathbf{r}}$, it follows $g_{\mathbf{r}}$ is a well-defined total $\{0,1\}$-valued function. From Claims 29 and 30, as well as the fact that, for all y, for all but finitely many t, for some xs, $< y, t >= \log\log(x)$, it follows that, for all $y > 0$ such that $\varphi_a^{\mathrm{TM}}(y)\downarrow$, there is a run of exactly y 1s in the binary representation of r, and thus $g_{\mathbf{r}}(y) = 1$. Assume by way of contradiction that there is some $y > 0$ such that $\varphi_a^{\mathrm{TM}}(y)\uparrow$ but $g_{\mathbf{r}}(y) = 1$. Then, there must be a run of exactly y 1s in r. Let x be the least number such that position x is in such a run. Then, by Claim 29,

$\pi_1(\log\log(x)) = y$. From the definition of r, it follows that $T(a, y, \pi_2(\log\log(x))) = 1$, a contradiction to the assumption that $\varphi_a^{\mathrm{TM}}(y)\!\uparrow$. The previous shows that the claim holds for all inputs > 0. It is clear that $\varphi_r^{\mathrm{TM}}(0) = \varphi_r^{\mathrm{TM}}(1) = 0$, therefore $g_{\mathbf{r}}(0) = 1$, and thus, by the assumption that 0 is in A, the claim is proven. □ Claim 31

The theorem follows from Claims 25, 26, and 31. □ Theorem 11

Chapter 4

A NON-UNIFORMLY C-PRODUCTIVE SEQUENCE & NON-CONSTRUCTIVE DISJUNCTIONS

4.1 Motivation

Let φ be an *acceptable programming system/numbering* of the partial computable functions: $\mathbb{N} = \{0, 1, 2, \ldots\} \to \mathbb{N}$, where, for $p \in \mathbb{N}$, φ_p is the partial computable function computed by *program* p of the φ-system; such numberings are characterized as being intercompilable with naturally occurring general purpose programming formalisms such as a full Turing machine formalism; let $W_p = \text{domain}(\varphi_p)$ [43, 44, 33, 46]. W_p is, then, the computably enumerable (c.e.) set $\subseteq \mathbb{N}$ accepted by φ-program p. We let Φ be a corresponding Blum Complexity Measure [3] for the φ-system.

My advisor taught a recursion theorem proof by employing a pair of cases that, for *each* q, $\{x \mid \varphi_x = \varphi_q\}$ is *not* c.e. The disjunction of cases used was:

$$\text{domain}(\varphi_q) \text{ is } \infty \text{ vs. is finite.} \tag{4.1}$$

A student asked *why* the proof involved an analysis by cases. The answer given straightaway to the student was that his teacher didn't know how else to do it.

The present chapter provides, *among other things*, a *better answer*. Of course, by Rice's Theorem [44], the cases (4.1) above are *non*-constructive.[1]

The better answer is: *any* proof that, for *each* q, $\{x \mid \varphi_x = \varphi_q\}$ is *not* c.e. *provably must* involve *some* such *non-constructivity*.

We assume the reader is familiar with the definitions and names of the levels of the (classical) Arithmetical Hierarchy as in Rogers' book [44]. The classically valid

[1] More about constructivity below.

Law of Excluded Middle (LEM) is: either a formula or its negation holds. The idea of constructive mathematical proof, beginning with the intuitionist (constructivist) Brouwer [2], from about 1908 (see [7]), is that mathematical proofs at every layer and step should permit the explicit presentation/*computation* of *examples proved to exist*. For this chapter, we need this concept only for parts of mathematics expressible in the language of First Order Peano Arithmetic (**PA**) [44, 35], and we again assume reader familiarity. This "proof-mineability" places some limitations (compared to classical mathematics) on the logical operators $\vee$ and $\exists$ (think of $\vee$ as a finitary version of $\exists$). Classical mathematics has no such absolute constraints on existence proofs. A nice example contrasting non-constructive and constructive proofs is found in [2] where it is made clear that *un*restricted LEM is a source of *non*-constructivity: in constructively permissible proofs by cases, it must be decidable as to which case/disjunct holds.

Brouwer's student, Heyting, formulated a variant of first order logic [35] called *intuitionistic logic* which is just like first order logic except it is missing unrestricted LEM; it captures thereby Brouwer's constructivist ideas in the context of logic [26]. *Heyting Arithmetic* (abbreviated: **HA**) is just **PA** but underpinned instead by intuitionistic logic (see also [52, 53]). It captures Brouwer's ideas about the mathematics *expressible in it*.

There has been recent interest in adding some *arithmetically limited version* of LEM (and other principles) to **HA**. One, then, gets theories of *strictly* intermediate strength between **HA** and **PA** [38, 1, 25]. For example, Π^0_2-LEM is the set of all instances of LEM, where, the instances are all and only those of the form

$$(\forall u)(\exists v)R(u,v,w) \ \vee \ \neg(\forall u)(\exists v)R(u,v,w), \tag{4.2}$$

for some computable predicate R.[3]

[2] Technically, Intuitionism is a brand of Constructivism somewhat more subjective in focus than general constructivism. We will not and need not explore herein the subjectivism of Brouwer's Intuitionism.

[3] It turns out, *in the cases there are quantifiers in front of* R, we can take such Rs to be

If we replace in (4.1) above 'is finite' by 'is not infinite', then the result above can be easily shown to be provable in (**HA** + Π^0_2-LEM).[4]

If, instead, we replace in (4.1) above, 'is infinite' by 'is not finite', the result is in Σ^0_2-LEM. Since, from [1], (**HA** + Π^0_2-LEM) is *strictly* less non-constructive than (**HA** + Σ^0_2-LEM), we choose to employ the indicated replacement for (4.1) above which is in Π^0_2-LEM:

$$\text{domain}(\varphi_q) \text{ is } \infty \text{ vs. is not } \infty. \tag{4.3}$$

Note that, while one can put any formula of **PA** into a *classically* equivalent normal form variant with all its quantifiers in front and alternating [44, 35], this equivalence, in some cases, is *not* provable in **HA**. *However*, from [1], for $\mathcal{A}$, a standard level (such as Π^0_2) in the arithmetical hierarchy, (**HA** + $\mathcal{A}$-LEM) *is* strong enough to *prove* the equivalence of the normal form variant of formulas classically equivalent to formulas defining the members of $\mathcal{A}$.

4.2 Basic Definition & Relevant Theorem

Recall that the *completely productive* (abbr: *c-productive*) sets ($\subseteq \mathbb{N}$) [39, 37, 16, 44] are the *effectively non-c.e. sets*, i.e., the sets S such that

$$(\exists \text{ computable } f)(\forall y)[f(y) \in ((S - W_y) \cup (W_y - S))]. \tag{4.4}$$

Idea: in (4.4) just above, $f(y)$ is a *counterexample to* $S = W_y$.[5]

---

primitive recursive [44]. *Then*, for example, Π^0_2-LEM becomes a computably decidable set. Hence, the axioms of (**HA** + Π^0_2-LEM) are computably decidable — importantly, then, making proof-checking there algorithmic.

[4] See the proofs of Proposition 1 and 2 below in Section 4.3.1.

[5] Below we'll write $((S - W_y) \cup (W_y - S))$ as $(S \triangle W_y)$, the symmetric difference of S, W_y.

Definition 1. *A* set sequence $S_q, q \in \mathbb{N}$, *is* uniformly c-productive *iff there is a* computable f *so that, for* all q, y, $f(q, y)$ *is a counterexample to* $S_q = W_y$.[6]

Relevance: a completely *constructive* proof that *each* S_q is *not* c.e. would *entail the* S_q*s forming a uniformly c-productive sequence.*[7]

Let $E_q = \{x \mid \varphi_x = \varphi_q\}$. Then:

Theorem 12. *The* set sequence $E_q, q \in \mathbb{N}$, *is* not *uniformly c-productive.*

We'll include the following short, sweet proof of Theorem 12 just above. Theorem 12 is generalized by Corollary 4 below.

Proof of Theorem 12. Suppose for contradiction that f is a computable function so that, for *all* q, y, $f(q, y)$ is a counterexample to E_q (i.e., $\{x \mid \varphi_x = \varphi_q\}$) $= W_y$.

