NON-OBFUSCATED UNPROVABLE PROGRAMS & MANY RESULTANT SUBTLETIES *

JOHN CASE AND MICHAEL RALSTON

Computer & Information Sciences, University of Delaware, Newark, DE 19716 USA
e-mail address: {case, mralston}@udel.edu

ABSTRACT. The *International Obfuscated C Code Contest* was a programming contest for the most creatively obfuscated yet succinct C code. By *contrast*, an interest herein 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 made by our first theorem, then, is that, then, *un*verifiable programs need *not* be obfuscated!

The first theorem and its proof is followed by a motivated, concrete example based on a remark of Hilary Putnam.

The first theorem has some non-constructivity in its statement and proof, and the second theorem implies some of the non-constructivity is inherent. That result, then, brings up the question of whether there is an acceptable programming system (numbering) for which some non-constructivity of the first theorem disappears. The third theorem shows this is the case, but for a subtle reason explained in the text. This latter theorem has a number of corollaries, regarding its acceptable programming system, and providing some surprises and subtleties about proving its program properties (including universality, and the presence of the composition control structure). The next two theorems provide acceptable systems with contrasting surprises regarding proving universality in them. Finally the next and last theorem (the most difficult to prove in the paper) provides an acceptable system with some positive and negative surprises regarding verification of its true program properties: the existence of the control structure composition is provable for it, but anything about true I/O-program equivalence for syntactically unequal programs is not provable.

INTRODUCTION

The *International Obfuscated C Code Contest* (see the Wikipedia 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*, our first interest herein 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 first theorem (Theorem 2.1 in Section 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 first theorem 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 (but with this never coming true being *un*provable in **T**).

A motivated, concrete, special case, based on a remark in Putnam [21], will be presented (also in Section 2.1 below).

As will be seen, the first theorem and its proof have some non-constructivity, and, with Theorem 2.3 in Section 2.2 below, some of this non-constructivity is seen to be inherent.

Considered next, in Section 2.3, is whether the just mentioned non-constructivity goes away for some acceptable programmming systems (numberings). The answer (Theorem 2.4) is affirmative, but for pleasantly *subtle* reasons spelled out in the section. This latter theorem has a number of corollaries (Corollaries 2.9, 2.10, and 2.11) regarding its acceptable programming system, and they provide some surprises and subtleties about proving its program properties, including in Corollary 2.10, about universality and the presence of the composition control structure.

Section 2.3 makes up most of the paper. Also within it (in Sections 2.3.1 and 2.3.2) are presented a number of positive and negative surprises regarding verification of true program properties.

In Section 2.3.1 Theorems 2.12 and 2.18 provide respective acceptable systems with contrasting surprises regarding proving universality in them. Of course any acceptable system has infinitely many universal programs, but Theorem 2.12 provides an acceptable system in which exactly one of these universal programs is provably so. By contrast, Theorem 2.18 provides a different acceptable system with *no* program which is *provably* universal.

Finally, in Section 2.3.2, the next and last theorem (Theorem 2.21), which is the most difficult to prove in the paper, also provides an acceptable system with some positive and negative surprises regarding verification of its true program properties: for this acceptable system, the existence of the control structure composition is provable for it, but anything about true I/O-program equivalence for syntactically unequal programs is not provable.

1. MATHEMATICAL PRELIMINARIES

1.1. **Complexity-Bounded Computability.** Let φ^{TM} be the *efficiently* laid out and Gödel-numbered acceptable programming system (numbering) from [28, Chapter 3 & Errata] and which is based on deterministic *multi-tape Turing Machines* (with base two I/O).[1] Its programs are named by all the 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 [1]. $(\varphi^{\mathrm{TM}}, \Phi^{\mathrm{TM}})$ is a *base* model for deterministic run time costs. φ^{TM}'s superscript is awkward when φ^{TM} is employed in subscripts, so, from this point on, we will write φ^{TM} as simply φ.

Herein, we will use the linear-time computable and invertible pairing function $\langle \cdot, \cdot \rangle$ from [28]: the binary representation of $\langle x, y \rangle$ is (by definition) an interleaving of the binary representations of x and y where we alternate x's and y's digits and start on the right with the least most significant y digit. For example, $\langle 15, 2 \rangle = 94$ — since $15 = 1111$ (binary), $2 = 0010$ (binary), and $94 = 10101110$ (binary). This function, clearly then, 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 [28], to code also triples, quadruples, ... of elements of $\mathbb{N}$ 1-1, onto $\mathbb{N}$: for all $n > 2$, and all $x_1, \ldots, x_n + 1$, $\langle x_1, \ldots, x_n + 1 \rangle = \langle x_1, \langle x_2, \ldots, x_n + 1 \rangle \rangle$. These functions also clearly satisfy the following

Lemma 1.1.