By the *Double Recursion Theorem* [44] there are programs q_0, y_0 each of which *creates a copy of itself and the other* (outside themselves[8]) and each *uses its copies together with an algorithm for* f *to compute* $f(q_0, y_0)$ with:

$$\varphi_{q_0} = \varphi_{f(q_0,y_0)} \ \& \tag{4.5}$$

$$W_{y_0} = \{f(q_0, y_0)\}. \tag{4.6}$$

From just above,

$$f(q_0, y_0) \in (\{x \mid \varphi_x = \varphi_{q_0}\} \cap W_{y_0}). \tag{4.7}$$

Clearly, then, $f(q_0, y_0)$ *fails* to be a counterexample to $E_{q_0} = W_{y_0}$, and we get a contradiction. □ Theorem 12

---

[6] These set sequences are, in motivation and mathematically, reasonably unrelated to Cleave's creative sequences [14].
This defined notion will be generalized a bit in a formal definition below (Definition 2 in Section 4.3).

[7] That's *entail* or *imply*, *not* hold-iff.

[8] There is no need for infinite regress.

In this chapter, as in others, we will use the linear-time computable and invertible pairing function $\langle \cdot, \cdot \rangle$ from the Royer-Case monograph [47]. This function maps all the pairs of elements of $\mathbb{N}$ 1-1, onto $\mathbb{N}$. We also employ this notation, based on iterating, $\langle \cdot, \cdot \rangle$, as in the Royer-Case monograph [47], to code also triples, quadruples, ... of elements of $\mathbb{N}$ 1-1, onto $\mathbb{N}$. Technically, each φ_p takes one argument $\in \mathbb{N}$. For some p and some $n > 1$, and some $x_1, \ldots, x_n$, we will sometimes write $\varphi_p(x_1, \ldots, x_n)$ as an abbreviation for $\varphi_p(\langle x_1, \ldots, x_n \rangle)$. We'll sometimes so abbreviate for other functions which technically take a single argument $\in \mathbb{N}$.

In the next paragraph we assume reader familiarity with employed notions and relevant characterizations thereof and treated in Rogers' book [44].

By *contrast*, $\{\langle q, x \rangle \mid \varphi_x = \varphi_q\}$ *is*, of course, c-productive, *Why? Insightful answer:* Let q_1 be such that, say, $\varphi_{q_1} = \lambda y.(1)$. Then, *easily* $\overline{K} \leq_1 \{x \mid \varphi_x = \varphi_{q_1}\} \leq_1 \{\langle q, x \rangle \mid \varphi_x = \varphi_q\}$ — and this entails [44] the c-productivity of $\{x \mid \varphi_x = \varphi_{q_1}\}$ as well as of $\{\langle q, x \rangle \mid \varphi_x = \varphi_q\}$. We see, then, that the c-productivity of $\{\langle q, x \rangle \mid \varphi_x = \varphi_q\}$ can be shown exercising only one q-value, q_1; whereas *uniform* c-productivity must exercise all q-values.

4.3 Characterizing the Index Set Cases

4.3.1 Uniform C-Productivity of S_q, $q \in M$

An *index set* [44] is a set of φ-programs M so that, for some class of (1-argument) partial computable functions $\mathfrak{S}$, $M = \{p \mid \varphi_p \in \mathfrak{S}\}$.

Example (complementary) index sets include those implicit in the non-constructive disjunction of cases mentioned early on above (in Section 4.1): $M_{\text{inf}} = \{q \mid \text{domain}(\varphi_q) \text{ } \infty\}$ vs. $M_{\text{fin}} = \{q \mid \text{domain}(\varphi_q) \text{ not } \infty\}$.

Next we give the promised generalization of the notion of uniformly c-productive. For computations $\downarrow$ means *converges* and $\uparrow$ means *diverges*, English terms used as in Rogers' book [44].

Definition 2. *For any* index set M, *we define the sequence of sets* S_q, $q \in M$, *to be* uniformly c-productive *iff,* for some partial computable η, *for* any $q \in M$ *&* any y, $\eta(q,y)\downarrow$ *to a counterexample to* $S_q = W_y$.

It can be seen from the proof of the Characterization Theorem (Theorem 13 in Section 4.3.2 below) that, for $S_q = E_q$, *when* η just above exists, it can be taken to be *total*.

Below we write δ for domain and ρ for range.

By a pair of Kleene Parametric Recursion Theorem (KPRT) [44] arguments, for each $M \in \{M_{\text{inf}}, M_{\text{fin}}\}$, the corresponding sequence E_q, $q \in M$, *is* uniformly c-productive:

Proposition 1. *The sequence* E_q, $q \in M_{\text{inf}}$, is *uniformly c-productive.*

Proof of Proposition 1. By KPRT, there is a computable function f st, for each q, y, z,

$$\varphi_{f(q,y)}(z) = \begin{cases} \uparrow, & \text{if } f(q,y) \in W_y \text{ in } \leq z \text{ steps;} \\ \varphi_q(z), & \text{otherwise.} \end{cases} \tag{4.8}$$

Suppose $q \in M_{\text{inf}}$.

Case one: $f(q,y) \in W_y$. Then by (4.8) just above, $\delta\varphi_{f(q,y)}$ not ∞, so, $\varphi_{f(q,y)} \neq \varphi_q$. Hence, $f(q,y) \notin E_q$.

Case two: $f(q,y) \notin W_y$. Then by (4.8) just above, $\varphi_{f(q,y)} = \varphi_q$. Hence, $f(q,y) \in E_q$.

Therefore, in any case, $f(q,y) \in (E_q \triangle W_y)$. □ Proposition 1

Proposition 2. *The sequence* E_q, $q \in M_{\text{fin}}$, is *uniformly c-productive.*

Proof of Proposition 2. By KPRT, there is a computable function f st, for each q, y, z,

$$\varphi_{f(q,y)}(z) = \begin{cases} 1, & \text{if } f(q,y) \in W_y \text{ in } \leq z \text{ steps;} \\ \varphi_q(z), & \text{otherwise.} \end{cases} \tag{4.9}$$

Suppose $q \in M_{\text{fin}}$.

Case one: $f(q,y) \in W_y$. Then by (4.9) just above, $\delta\varphi_{f(q,y)}$ is ∞, so, $\varphi_{f(q,y)} \neq \varphi_q$. Hence, $f(q,y) \notin E_q$.

Case two: $f(q,y) \notin W_y$. Then by (4.9) just above, $\varphi_{f(q,y)} = \varphi_q$. Hence, $f(q,y) \in E_q$.

Therefore, in any case, $f(q,y) \in (E_q \triangle W_W)$. □ PROPOSITION 1

These two just prior proofs each involve Σ^0_1-LEM, already strictly subsumed in Π^0_2-LEM [1]. Here is why. Each of these two proofs employ for its only non-constructivity, for some computable f, the *disjunction* of cases

$$f(q,y) \in W_y \ \vee \ \neg[f(q,y) \in W_y], \tag{4.10}$$

clearly in Σ^0_1-LEM.

This provides a now-known-to-be *necessarily non*-constructive proof that, for *each* $q \in \mathbb{N}$, $E_q = \{x \mid \varphi_x = \varphi_q\}$ is *not* c.e. It's non-constructivities (plural) can all be handled in (**HA** + Π^0_2-LEM).

4.3.2 The Characterization

There are divisions into non-constructive (top level) disjunctions *besides* the above example for proving that, for each $q \in \mathbb{N}$, $E_q = \{x \mid \varphi_x = \varphi_q\}$ is not c.e.

For example, the division into $\{q \mid \textit{domain}(\varphi_q) \textit{ is not computable}\}$ vs. its complement also works, *but* it's Π^0_3-LEM, *more non*-constructive than Π^0_2-LEM above [1].