(1) $\langle x_1, \ldots, x_n \rangle$ *is odd implies* x_n *is odd;*
(2) $\lambda x_1, \ldots, x_n . \langle x_1, \ldots, x_n \rangle$ *is monotonically increasing in each of its arguments; and,*
(3) *for all* $x_1, \ldots, x_n, \max(x_1, \ldots, x_n) \leq \langle x_1, \ldots, x_n \rangle$.

For example, in the proof of Theorem 2.21 below, the just above lemma will see explicit and implicit application.

LinearTime is the class of functions: $\mathbb{N} \to \mathbb{N}$ each computable by some φ-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 defined just above, we can and sometimes will speak of multi-argument functions as being (or not being) in **LinearTime**.

For $k \in \mathbb{N}$, k could be a numerically named program of φ 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.[2]

---

[1]In general, the *acceptable* programming systems [24, 25, 18, 22, 23, 27] can be characterized as those programming systems for all the 1-argument partial computable functions: $\mathbb{N} \to \mathbb{N}$ which are inter-compilable with the natural system φ^{TM}. Rogers [24, 25] characterized the acceptable systems as those with universality and for which Kleene's S-m-n holds. This latter is more than enough to get (not necessarily efficient) recursion theorems in acceptable systems.

[2]This formula can be derived as the minimum 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 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 ($\mathcal{O}$-notation is explained in [8]). A problem comes with complexity bounds of more than one argument. Jim Royer gave the following 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 $\mathcal{O}((f)_+)$.

Rogers [25] uses the terms 'converges' for computations which halt and provide output and 'diverges' for those that do not. Herein we use the respective notations (due to Albert Meyer) ↓ and ↑ in place of those terms of Rogers.

From [28, Lemma 3.14], there are small positive $a \in \mathbb{N}$ and function if-then-else $\in$ **LinearTime** such that, for all $p_0, p_1, p_2, x \in \mathbb{N}$,

$$\varphi_{\text{if-then-else}(p_0,p_1,p_2)}(x) = \begin{cases} \varphi_{p_1}(x), & \text{if } \varphi_{p_0}(x)\downarrow \neq 0; \\ \varphi_{p_2}(x), & \text{if } \varphi_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise}; \end{cases} \tag{1.1}$$

and

$$\Phi^{\text{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) \leq \begin{cases} a \cdot (\Phi^{\text{TM}}_{p_0}(x) + \Phi^{\text{TM}}_{p_1}(x))_+, & \text{if } \varphi_{p_0}(x)\downarrow \neq 0; \\ a \cdot (\Phi^{\text{TM}}_{p_0}(x) + \Phi^{\text{TM}}_{p_2}(x))_+, & \text{if } \varphi_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise}. \end{cases} \tag{1.2}$$

Essentially from (the $k, m = 1$ case of) [28, Theorem 4.8], we have the following constructive, *efficient*, and *parametrized* version of *Kleene's* 2nd (not Rogers') Recursion Theorem [25, Page 214].

There are small positive $b \in \mathbb{N}$ and function krt $\in$ **LinearTime** such that, for all *parameter values p, tasks r, inputs $x \in \mathbb{N}$*:

$$\varphi_{\text{krt}(p,r)}(x) = \varphi_r(\langle \text{krt}(p,r), p, x \rangle); \tag{1.3}$$

and

$$\Phi^{\text{TM}}_{\text{krt}(p,r)}(x) \leq b \cdot (|p| + |r| + |x| + \Phi^{\text{TM}}_r(\langle \text{krt}(p,r), p, x \rangle)). \tag{1.4}$$

Intuitively, above in (1.3), on the left-hand side, the φ-program krt(p, r) *has p, r stored inside*, and, on x, it: *makes a self-copy* (in linear-time), *forms $y = \langle self\text{-}copy, p, x \rangle$* (in linear-time), and *runs task r on this y*. From (1.4) just above, for each p, r, any super-linear cost of running φ-program krt(p, r) on its input is from the running of φ-task r on its linear-time producible input.