Let F_x, $x \in \mathbb{N}$, be a *canonical indexing* [44, 33] of all/only the *finite functions*: $\mathbb{N} \to \mathbb{N}$. Below we identify partial functions with their graphs (as sets of pairs). We have for our characterization:

Theorem 13. *For any index set M and corresponding sequence of sets $E_q (= \{x \mid \varphi_x = \varphi_q\})$, $q \in M$, the sequence is uniformly c-productive iff*

$$(\exists \textit{ c.e. } A \subseteq \overline{M})(\forall x)(\exists y \in A)[\varphi_y \supseteq F_x]. \tag{4.11}$$

Without loss of generality*: in each direction y can be taken to be* algorithmic *in* x.

The disjoint index sets partitioning $\mathbb{N}$, $\{q \mid \text{domain}(\varphi_q) \neq \emptyset\}$ (c.e.) vs. its complement, $\{q \mid \text{domain}(\varphi_q) = \emptyset\}$, do *not* work — since, by our just above characterization and the Rice-Shapiro-Myhill-McNaughton Theorem [44],

Corollary 4. *For* any c.e. index set M, E_q, $q \in M$, *is* not *uniformly c-productive.*

Since $\mathbb{N}$ is trivially a c.e. index set, Theorem 12 above in Section 4.1 follows from Corollary 4 just above. Since [44] the c.e. sets are the Σ^0_1 sets and their complements are the Π^0_1 sets, the problem of proving, for each $q \in \mathbb{N}$, E_q is not c.e. is upper-bounded (can be done) with (**HA** + Π^0_2-LEM) (by Propositions 1 and 2 in Section 4.3.1 above) and lower-bounded (can't be done) with (**HA** + Σ^0_1-LEM)[9] — the latter at least for employing a division into cases involving *index sets*. We conjecture that (**HA** + Δ^0_2-LEM) is a *maximal* lower-bound (cannot be done) — at least for employing a division into cases involving *index sets*. In general, we don't know about divisions into cases *not* involving index sets and we don't know whether (**HA** + Π^0_2-LEM) is a *minimum* required with respect to the Arithmetically-Limited-LEMs from Akama, Berardi, Hayashi, and Kohlenbach's paper [1].

Proof of the Characterization Theorem 13. Fix index set M.

Claim 32. *If* $E_q, q \in M$, *is uniformly c-productive, then*

$$(\exists \text{ } C.E. \text{ } A \subseteq \overline{M})(\forall x)(\exists y \in A)[\varphi_y \supseteq F_x]. \tag{4.12}$$

Proof of Claim 32. Let partial computable η be a witness to the uniform c-productivity of $E_q, q \in M$; that is, for each q, y st $q \in M$, $\eta(q, y)\downarrow \in (E_q \triangle W_y)$.

By Kleene's S-m-n theorem [44] there is a computable function t st, for each x,

$$W_{t(x)} = \{z \mid \varphi_z \supseteq F_x\}. \tag{4.13}$$

[9] From [1], (**HA** + Π^0_1-LEM) is strictly less non-constructive than (**HA** + Σ^0_1-LEM).

By Kleene's Parametric Recursion Theorem [44] there is a computable function s st, for each x, z,

$$\varphi_{s(x)}(z) = \begin{cases} F_x(z), & \text{if } z \in \delta F_x; \\ \varphi_{\eta(s(x),t(x))}(z), & \text{otherwise.} \end{cases} \tag{4.14}$$

Subclaim 1. $(\forall x)[s(x) \in \overline{M}]$

Proof of Subclaim 1. Suppose by way of contradiction there exists *some* x such that that $s(x) \in M$. Fix that x. By the assumption that $s(x) \in M$,

$$\eta(s(x), t(x))\downarrow \in (E_{s(x)} \triangle W_{t(x)}). \tag{4.15}$$

Case one: $\eta(s(x), t(x)) \in W_{t(x)}$. Then, for all z, one of two subcases holds.

Subcase one: $z \in \delta F_x$. In this subcase, as $\varphi_{\eta(s(x),t(x))}$ extends F_x, $\varphi_{\eta(s(x),t(x))}(z) = F_x(z)$, and, by the first clause of (4.14), $\varphi_{s(x)}(z) = F_x(z)$; therefore, $\varphi_{s(x)}(z) = \varphi_{\eta(s(x),t(x))}(z)$.

Subcase two: $z \notin \delta F_x$. In this subcase, by the second clause of (4.14), $\varphi_{s(x)}(z) = \varphi_{\eta(s(x),t(x))}(z)$.

Thus, $(\forall z)[\varphi_{s(x)}(z) = \varphi_{\eta(s(x),t(x))}(z)]$. This means $\eta(s(x), t(x)) \in E_{s(x)} \wedge \eta(s(x), t(x)) \in W_{t(x)}$; a contradiction to (4.15).

Case two: $\eta(s(x), t(x)) \notin W_{t(x)}$. Then there exists *some* $z \in \delta F_x$ such that $\varphi_{\eta(s(x),t(x))}(z) \neq F_x(z)$, thus, by clause one of (4.14), $\varphi_{\eta(s(x),t(x))} \neq \varphi_{s(x)}$. This means $\eta(s(x), t(x)) \notin E_{s(x)} \wedge \eta(s(x), t(x)) \notin W_{t(x)}$; a contradiction to (4.15).

Both cases lead to a contradiction; the subclaim immediately follows.

□ Subclaim 1

Let $A = \rho s$. The claim then follows from Subclaim 1, the fact that A is the range of a computable function, and (4.14). □ Claim 32

Claim 33. *If there exists a c.e. set A that is a subset of $\overline{M}$ such that, for any finite function F, there is a program in A which computes a function that extends F, then the sequence of sets $E_q, q \in M$, is uniformly c-productive.*

Proof of Claim 33. Assume the existence of such an A.

> Let s be a φ-version of the following program on inputs q, y, e.
>
> Determine how many steps it takes to get an output from program y on input e; if it never terminates, never terminate from this point.
>
> Compute the canonical index of the finite function F whose graph is the set of $(z, \varphi_q(z))$ such that $\Phi_y(e) > z$ and $\Phi_y(e) > \Phi_q(z)$.
>
> Dovetail an algorithmic enumeration of A, until, if ever, an element a is found such that φ_a extends F. If such an a is found, output that a. Otherwise, $\uparrow$.

The φ-program s just below in (4.16) is from the second line in this proof of Claim 33. By Kleene's Parametric Recursion Theorem [44] there is a computable function f st, for each q, y, z,

$$\varphi_{f(q,y)}(z) = \begin{cases} \varphi_q(z), & \text{if } \Phi_y(f(q,y)) > z \\ & \quad \wedge\ \Phi_y(f(q,y)) > \Phi_q(z); \\ \varphi_{\varphi_s(q,y,f(q,y))}(z), & \text{if } \varphi_y(f(q,y))\downarrow \\ & \quad \wedge\ \varphi_s(q,y,f(q,y))\downarrow \\ & \quad \wedge\ [\Phi_y(f(q,y)) \leq z \\ & \qquad \vee\ \Phi_y(f(q,y)) \leq \Phi_q(z)]; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{4.16}$$

Subclaim 2. *f is a witness to the uniform c-productivity of $E_q, q \in M$.*

Proof of Subclaim 2. Fix q and y.

Case one: $f(q,y) \in W_y$. Thus, $\Phi_y(f(q,y))$ converges. Because of this, we can compute the canonical index of the finite function F whose graph is $\{(z, \varphi_q(z)) \mid \Phi_y(f(q,y)) > z \ \wedge\ \Phi_y(f(q,y)) > \Phi_q(z)\}$; from that and the assumption about A, it follows that $\varphi_s(q,y,f(q,y))\downarrow$ to an $a \in A$ such that φ_a extends F. By careful examination of the clauses in (4.16) and the behavior of program s on $q, y, f(q,y)$, it follows that $\varphi_{f(q,y)} = \varphi_a$. By assumption about A, $a \in \overline{M}$, from which it follows that $f(q,y) \in \overline{M}$, thus $f(q,y) \notin E_q$.

Case two: $f(q,y) \notin W_y$. Then, for all z such that $\varphi_q(z)\downarrow$, $\varphi_{f(q,y)}(z) = \varphi_q(z)$ by the first clause of (4.16), and for all z such that $\varphi_q(z)\uparrow$, $\varphi_{f(q,y)}(z)\uparrow$ by the third clause of (4.16); thus, $\varphi_{f(q,y)} = \varphi_q$. Therefore, $f(q,y) \in E_q$.

Thus, by Case one and Case two, $(\forall q, y)[f(q,y) \in E_q \mathbin{\triangle} W_y]$: f is a witness to the uniform c-productivity of the sequence $E_q, q \in M$. □ Subclaim 2

The claim follows from Subclaim 2 and the computability of f. □ Claim 33

.

The theorem follows immediately from Claims 32 and 33. □ Theorem 13

4.3.3 Another Corollary of the Characterization

Above, we've looked at proving *each* E_q is *not* c.e., with the proof's top level *non*-constructivity confined to disjunctions with *two* disjuncts as to which of *two* partitioning index sets contains q. How about such disjunctions but with numbers of such irreducible disjuncts *more than* 2? Thanks to our characterization:

Corollary 5. *For each $n \geq 2$, there are n pairwise disjoint, non-trivial index sets $M_0, \ldots, M_{n-1}$ unioning to $\mathbb{N}$ such that:*

- *For each $i < n$, the sequence $E_q = \{x \mid \varphi_x = \varphi_q\}$, $q \in M_i$,* is *uniformly c-productive,* but
- *For each $i, j < n$ such that $i \neq j$, the sequence E_q, $q \in (M_i \cup M_j)$, is* not *uniformly c-productive.*

A simple extension of the just above Corollary's proof (just below) yields the $n = \omega$, infinitary, irreducible disjunctions case too. We omit the details of that extension.

Proof of Corollary 5. Assume $n \geq 2$.

Let

$$O = \{\langle i, j\rangle \mid i \neq j \ \wedge \ i < n \ \wedge \ j < n\}. \tag{4.17}$$

Let

$$O_0 = \{q \mid \varphi_q(0)\uparrow \ \vee \ [\varphi_q(0)\downarrow \notin O \ \wedge \ W_q \text{ infinite}]\}. \tag{4.18}$$

Let

$$O_1 = \{q \mid \varphi_q(0)\downarrow \notin O \ \wedge \ W_q \text{ finite}\}. \tag{4.19}$$

For each $i < n$, let

$$S_i = \begin{cases} \{q \mid [\varphi_q(0)\downarrow \in \{\langle x, i\rangle \mid \langle x, i\rangle \in O\} \ \wedge \ W_q \text{ finite}] \vee \\ \qquad [\varphi_q(0)\downarrow \in \{\langle i, x\rangle \mid \langle i, x\rangle \in O\} \ \wedge \ W_q \text{ infinite}]\}. \end{cases} \tag{4.20}$$

Let

$$M_0 = (S_0 \cup O_0). \tag{4.21}$$

Let

$$M_1 = (S_1 \cup O_1). \tag{4.22}$$

For each $i \geq 2$ st $i < n$, let

$$M_i = S_i. \tag{4.23}$$

Claim 1. *The union of all M_i is $\mathbb{N}$.*

Proof of Claim 1. Suppose by way of contradiction there exists some q such that q is not in the union of all the M_i.

Case one: $\varphi_q(0)\uparrow$. Then, $q \in M_0$ by (4.18); a contradiction.

Case two: $\varphi_q(0)\downarrow \notin O$, and W_q is infinite. Then, $q \in M_0$ by (4.18); a contradiction.

Case three: $\varphi_q(0)\downarrow \notin O$, and W_q finite. Then, $q \in M_1$ by (4.19); a contradiction.

Case four: $\varphi_q(0)\downarrow \in O$. Then, by (4.17), there exists some i, j such that $i \neq j, i < n, j < n, \langle i, j\rangle \in O$, and $\varphi_q(0) = \langle i, j\rangle$.

Subcase one: W_q finite. Then, by (4.20), q is in M_j; a contradiction.

Subcase two: W_q is infinite. Then, by (4.20), q is in M_i; a contradiction.

Hence, in any case we have a contradiction. Therefore, q is in the union of all M_i, and the claim follows. □ Claim 1

Claim 2. *For any $i < n$, and any $j < n$, st $i \neq j$, M_i and M_j are disjoint.*

Proof of Claim 2. Suppose by way of contradiction there exists some $i, j < n$, and there exists q such that $i \neq j$ and $q \in (M_i \cap M_j)$.

Case one: $\varphi_q(0)\uparrow$. Then, $q \in M_0$ by (4.18). By (4.19) and (4.20), q is not in any other set; a contradiction.

Case two: $\varphi_q(0)\downarrow \notin O$, and W_q is infinite. Then, $q \in M_0$ by (4.18). By (4.19) and (4.20), q is not in any other set; a contradiction.

Case three: $\varphi_q(0)\downarrow \notin O$, and W_q finite. Then, $q \in M_1$ by (4.19); By (4.18) and (4.20), q is not in any other set; a contradiction.

Case four: $\varphi_q(0)\downarrow \in O$, and W_q is infinite. Then, by (4.17), $\varphi_q(0) = \langle i', j' \rangle, i' < n, j' < n$, and $i' \neq j'$. q is not in O_0 nor O_1 by (4.18) and (4.19). By (4.20), q is in $S_{i'}$, and not in $S_{j'}$. Then, by (4.21), (4.22), and (4.23), q is in exactly and only $M_{i'}$, a contradiction.

Case five: $\varphi_q(0)\downarrow \in O$, and W_q finite. Then, by (4.17), $\varphi_q(0) = \langle i', j' \rangle, i' < n, j' < n$, and $i' \neq j'$. q is not in O_0 nor O_1 by (4.18) and (4.19). By (4.20), q is in $S_{j'}$, and not in $S_{i'}$. Then, by (4.21), (4.22), and (4.23), q is in exactly and only $M_{j'}$, a contradiction.

All of the cases lead to a contradiction. Therefore, there are no such i, j, q; the claim immediately follows. □ Claim 2

Claim 3. *For any $i, j < n$ st $i \neq j$, the sequence $E_q, q \in (M_i \cup M_j)$ is* not *uniformly c-productive.*

Proof of Claim 3. By Theorem 13, $(M_i \cup M_j)$ is uniformly c-productive if and only if $\overline{M_i \cup M_j}$ contains a c.e. set A st A has a program for some extension of *each* finite function. Suppose for contradiction the claim fails; hence, there is such a c.e. A.

By (4.20), all finite function extensions of $(0, \langle j, i \rangle)$ have all their programs in M_i, and all infinite partial computable extensions of $(0, \langle j, i \rangle)$ have all their programs in M_j; thus, all partial computable extensions of $(0, \langle j, i \rangle)$ have all their programs in $(M_i \cup M_j)$; hence, the complement of that set (which includes A) has no programs for such extensions, a contradiction. □ Claim 3

Claim 4. *For any $i < n$, the set M_i is uniformly c-productive.*

Proof of Claim 4. By Kleene's S-m-n Theorem [44], there exist computable *functions* q_1, q_2, q_3 such that, for each x, z,

$$\varphi_{q_1(x)}(z) = \begin{cases} F_x(z), & \text{if } z \in \delta F_x; \\ 0, & \text{otherwise.} \end{cases} \tag{4.24}$$

$$\varphi_{q_2(x)}(z) = F_x(z). \tag{4.25}$$

$$\varphi_{q_3(x)}(z) = \begin{cases} \langle i, i \rangle, & \text{if } z = 0; \\ F_x(z), & \text{otherwise.} \end{cases} \tag{4.26}$$

Case one: $i \geq 1$ st $i < n$. Let p be the code number of a program as follows:

$$\varphi_p(x) = \begin{cases} q_1(x), & \text{if } (\exists y)[F_x(0)\downarrow = \langle y, i \rangle \ \wedge \ F_x(0) \in O]; \\ q_2(x), & \text{otherwise.} \end{cases} \tag{4.27}$$

Subclaim 3. *φ_p is total.*

Proof of Subclaim 3. Determining if $F_x(0)$ is defined is computable, treating it as an ordered pair and extracting the second component to compare to i is also computable, and determining if something is a member of O is computable in a similar fashion. Thus, the subclaim follows. □ Subclaim 3

Subclaim 4. *The range of φ_p is a subset of $\overline{M_i}$*

Proof of Subclaim 4. For all x, one of the following two cases holds:

Case (a): $(\exists y)[F_x(0)\downarrow = \langle y, i \rangle \ \wedge \ F_x(0) \in O]$. Then, by (4.27) $(\exists y)[\varphi_{\varphi_p(x)}(0)\downarrow = \langle y, i \rangle \ \wedge \ \langle y, i \rangle \in O]$, and $W_{\varphi_p(x)}$ is infinite. Therefore, $\varphi_p(x)$ is *not* in M_i.