1.2. **Computably Axiomatized, Powerful, True Theories.** Let **T** be a *computably axiomatized* first order (fo) theory extending fo Peano arithmetic (**PA**) [20, 25] — *but with numerals represented in base two to avoid size blow up from unary representation* (see [3, Page 29])[3] — *and which does not prove (standard model for* **PA** *[20]) falsehoods expressible in f.o. arithmetic.*

---

[3] Lets 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., [20], the numeral $\overline{n}$ for natural number n is $S^{(n)}(\overline{0})$, where $S^{(0)}(\overline{0}) = \overline{0}$ and $S^{(n+1)}(\overline{0}) = S(S^{(n)}(\overline{0}))$ — featuring 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 [3, Page 29], herein, by constrast 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 $(n > 1)$, $n \in \mathbb{N}$,

$$\overline{2n} = (\overline{2} \cdot \overline{n}); \tag{1.5}$$

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

$$\overline{(2n) + 1} = S(\overline{2n}). \tag{1.6}$$

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

T could be, *for example*: fo Peano arithmetic (**PA**) itself, the two-sorted fo Peano arithmetic permitting quantifiers over numbers and sets of numbers [25, 29] (a second order arithmetic), Zermelo Frankel Set Theory with Choice (**ZFC**) [13], **ZFC** + ones favorite large cardinal axiom [26, 14, 9, 15], etc.

If E is an expression such as 'the partial function computed by φ-program number p is total' and which is expressible in **PA** and where p is a particular element of $\mathbb{N}$, we shall write $\ll$ 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* $\ll$ E′ $\gg$ *can be algorithmically obtained from* $\ll$ E $\gg$ *in linear-time in* $(|\ll$ E $\gg|+|k|)$*.*

By [28, Theorem 3.6 & Corollary 3.7] and their proofs, the running of a *carefully crafted, time-bounded, φ-universal simulation* up through time t takes time a little worse than exponential in $|t|$. Early complexity theory, e.g., [2, 17, 16], provided delaying tricks to achieve polynomial time. From [28, Theorem 3.20] and its proof, the above mentioned carefully crafted, time-bounded universal simulation of any φ-program can be *uniformly delayed* by a log log factor on the time-bound to run *in* **LinearTime**.

The theorems of **T** form a *computably enumerable* set, so we can/do fix a predicate logic complete automatic theorem prover (such as resolution) for **T**. This theorem prover can be time-bounded universally simulated — but, as in the just prior paragraph, that simulation can be *delayed* by a log log factor on the time-bound to, then, run *in* **LinearTime**. Let

$$\mathbf{T} \vdash_x \ll \mathsf{E} \gg \tag{1.7}$$

mean that *a delayed by such a* log log *factor, linear-time computable, time-bounded universal simulation* of the fixed automatic theorem prover proves $\ll$ E $\gg$ from **T** *within* x steps — that's *linear-time in* $(|\ll$ E $\gg|+|x|)$.

Let D_x be the finite set $(\subseteq \mathbb{N})$ with canonical index x (see, e.g., [25]). x codes, for example, *both* how to list D_x *and* how to know when the listing is done. Herein, 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 $\{u, u+v\}$, let the code be $\langle u, v \rangle + 1$. This coding (suggested by a referee to replace our original one) is linear-time codable/decodable (and is 1-1, and, unlike our original, is onto).[4]

2. Results

2.1. **Non-Obfuscated Unprovable Programs.**

Theorem 2.1. *There* exists $g \in$ **LinearTime** *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:*

$$\varphi_q = \varphi_p; \tag{2.1}$$

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

$$\Phi_q^{\mathrm{TM}}(x) \le c \cdot (|p| + |x| + \Phi_p^{\mathrm{TM}}(x)); \tag{2.2}$$

$$|q| \le d \cdot |p|; \tag{2.3}$$

---

[4] As an aside: [7] canonically codes *any size* finite sets in cubic time & decodes them in linear-time.

yet

$$\mathbf{T} \not\vdash \ll \varphi_q = \varphi_p \gg. \tag{2.4}$$

Our proof below of Theorem 2.1, as will be seen, makes it *easily transparent* that $\varphi_q = \varphi_p$. Hence, q is *not obfuscated*, yet its correctness (at computing φ_p), 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 [28, 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 (2.4) above), it bounces off the rubber wall (i.e., contradiction) toward our goal, here (2.4), instead.[5]

Proof of Theorem 2.1.

By *two* applications of *linear-time*: krt, if-then-else (these from Section 1.1 above), and $\lambda \mathsf{E}, x\,.\,(\mathbf{T} \vdash_x \mathsf{E})$ (this from Section 1.2 above), from *any* φ-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,

$$\varphi_{e_{1,p}}(x) = \begin{cases} \varphi_p(x) + 1, & \text{if } \mathbf{T} \vdash_x \ll \varphi_{e_{1,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise;} \end{cases} \tag{2.5}$$

and

$$\varphi_{e_{2,p}}(x) = \begin{cases} 0, & \text{if } \mathbf{T} \vdash_x \ll \varphi_{e_{2,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2.6}$$

Let $g \in \mathbf{LinearTime}$ be such that, for each p, $D_{g(p)} = \{e_{1,p}, e_{2,p}\}$. We consider cases regarding p for the choice of the associated $q \in D_{g(p)}$.

Case (1). domain(φ_p) is infinite. Suppose for contradiction, for some x, $\mathbf{T} \vdash_x \ll \varphi_{e_{1,p}} = \varphi_p \gg$. Since, by assumption, $\mathbf{T}$ *does not prove false such sentences*, $\varphi_{e_{1,p}} = \varphi_p$, and by (2.5) above, for all $x' \geq x$, $\varphi_{e_{1,p}}(x')$ *also* $= \varphi_p(x') + 1$, but, since domain(φ_p) is infinite, we have a contradiction. Choose $q = e_{1,p}$. Then, trivially, again by (2.5), $\varphi_q = \varphi_p$, but $\mathbf{T}$ does not prove it.

Case (2). domain(φ_p) is finite. Suppose for contradiction, for some x, $\mathbf{T} \vdash_x \ll \varphi_{e_{2,p}} = \varphi_p \gg$. Since, by assumption, $\mathbf{T}$ *does not prove false such sentences*, $\varphi_{e_{2,p}} = \varphi_p$, and by (2.6) above, for all $x' \geq x$, $\varphi_{e_{2,p}}(x')$ also $= 0$, making domain($\varphi_{e_{2,p}}$) infinite, and, hence, domain(φ_p) is infinite, a contradiction. Choose $q = e_{2,p}$. Then, trivially, again by (2.6), $\varphi_q = \varphi_p$, 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, $\lambda \mathsf{E}, x\,.\,(\mathbf{T} \vdash_x \mathsf{E}) \in \mathbf{LinearTime}$, and by the complexity upper bounds (1.2) and (1.4) (in Section 1.1 above) as well as the assertion (in Section 1.2 above) of the linear-time (and, hence, linear size) cost of substituting numerals into formulas of $\mathbf{PA}$, we have small positive c, d such that the theorem's time complexity bound (2.2) and it's program size bound (2.3) above each hold.   ☐ THEOREM 2.1

Next is the promised, motivated, concrete example.

Putnam [21] notes that the typical inductive definitions of grammaticality (i.e., well-formedness) for propositional logic formulas parallel the typical definitions of truth (under

---

[5]More discussion on identifying contradictions with walls, a.k.a. boundaries, can be found on [28, Page 131].

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.[6] 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-program. Then by Theorem 2.1 above and its proof also above, there is an obviously semantically equivalent φ-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* [25] q is a bit unlike p — since it performs an always false (quick) test p doesn't.

2.2. **A Constructivity Concern.** It's interesting to ask: can the condition $|D_{g(p)}| = 2$ in Theorem 2.1 be improved to $|D_{g(p)}| = 1$? If so, it makes sense to replace a singleton set, $\{q\}$, by just q and use $g(p) = q$ (not the code of $\{q\}$). Anyhow, the answer to the question is, No (see Theorem 2.3 below). Before we present and prove this theorem, it is useful to have for its proof the unsurprising lemma (Lemma 2.2) just below.[7]

Lemma 2.2. *If $\varphi_p(x)\downarrow = y$, then*

$$\mathbf{PA} \vdash \ll \varphi_p(x)\downarrow = y \gg. \tag{2.7}$$

Proof of Lemma 2.2. The relation, in p, x, y, t, that holds iff $\varphi_p(x)\downarrow = y$ within t steps, where the steps are measured by the natural Φ^{TM}, is trivially computable (a.k.a. recursive) [1].