Case (b): Otherwise. Thus, $\varphi_{\varphi_p(x)} = F_x$, and therefore $W_{\varphi_p(x)}$ is finite. By (4.20), all programs which are in M_i and compute finite functions are such that the

value of their computed function on input 0 is $\langle y, i\rangle$ for some $y < n$. If F_x were such a function, Case (a) would hold instead. Therefore, $\varphi_p(x)$ is *not* in M_i.

□ Subclaim 4

Subclaim 5. *For all x, there is a program in the range of φ_p which computes a partial function that extends F_x.*

Proof of Subclaim 5. This follows directly from (4.27) and Subclaim 3.

□ Subclaim 5

By Subclaims 3, 4, and 5, the range of φ_p is a c.e. subset of $\overline{M_i}$ such that for any finite function F_x, there is a program in that c.e. subset which extends F_x. By Theorem 13, M_i is uniformly c-productive in this case.

Case two: $i = 1$. Let p be the code number of a program as follows:

$$\varphi_p(x) = \begin{cases} q_1(x), & \text{if } (\exists y)[F_x(0)\!\downarrow = \langle y, i\rangle \ \wedge \ F_x(0) \in O] \ \vee \ F_x(0)\!\downarrow \notin O; \\ q_2(x), & \text{otherwise.} \end{cases} \tag{4.28}$$

The proof of Subclaim 3 is applicable to an identical subclaim *mutatis mutandis*. The proof of Subclaim 4 works if Case (a) is replaced with two subcases; Subcase i is identical to Case (a), while Subcase ii is the case where $F_x(0)\!\downarrow \notin O$; in that instance, $\varphi_p(x)$ is in M_0 by (4.18), and thus by Claim 2 is not in M_1. The proof of Subclaim 5 holds *mutatis mutandis*. Thus, just as per Case one, M_1 is uniformly c-productive.

Case three: $i = 0$. Let p be the code number of a program as follows:

$$\varphi_p(x) = \begin{cases} q_3(x), & \text{if } 0 \notin F_x; \\ q_1(x), & \text{if } (\exists y)[F_x(0)\!\downarrow = \langle y, i\rangle \ \wedge \ F_x(0) \in O]; \\ q_2(x), & \text{otherwise.} \end{cases} \tag{4.29}$$

The proof of Subclaim 3 is applicable to an identical subclaim *mutatis mutandis*. The proof of Subclaim 4 works if Case (b) is renamed to Case (c) and a new Case (b) is inserted between Case (a) and Case (c); the new Case (b) is: 0 is not in F_x. In Case (b),

φ_q is finite and $\varphi_q(0)\downarrow \notin O$; thus, by (4.19), q in M_1. Thus, by Claim 2, q is not in M_0. The proof of Subclaim 5 holds *mutatis mutandis*. Thus, just as per Cases one and two, M_0 is uniformly c-productive.

As M_i is uniformly c-productive for all $i < n$, the claim holds. □ Claim 4

Claims 1, 2, 3, 4 together provide exactly the statement of the corollary.

□ Corollary 5

4.4 Further Examples and Future Work

We see that $\overline{E_q} = \{x \mid \varphi_x \neq \varphi_q\}$. Clearly, for *each* q such that $\text{domain}(\varphi_q) = \emptyset$, $\overline{E_q}$ *is c.e.*

Let $M_{\text{ne}} = \{q \mid \text{domain}(\varphi_q) \neq \emptyset\}$.

Then, by an omitted Σ^0_1-LEM KPRT argument, the sequence, $\overline{E_q}$, $q \in M_{\text{ne}}$, *is* uniformly c-productive — as witnessed by a *total* computable function. Again, our division into cases is of the form (4.10), and it is open whether this is minimally non-constructive.

The sequence $\{x \mid W_x = W_q\}$, $q \in \mathbb{N}$, and its complementary sequence satisfy results like the E_qs and $\overline{E_q}$s *mutatis mutandis*, e.g., for the Characterization Theorem, F_x is replaced by Rogers' [44] D_x, the finite *set* with canonical index x. We omit the straightforward details.

An aside: my advisor initially suspected that each index set is either c.e. or c-productive, but couldn't prove it; we can now prove that *some* index set S (perhaps a possible S_q) is *neither* c.e. *nor* c-productive! We prove this in the next subsection (Section 4.4.1). Curiously, then, by *a* proof of Rice's Theorem [44], $\overline{S}$ *must be* $\geq_1 \overline{K}$, hence [44] *c-productive*.

4.4.1 An Index Set Neither C.E. Nor C-Productive

Theorem 14. *Given any set S, there exists an index set S' such that S' is c.e if and only if S is c.e., and, if S' is c-productive, then so is S.*

Proof of Theorem 14. Let $S' = \{p \mid \varphi_p(0)\downarrow \in S\}$.

Clearly S' is an index set.

Clearly too, if S is c.e., then S' is c.e..

Claim 34. *If S' is c.e., then S is c.e..*

Proof of Claim 34. Assume S' is c.e.. Then there exists s' such that $W_{s'} = S'$. Let s be a program such that $W_s = \{x \mid (\exists p \in W_{s'})[\varphi_p(0) = x]\}$. Clearly, $S = W_s$, therefore S is c.e. □ Claim 34

Claim 35. *If S' is c-productive, then S is c-productive.*

Proof of Claim 35. Assume S' is c-productive as witnessed by computable function f'.

Let p be defined such that $\varphi_p(x, y)$ is the yth element (if any) in a double-dovetailed enumeration of programs which on input 0 output a value in W_x, and undefined if no such element exists.

By S-m-n there is a computable function $\lambda x.(p_x)$ st, for each x, y, $\varphi_{p_x}(y) = \varphi_p(x, y)$.

Let θ be the partial computable function st, for each x, $\theta(x) = \varphi_{f'(p_x)}(0)$.

Suppose by way of contradiction that θ is not total. Let x be least st $\theta(x)\uparrow$. Then, by the assumption that f' witnesses the c-productivity of S', there is some $w = f'(p_x)$ st $\varphi_w(0)\uparrow$. Then, $w \notin W_{p_x}$. Furthermore, by the definition of S', $w \notin S'$. But this means that f' does not witness the c-productivity of S' on input p_x, a contradiction. Therefore, $f = \theta$ is total. So, f is computable st, for each x, $f(x) = \varphi_{f'(p_x)}(0)$.

By the assumption that f' witnesses the c-productivity of S', it follows that, for each x, exactly one of the following two cases holds:

Case one: $f'(p_x) \in (W_{p_x} - S')$. By the construction of p_x and the definition of S', this implies that $\varphi_{f'(p_x)}(0) \in (W_x - S)$. Thus, $f(x) \in (W_x - S)$.

Case two: $f'(p_x) \in (S' - W_{p_x})$. Thus, $f(x) \in (S - W_x)$.

Therefore, by Cases one and two, f is a witness to the c-productivity of S.

□ Claim 35

The theorem follows from the claims. □ Theorem 14

The desired result for the present subsection follows.

Corollary 6. *There exists an index set which is neither c.e. nor c-productive.*

Proof of Corollary 6. Immune sets are neither c.e. nor c-productive, thus, by Theorem 14 above, there exists an index set S', corresponding to an immune set S, which is neither c.e. nor c-productive. □ Corollary 6

4.4.2 Some Subrecursive Examples

In this section we consider a wide variety of "natural" *sub*recursive programming systems ψ, including for such subrecursive classes as: for $k > 0$, the functions computable in time bounded by a k-degree polynomial; the polynomial time computable functions; the elementary recursive functions; and other levels of the Meyer-Ritchie [36] loop hierarchy.

Let $C_q = \{x \mid \psi_x = \psi_q\}$. Trivially, each $\overline{C_q}$ *is* c.e.

In this section our main goal is to prove Theorem 16 below (in Section 4.4.2.2 below) that the sequence $C_q, q \in \mathbb{N}$, *is* uniformly c-productive, and, interestingly, as witnessed by a two-argument function f *computable in linear-time in the lengths of its inputs*. We had had an f linear-time in the length of its second input, and Jim Royer supplied a suggestion which enabled getting f linear-time in the lengths of both its inputs.

Before we prove that theorem, we describe a bit informally (but with references to more detail) nice clocked programming systems for subrecursive classes such as the above. We spell out two Assumptions as to for which subrecursive classes and corresponding programming systems we can prove this theorem. We prove (mostly with a few citations and hints as to underlying ideas from the Royer-Case monograph

[47]), *as an example*, Theorem 15 (in Section 4.4.2.2) below that, for $k > 0$, the nice clocked systems for the class of functions computable in k-degree polynomial time *do* satisfy our two Assumptions.

Let θ^k be such a programming system. For this system, let f be the *associated* function mentioned above. For each *fixed* q, y, $f(q, y)$ *is* a θ^k-program, which, *then*, on any input z, runs within time $\mathcal{O}(|z|^k)$. For *these particular* θ^k*-systems*, the *lower*-bound run time cost of running corresponding *universal (simulator)* programs is known to be very high. We examine the best known (exponential) *upper*-bound cost of simulating, for the associated f, the run time cost, as a function of *all of* q, y, z, of computing $\theta^k_{f(q,y)}(z)$. Lastly, Theorem 17 (in Section 4.4.2.2) below lays out an explicit $\mathcal{O}$-formula for this exponential cost.

4.4.2.1 Preliminaries

Residually unexplained notation or terminology is from Rogers' book and/or the Royer-Case monograph [44, 47].

As above we choose our multi-argument "pairing" $\langle \cdot, \ldots, \cdot \rangle$ (and unpairing) function(s) to be computable in linear-time in the lengths of arguments on multi-tape Turing Machines.[10] For one-argument functions α we write $\alpha(z_1, \ldots, z_m)$ to mean $\alpha(\langle z_1, \ldots, z_m \rangle)$ [47].

For *this* section we suppose $\mathfrak{S}$ is a c.e. class of one-argument computable functions [44] and that ψ is a "natural" *sub*recursive programming system (numbering) for this $\mathfrak{S}$, with ψ's universal function $\lambda\langle q, z\rangle.(\psi_q(z))$ being (at least) computable[11],

---

[10] Of course, as in Rogers' book [44], for each $m > 1$, $\lambda y_1, \ldots, y_m.(\langle y_1, \ldots, y_m \rangle) : \mathbb{N}^m \to \mathbb{N}$ is 1-1 and onto. For this chapter, we won't need names for the unpairing functions.

[11] Of course, since all the ψ_qs are total, $\lambda\langle q, z\rangle.(\psi_q(z))$ is not in $\mathfrak{S}$.
In [47] there are many calculations of complexity upper-bounds (and some lower-bounds) for universal functions. In the proof of the last theorem below in Section 4.4.2.2 (Theorem 17), we employ one such upper-bound.

and *where* $\mathfrak{S}$ and ψ satisfy the "closure" properties Assumptions 1 and 2 spelled out further below.

First it is pedagogically useful to provide typical examples of *such* $\mathfrak{S}$ and ψ.

We let φ^{TM} be the acceptable programming system based on *efficiently numbered*, deterministic, base 2 I/O, multi-tape Turing machines (TMs) — all from [47, Chapter 3 & Errata]. Φ^{TM} is the corresponding TM step-counting Blum Complexity Measure [3].

Suppose $k > 0$.

We let $\mathcal{P}\text{time}_k$ [47, Definition 3.3] be the class of functions: $\mathbb{N} \to \mathbb{N}$ each computed by *some* φ^{TM}-program which, on each input z, has Φ^{TM}-complexity in $\mathcal{O}(|z|^k)$.

$\mathcal{L}\text{time}$ denotes $\mathcal{P}\text{time}_1$, the class of linear-time computable functions. Of course $\mathcal{P}\text{time} = \bigcup_{k>0} \mathcal{P}\text{time}_k$ is the class of polynomial time computable functions.

We let θ^k be a very natural, so-called $\mathcal{L}$time-effective *clocked* programming system (numbering) with respect to $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$ and *for* the class $\mathcal{P}\text{time}_k$ (see especially [47, Sections 3.2.2 & 4.2 & Chapter 6]). Associated with θ^k are functions trans and positive-valued bound, each nicely $\in \mathcal{L}\text{time}$. Each θ^k-program q *directly/efficiently codes both* a φ^{TM}-program p *and* a positive coefficient a for the run time (upper) bound $(a|z|)^k$.[12]

Here is how to think about the running of θ^k-program q on input z. The φ^{TM}-program trans(q) does the work: trans(q) on z computes $(a|z|)^k$ and directly runs p on z for no more than $(a|z|)^k$ steps. If p halts by then, trans(q)'s (as well as q's) output is p's output; else, trans(q) outputs some default value (which is, then, also q's output value). The working of trans(q), of course, may take a bit more than time on z than

[12] As in [47] we are using $(a|z|)^k$ instead of the more natural $a(|z|)^k$. Curiously, the latter does not provide $\mathcal{L}$time-effective full composition for all of $\mathcal{P}$time, but the former (with an extra $+1$) does. $\mathcal{L}$time-effective inner and outer composition with $\mathcal{L}$time *for* $\mathcal{P}\text{time}_k$ *does* work for the latter, but with somewhat larger coefficients for the compositions and a little smaller exponent for [47, Theorem 6.4]. We employ this latter theorem in the proof of our Theorem 17 below, so we'll stick with $(a|z|)^k$ for $\mathcal{P}\text{time}_k$ for our Section 4.4.2.2 below.

$(a|z|)^k$, e.g., because it spends time on computing this run time bound coded in q; however, in any case, trans(q) on z runs within time $(\text{bound}(q)|z|)^k$. In symbols:

$$(\forall q, z)[\theta^k_q(z) = \varphi^{\text{TM}}_{\text{trans}(q)}(z) \ \ \& \ \ \Phi^{\text{TM}}_{\text{trans}(q)}(z) \leq (\text{bound}(q)|z|)^k]. \tag{4.30}$$

Both general and specific technical details about various clocked systems are in [47, Section 3.2.2 & Chapters 4–6].

Technical details for many $\mathcal{L}$time-effective clocked systems, including *also* for the polynomial time computable functions, the elementary recursive functions, other levels of the Meyer-Ritchie [36] loop hierarchy, ..., can be found in [47, Section 4.2 & Chapters 5 & 6].

As we will see in Theorem 15 further below, $(\mathfrak{S}, \psi) = (\mathcal{P}\text{time}_k, \theta^k)$ satisfies our above mentioned Assumptions 1 and 2 which are presented shortly below. Actually, the other complexity classes and corresponding clocked systems mentioned at the end of the just prior paragraph also satisfy Assumptions 1 and 2, but we omit herein verification details regarding that.