Suppose $\varphi_p(x)\downarrow = y$. Then there is some t such that $\varphi_p(x)\downarrow = y$ within t steps. By Gödel's Lemma [12, 20] that recursive relations are numeralwise provably-representable in, e.g., **PA**, **PA** $\vdash \ll \varphi_p(x)\downarrow = y$ within t steps $\gg$. By existential generalization *inside* **PA**, we have **PA** $\vdash \ll (\exists t)[\varphi_p(x)\downarrow = y$ within t steps] $\gg$. Hence, **PA** $\vdash \ll \varphi_p(x)\downarrow = y \gg$.
$$\square \text{ LEMMA 2.2}$$

The next theorem implies that, in Theorem 2.1 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 2.3) provides positive cases regarding proving true program properties in **PA**.

Theorem 2.3. *It is* not *the case that there exists computable g such that, for any p, for $q = g(p)$,*

$$\mathbf{T} \nvdash \ll \varphi_q = \varphi_p \gg. \tag{2.8}$$

---

[6]In computer science these inductive definitions would be called recursive and, as program code, can easily be run iteratively — for efficiency.

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

Proof of Theorem 2.3. Suppose for contradiction otherwise.

Suppose d is a φ-program for g, i.e., suppose $\varphi_d = g$.

Of course $\lambda p, x \,\textbf{.}\, [\varphi_{\varphi_d(p)}(x)]$ is partially computable, and, importantly, this is provable in **PA**. We sketch how we know the provability in **PA**.

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

$$\textbf{PA} \vdash \,\ll (\forall p, x)[\varphi_u(p, x) = \varphi_p(x)] \gg. \tag{2.9}$$

For φ, the construction of a relatively efficient, but *time-bounded variant* of such a u is outlined in the proof of [28, Theorem 3.6]. This construction can be altered to remove the time-boundedness and just get a suitable u.

Another step would be to spell out a φ-program c for a computable function comp_2 for computing a φ-program for the composition of the partial functions computed by its φ-program arguments as in the $m = 2$ case of [28, Lemma 3.10] [8] and its proof — where, again, **PA** proves correctness (including $\mathrm{comp}_2 = \varphi_c$ is total).

Relevance of u and c: clearly we have,

$$\varphi_{\varphi_d(p)}(x) = \varphi_u(\varphi_d(p), x), \tag{2.11}$$

and the right-hand side of (2.11) just above is a relevant composition and, then, can be further expanded employing c.[9] With u and c, then, we can explicitly compute a φ-program for $\lambda p, x \,\textbf{.}\, [\varphi_{\varphi_d(p)}(x)]$ and prove it correct in **PA**.

So, then, by the Constructive Kleene's Second Recursion Theorem *but without the parameter p as above in Section 1.1 above*, we have a (self-referential) p_0 such that, for any x,

$$\varphi_{p_0}(x) = \varphi_{\varphi_d(p_0)}(x). \tag{2.12}$$

Below we'll refer to this parameter-free version of the above Constructive Kleene Theorem as **KRT**. Then we have a φ-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 provable in **PA**.

Hence, by our remarks above about computing and proving correct a program for $\lambda p, x \,\textbf{.}\, [\varphi_{\varphi_d(p)}(x)]$, we can explicitly compute a p_0 as in (2.12) and prove it correct in **PA**; we have in particular,

$$\textbf{PA} \vdash \,\ll \varphi_{p_0} = \varphi_{\varphi_d(p_0)} \gg. \tag{2.13}$$

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_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-program for g, we have that $\varphi_d(p_0)\!\downarrow\,=$ to some explicit numerical value q_0. Therefore, from Lemma 2.2 above,

$$\textbf{PA} \vdash \,\ll \varphi_d(p_0)\!\downarrow\,= q_0 \gg. \tag{2.14}$$

---

[8] In that Lemma 3.10, we have, in effect, for the arbitrary m case, for all $p_0, \ldots, p_m$,

$$\varphi_{comp_m(p_0,\ldots,p_m)}(x) = \varphi_{p_0}(\varphi_{p_1}(x), \ldots, \varphi_{p_m}(x)): \tag{2.10}$$

[9] Further below in Section 2.3, we'll consider, among other things, some programming systems with provability subtleties regarding universality and/or composition. *This* composition, though, we be as in the $m = 1$ case of [28, Lemma 3.10] (see Footnote 8 just above).

Hence, by substitution of equals for equals and reflexivity of equals *inside* **PA**, (2.13), and (2.14),

$$\textbf{PA} \vdash \ll \varphi_{q_0} = \varphi_{p_0} \gg, \tag{2.15}$$

a contradiction to our beginning assumption — since **T** extends **PA**. □ THEOREM 2.3

2.3. **Subtleties.** So far we have considered the natural, deterministic complexity theory relevant, acceptable system, φ. After we obtained Theorem 2.3 just above — which shows a condition in Theorem 2.1 further above (in Section 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 2.3. We initially obtained the first part of the next theorem (Theorem 2.4) which provides such a ψ, *but* we didn't, then, know whether our ψ was acceptable. We subsequently obtained Theorem 2.4'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 2.4 and before its proof.

Theorem 2.4. *There is a programming system ψ and a computable g such that, for all p, $\psi_{g(p)} = \psi_p$, yet, for $q = g(p)$, $\textbf{T} \nvdash \ll \psi_q = \psi_p \gg$.*

Furthermore, ψ is acceptable, and, surprisingly,

$$(\forall p)[\psi_p = \varphi_p]. \tag{2.16}$$

How can (2.16) be true — in the light of the rest of the just above theorem (Theorem 2.4)? It seems to contradict Theorem 2.3 further above. The answer is that, in the proof just below of the just above theorem (Theorem 2.4), the needed ψ is, in effect, *defined by* an unusual φ-program e in (2.19, 2.20) below, and, *in the language of* **PA**, for formulating (un)provability *about ψ* in **T**, ψ is, of course, *represented by its defining e.*[10] φ itself, on the other hand, can be and is understood to be naturally (not unusually) represented in the language of **PA**.[11]

To aid us in some proofs below, including that of the above Theorem 2.4, we present the following lemma (Lemma 2.5), where the recursion theorem part of its proof is from H. Friedman [11].

Lemma 2.5 (**T**-Provable Padding-Once)**.** *Suppose α is any acceptable programming system such that $\textbf{T}$ proves α's acceptability.*

Then, there is a total computable function g such that for any p, $g(p) \neq p$, but $\alpha_{g(p)} = \alpha_p$.[12]

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

Proof of Lemma 2.5. Assume the hypothesis, i.e., that **T** proves α's acceptability.

Then **T** proves Kleene's S-m-n Theorem, so we obtain that **T** proves the Parameterized Second Kleene Recursion Theorem (as above in Section 1.1, but with witnessing functions not necessarily in **LinearTime**).

---

[10] An original source for unusual representations in arithmetic (as is our e) is [10].

[11] See the informal discussion about the notation $\ll \mathsf{E} \gg$ in Section 1.2 above, where, in effect, the particular example $\ll \varphi_p$ is total $\gg$ is employed.

[12] Of course, a more general, constructive *infinite* padding holds [18], and we need a version of that further below.

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

$$\alpha_{f(p)}(x) = \begin{cases} \alpha_p(x), & \text{if } f(p) \neq p; \\ \alpha_{p+1}(x), & \text{if } f(p) = p. \end{cases} \tag{2.17}$$

Then, let g be defined as follows.

$$g(p) = \begin{cases} f(p), & \text{if } f(p) \neq p; \\ p+1, & \text{if } f(p) = p. \end{cases} \tag{2.18}$$

We consider two cases.

Case one: $f(p) \neq p$. Then, from (2.17), $\alpha_{f(p)} = \alpha_p$, and, from (2.18), $g(p) = f(p) \neq p$.

Case two: $f(p) = p$. Then, from (2.17), $\alpha_{f(p)} = \alpha_{p+1}$, which, by Case two, $= \alpha_p$. From (2.18), $g(p) = p + 1 \neq p$.

By this case-analysis, g satisfies Padding-Once. The above is so simple as to be provable in **T** — as needed. $\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$ LEMMA 2.5

Proof of Theorem 2.4. By Kleene's second recursion theorem (again without parameter), there is a (self-referential) φ-program e and an associated ψ both such that, for each p, x,

$$\psi_p(x) \stackrel{\text{def}}{=} \varphi_e(\langle p, x \rangle), \text{ which } = \tag{2.19}$$

$$\begin{cases} p, & \text{if } \mathbf{T} \vdash_x \ll (\exists q, r \mid q \neq r)[\psi_q = \psi_r] \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2.20}$$

N.B. The mentions of ψ in (2.20) just above with variable subscripts q, r should be understood, employing ψ's definition (2.19) above, to be $\varphi_e(\langle q, \cdot \rangle), \varphi_e(\langle r, \cdot \rangle)$, respectively.