Assumption 1. $\mathfrak{S}$ *contains the identity function and is closed under inner and outer composition with* $\mathcal{L}$time*; hence, in particular,* $\mathfrak{S}$ *contains* $\mathcal{L}$time *and is closed under finite variants.* Moreover*, the closure of* $\mathfrak{S}$ *under* outer *composition with* $\mathcal{L}$time *is* $\mathcal{L}$time*-effective for* ψ *in the following somewhat weak sense.*[13] *For each* $g \in \mathcal{L}$time *and each* $m > 0$*, there is a function* comp $\in \mathcal{L}$time *st, for each* $q_1, \ldots, q_m, z$*,*

$$\psi_{\text{comp}(q_1,\ldots,q_m)}(z) = g(\psi_{q_1}(z), \ldots, \psi_{q_m}(z)). \tag{4.31}$$

Assumption 2. ψ *satisfies the* Constructive $\mathcal{L}$time-effective Parametric Recursion Theorem*, i.e., for each* $m > 0$*, there is a function* $r \in \mathcal{L}$time *st, for each* $p, y_1, \ldots, y_m, z$*,*

$$\psi_{r(p,y_1,\ldots,y_m)}(z) = \psi_p(r(p, y_1, \ldots, y_m), y_1, \ldots, y_m, z). \tag{4.32}$$

Intuitively, (4.32) says that ψ*-program* $r(p, \vec{y})$ *has* $p, \vec{y}$ *stored inside, and, on input* z*: it creates a self-copy (seen on the right-hand side of (4.32)); it pulls* $p, \vec{y}$ *out of storage;*

[13] This weak sense is a little more than enough to prove our Theorem 16 below.

and it runs ψ-task/program p on its self-copy, $\vec{y}, z$. Task p represents the use *which $r(p, \vec{y})$ makes of its self-copy/self-knowledge (and of its stored $p, \vec{y}$ and input z).*[14]

We employ the convenient discrete log function from [47, Page 22]: for any $x \in \mathbb{N}$, $\log x \stackrel{\text{def}}{=} (\lfloor \log_2 x \rfloor, \text{ if } x > 1; 1, \text{ if } x \leq 1)$.

4.4.2.2 Results

Theorem 15. $(\mathfrak{S}, \psi) = (\mathcal{P}\text{time}_k, \theta^k)$ *satisfies our Assumptions 1 and 2 just above.*

Proof of Theorem 15. The essential idea behind the proof of this theorem (and of similar results for the other nicely clocked programming systems mentioned above) is that clocked systems *inherit effective* closure properties (which can be thought of as instances of control structures, e.g., see [47, Section 4.2.3] & [46]) from the system, e.g., φ^{TM}, out which they are built.[15]

One gets Assumption 1 *and much more*, for $(\mathcal{P}\text{time}_k, \theta^k)$, from [47, Theorem 6.3(a, b)].

Assumption 2 *and more*, for $(\mathcal{P}\text{time}_k, \theta^k)$, follows from [47, Theorem 6.3(d)].

□ Theorem 15

---

[14] Our Assumptions 1 and 2 imply the analogous pair of assumptions in [9], but, thanks to the weak $\mathcal{L}$time-effectivity part of Assumption 1 above, the assumptions in this chapter are apparently stronger.
Actually, all the classes and corresponding systems mentioned above satisfy much stronger $\mathcal{L}$time-effective closure under compositions (including inner ones too) with $\mathcal{L}$time. For $k \geq 2$, $\mathcal{P}\text{time}_k$ is *not* closed under compositions with $\mathcal{P}\text{time}_k$, but, each of the other subrecursive classes mentioned just above together with corresponding clocked system satisfies full $\mathcal{L}$time-effective closure under arbitrary compositions of functions *within the class*.
Moreover, each class above and corresponding clocked system satisfies the Constructive, $\mathcal{L}$time-effective *Multiple* Parametric Recursion theorem. For some details, see the above mentioned parts of [47].

[15] They also inherit corresponding complexity e.g., Φ^{TM}, properties from the control structures.

As above, let $C_q = \{x \mid \psi_x = \psi_q\}$. As noted above: trivially, each $\overline{C_q}$ *is* c.e., and, as indicated above, the next theorem's proof employs help from Jim Royer. It's proof is constructive *relative to* the limited non-constructivity Σ_1^0-LEM [1].

Theorem 16. *The sequence* $C_q, q = 0, 1, 2, \ldots$ is *uniformly c-productive — as witnessed by a two-argument function* $f \in \mathcal{L}\text{time}$.

Proof of Theorem 16. We let $\Phi^{\text{SlowedDownTM}}$ be the special, *slowed-down/delayed* step-counting measure associated with the acceptable φ^{TM}-system from [47, Theorem 3.20]. In the proof of [47, Theorem 3.20], for the case of $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$, $\Phi^{\text{SlowedDownTM}}$ is obtained from the standard Φ^{TM} measure associated with φ^{TM}, by a log log-delay trick. $\Phi^{\text{SlowedDownTM}}$ has the nice property (among others) that the *predicate* $T \stackrel{\text{def}}{=}$

$$\lambda w, y, z.(1, \text{ if } \Phi_y^{\text{SlowedDownTM}}(w) \leq z; 0, \text{ otherwise}) \in \mathcal{L}\text{time!} \tag{4.33}$$

It is an immediate φ^{TM}-programming exercise to show

$$\mathcal{L}\text{time is closed under definition by } \textit{if-then-else}. \tag{4.34}$$

Hence, by (4.33, 4.34), g just below is clearly $\in \mathcal{L}\text{time}$.

$$g(w, x, y, z) = \begin{cases} 1 + x, & \text{if } T(w, y, z); \\ x, & \text{otherwise.} \end{cases} \tag{4.35}$$

Therefore, by Assumption 1, there is a function $\text{cmp} \in \mathcal{L}\text{time}$ st,

$$(\forall w, q, y, z)[\psi_{\text{cmp}(q)}(w, y, z) = g(w, \psi_q(z), y, z)]. \tag{4.36}$$

Hence, by (4.35, 4.36), for all w, q, y, z,

$$\psi_{\text{cmp}(q)}(w, y, z) = \begin{cases} 1 + \psi_q(z), & \text{if } T(w, y, z); \\ \psi_q(z), & \text{otherwise.} \end{cases} \tag{4.37}$$

By the $m = 2$ case of Assumption 2, we have that there is a function $r \in \mathcal{L}\text{time}$ st, for all p, q, y, z,

$$\psi_{r(p,q,y)}(z) = \psi_p(r(p, q, y), q, y). \tag{4.38}$$

From the $p = \mathrm{cmp}(q)$ case of (4.38) and the $w = r(\mathrm{cmp}(q), q, y)$ case of (4.37), for all, q, y, z,

$$\psi_{r(\mathrm{cmp}(q),q,y)}(z) = \begin{cases} 1 + \psi_q(z), & \text{if } T(r(\mathrm{cmp}(q), q, y), y, z); \\ \psi_q(z), & \text{otherwise.} \end{cases} \tag{4.39}$$

Let $f(q, y) = r(\mathrm{cmp}(q), q, y)$. f is a composition of functions $\in \mathcal{L}\mathrm{time}$; hence, f is also $\in \mathcal{L}\mathrm{time}$.

It is, then, straightforward to verify that, for each $q, y \in \mathbb{N}$,

$$f(q, y) \in (C_q \triangle W_y); \tag{4.40}$$

however, this verification involves consideration of the cases $f(q, y) \in W_y$ vs. not, and this is what employs Σ_1^0-LEM. □ Theorem 16

Future work: *minimize* non-constructivity here & elsewhere, a Reverse Mathematics project.

From here until the end of the present section we will work primarily with $(\mathfrak{S}, \psi) = (\mathcal{P}\mathrm{time}_k, \theta^k)$. To make this clear below, we will mostly write θ^k in place of ψ.

Let f be from Theorem 16 *for the case of* $(\mathfrak{S}, \psi) = (\mathcal{P}\mathrm{time}_k, \theta^k)$. We know already that $f(q, y)$ can be computed in linear-time in the lengths of q, y. Of course, for *each fixed* q, y, $f(q, y)$ is a θ^k-program, and, so, $\theta^k_{f(q,y)}$ runs within time some k-degree polynomial evaluated at the lengths of its arguments.

We are interested next in how hard it is to compute $\theta^k_{f(q,y)}(z)$ *as a function of all of* q, y, z. We, then, run up against the high cost of *universality* for the θ^k-system. From [47, Theorem 6.5], the general cost of universality even for θ^1 is worse than $\mathcal{P}\mathrm{time}$. Kozen [30, Theorem 7.4] showed the general cost of universality for sensible systems for all of $\mathcal{P}\mathrm{time}$ is worse than $\mathcal{P}\mathrm{space}$.

Theorem 17. *For some constants* $b, c > 0$, $\theta^k_{f(q,y)}(z)$ *is computable in*

$$\mathcal{O}(2^{b(ck+1)|(|q|+|y|)|}|z|^k \log |z|) \text{ time.} \tag{4.41}$$

PROOF OF THEOREM 17. Since $f \in \mathcal{L}\text{time}$,

$$(\exists \text{ a constant } b > 0)(\forall q, y)[|f(q, y)| \leq b(|q| + |y|)]. \tag{4.42}$$

Mostly from [47, Theorem 6.4] and parts of its proof, we have: for some constant
$c > 0$, $\theta_p^k(z)$ is computable in

$$\mathcal{O}(2^{(ck+1)|p|}|z|^k \log|z|) \text{ time.} \tag{4.43}$$

In particular, we see that the run time just above is exponential in the size of the program p to be simulated.

The proof of [47, Theorem 6.4] makes use of the inequalities (4.30) above re trans and bound, that they are each in $\mathcal{L}\text{time}$, the sentence *just before* [47, Theorem 4.16], and, also, [47, Corollary 3.7] which provides an upper-bound on the general time cost of *time-bounded* $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$-universal simulation. *That* cost is *exponential in the length of the time-bound's value.*

Therefore, from (4.42, 4.43), we have the desired theorem. □ THEOREM 17

BIBLIOGRAPHY

[1] Y. Akama, S. Berardi, S. Hayashi, and U. Kohlenbach. An arithmetical hierarchy of the law of excluded middle and related principles. *19th Annual Symposium on Logic in Computer Science (LICS'04)*, pages 192–201, 2004.

[2] A. Bauer. Constructive gem: irrational to the power of irrational that is rational, 2009. See `math.andrej.com/2009/12/28/` for the article.

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

[4] M. Blum. On effective procedures for speeding up algorithms. *J. ACM*, 18(2):290–305, 1967. http://doi.acm.org/10.1145/321637.321648.

[5] A. Borodin, R. Constable, and J. Hopcroft. Dense and nondense families of complexity classes. In *Tenth Annual Symposium on Switching and Automata Theory*, pages 7–19, 1969.

[6] D. Bridges and C. Calude. On recursive bounds for the exceptional values in speed-up. *Theoretical Computer Science*, 132:387–394, 1994.

[7] L.E.J. Brouwer. In D. van Dalen, editor, *Brouwer's Cambridge Lectures on Intuitionism*. Cambridge University Press, 1981.

[8] S. Buss. *Bounded Arithmetic*. Bibliopolis, Naples, 1986. Revision of 1985 Ph.D. Thesis: `http://www.math.ucsd.edu/~sbuss/ResearchWeb/BAthesis/` (Department of Mathematics, Princeton University).

[9] J. Case. Effectivizing inseparability. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 37:97–111, 1991. http://www.eecis.udel.edu/~case/papers/mkdelta.pdf corrects missing set complement signs in definitions in the journal version.

[10] J. Case and T. Kötzing. Difficulties in forcing fairness of polynomial time inductive inference. In *20th International Conference on Algorithmic Learning Theory (ALT'09)*, volume 5809 of *Lecture Notes in Artificial Intelligence*, pages 263–277, 2009.

[11] J. Case and M. Ralston. Non-obfuscated yet unprovable programs, 2011. *Asian Logic Conference 2011*, Wellington, NZ. See `www.eecis.udel.edu/~case/slides/obfusc-slides.pdf` for the slides.

[12] J. Case and M. Ralston. Beyond Rogers' non-constructively computable function. In P. Bonizzoni, V. Brattka, and B. Löwe, editors, *The Nature of Computation - Ninth Conference of Computability in Europe (CiE 2013), Proceedings*, volume 7921 of *Lecture Notes In Computer Science*, pages 45–54. Springer, Berlin, 2013.

[13] J. Case, M. Ralston, and Y. Akama. A non-uniformly c-productive sequence & non-constructive disjunctions, 2013. *Asian Logic Conference 2013*, Guangzhou, P.R. China. See `www.eecis.udel.edu/~case/slides/unif-prod-slides.pdf` for the slides.

[14] J.P. Cleave. Creative functions. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 10:205–212, 1961.

[15] M. Davis, R. Sigal, and E. Weyuker. *Computability, Complexity, and Languages*. Academic Press, second edition, 1994.

[16] J. Dekker. Productive sets. *Trans. of AMS*, 78:129–149, 1955.

[17] F. Drake. *Set Theory: An Introduction to Large Cardinals*. North-Holland, 1974.

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

[19] H. Friedman. A proof of padding-once, 1974. Private communication.

[20] M. Genesereth. *Logical Foundations Of Artificial Intelligence*. Morgan Kaufmann, 1987.

[21] K. Gödel. On formally undecidable propositions of Principia Mathematica and related systems I. In S. Feferman, editor, *Kurt Gödel. Collected Works. Vol. I*, pages 145–195. Oxford Univ. Press, 1986.

[22] C. Green. Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 219–239. Morgan Kaufmann, 2081.

[23] P. Halmos. *Naive Set Theory*. Springer-Verlag, NY, 1974.

[24] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[25] S. Hayashi. Mathematics based on incremental learning — excluded middle and inductive inference. *Theoretical Computer Science*, 350:125–139, 2006.

[26] A. Heyting. *Intuitionism: An Introduction*. North-Holland, Amsterdam, 1971. Third edition.

[27] T. Jech. *Set Theory*. Academic Press, NY, 1978.

[28] A. Kanamori. *The Higher Infnite: Large Cardinals in Set Theory from their Beginnings*. Springer-Verlag, 2008.

[29] U. Kohlenbach. Proof theory and computational analysis. *Electronic Notes in Theoretical Computer Science*, 13:124–157, 1998.

[30] D. Kozen. Indexings of subrecursive classes. *Theoretical Computer Science*, 11:277–301, 1980.

[31] R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:155–171, 1975.

[32] M. Machtey. On the density of honest subrecursive classes. Technical report, Computer Science Department, Purdue University, 1973.

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

[34] Y. Marcoux. Composition is almost (but not quite) as good as s-1-1. *Theoretical Computer Science*, 120:169–195, 1993.

[35] E. Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall, London, fifth edition, 2009.

[36] A. Meyer and D. Ritchie. The complexity of loop programs. In *Proceedings of the 22nd National ACM Conference*, pages 465–469. Thomas Book Co., 1967.

[37] J. Myhill. Creative sets. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 1:97–108, 1955.

[38] M. Nakata and S. Hayashi. A limiting first order realizability interpretation. *Scientiae Mathematicae Japonicae Online*, 5:421–434, 2001.

[39] E. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50:284–316, 1944.

[40] H. Putnam. What is innate and why: Comments on the debate. In M. Piattelli-Palmarini, editor, *Language and Learning: The Debate between Jean Piaget and Noam Chomsky*, pages 287–309. Harvard University Press, Cambridge, MA, 1980.

[41] G. Riccardi. *The Independence of Control Structures in Abstract Programming Systems*. PhD thesis, SUNY Buffalo, 1980.

[42] G. Riccardi. The independence of control structures in abstract programming systems. *Journal of Computer and System Sciences*, 22:107–143, 1981.

[43] H. Rogers. Gödel numberings of partial recursive functions. *Journal of Symbolic Logic*, 23:331–341, 1958.

[44] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York, 1967. Reprinted, MIT Press, 1987.

[45] J. Roitman. *Introduction to Modern Set Theory*. 2011. `http://www.math.ku.edu/~roitman/stb3fullWeb.pdf finds the revision to the out of print original`.

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

[47] J. Royer and J. Case. *Subrecursive Programming Systems: Complexity and Succinctness*. Research monograph in *Progress in Theoretical Computer Science*. Birkhäuser Boston, 1994. See `www.eecis.udel.edu/~case/RC94Errata.pdf` for corrections.

[48] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, NJ, third edition, 2012.

[49] C. Schnorr. Does the computational speed-up concern programming? In *Proceedings of the First International Colloquium on Automata, Languages and Programming*. North Holland, Amsterdam, 1972.

[50] C. Schnorr. *Rekursive Funktionen und ihre Komplexität*. Teubner, Stuttgart, 1974.

[51] S. Simpson. *Subsystems of Second Order Arithmetic*. Springer-Verlag, 1999.

[52] A. Troelstra, editor. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer, 1973.

[53] A. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction, Volume I*. Studies in Logic and The Foundations of Mathematics, Number 121. Elsevier, 1988.