Claim 2.6. $\mathbf{T} \nvdash \ll (\exists q, r \mid q \neq r)[\psi_q = \psi_r] \gg$.

Proof of Claim 2.6. 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 (2.19, 2.20) 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, **T** has proven a sentence of first order arithmetic which is false in the standard model, a contradiction. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$ CLAIM 2.6

Claim 2.7. $(\forall p)[\psi_p = \varphi_p]$; hence, ψ is acceptable.

Proof of Claim 2.7. By Claim 2.6, the first clause in (2.20) above is false for each p, x. Therefore, by (2.19, 2.20) above, $(\forall p, x)[\varphi_e(\langle p, x \rangle) = \varphi_p(x)]$; hence, $(\forall p)[\psi_p = \varphi_p]$ — making ψ acceptable too. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$ CLAIM 2.7

Claim 2.8. 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 2.8. The acceptability of φ is provable in **PA**, hence, in **T**. By Lemma 2.5, there exists a computable g such that $(\forall p)[g(p) \neq p \ \wedge \ \varphi_{g(p)} = \varphi_p]$. By Claim 2.7, $\psi = \varphi$; 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 2.2 above), $\mathbf{PA} \vdash \ll q \neq p \gg$. Then, by this and existential generalization in **T**, $\mathbf{T} \vdash \ll (\exists q, r \mid q \neq r)[\psi_q = \psi_r] \gg$, a contradiction to Claim 2.6 above. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ □ CLAIM 2.8

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ □ THEOREM 2.4

For the next three corollaries (Corollaries 2.9, 2.10, and 2.11), the mentioned ψ is that from Theorem 2.4 and its proof, including (2.19, 2.20) above.

Corollary 2.9. $\psi = \varphi$, *but* $\mathbf{T} \nvdash \ll \psi = \varphi \gg$.

Proof of Corollary 2.9. $\psi = \varphi$ is from Theorem 2.4 above. Suppose for contradiction $\mathbf{T} \vdash \ll \psi = \varphi \gg$.

Then, from this and the proof of Theorem 2.3 above, one obtains a Theorem 2.3 *but* with ψ replacing φ. This contradicts Theorem 2.4 above (which is also about ψ). $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ □ COROLLARY 2.9

To understand the corollary (Corollary 2.10) and its proof just below, it may be useful to review the roles of φ-programs u, c, k in the proof of Theorem 2.3 above. This corollary says there can be *no* analog of *all three* of these programs for ψ (in place of φ).

Corollary 2.10. *There are* no u, c, k *such that* simultaneously:

$$\mathbf{T} \vdash \ll u \text{ is a witness to universality in } \psi \gg, \tag{2.21}$$

$$\mathbf{T} \vdash \ll c \text{ is a witness to composition in } \psi \gg^{13}, \& \tag{2.22}$$

$$\mathbf{T} \vdash \ll k \text{ is a witness to } \mathbf{KRT} \text{ in } \psi \gg. \tag{2.23}$$

Proof of Corollary 2.10. Suppose for contradiction otherwise. Then, enough is provable in **T** *about* ψ to make Theorem 2.3 above *also* provable for ψ — in place of φ. This contradicts Theorem 2.4 about ψ. $\qquad\qquad\qquad\qquad\qquad\qquad$ □ COROLLARY 2.10

Regarding Corollary 2.10 just above, it is well known that, from [18, 19], 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 (2.23) 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 2.10 we'd ostensibly also need these auxiliary ψ-programs to be **T**-provably correct.[14] The **T**-provable correctness is the hard part.

Corollary 2.11. $\mathbf{T} \nvdash \ll \psi \text{ is acceptable} \gg$.

---

[13] *This* composition is the $m = 2$ case of [28, Lemma 3.10] (see Footnote 8 further above).

[14] Machtey and Young's construction [18, 19] of an S-m-n function out of a composition function, for example, employs auxiliary ψ-programs q_0, q_1 such that

$$\psi_{q_0} = \lambda z \,\centerdot\, \langle 0, z \rangle; \text{ and } \psi_{q_1} = \lambda \langle y, z \rangle \,\centerdot\, \langle y + 1, z \rangle. \tag{2.24}$$

Marcoux's more efficient solution [19] employs three such auxiliary ψ-programs.

Proof of Corollary 2.11.   Assume for contradiction otherwise. Then, by Lemma 2.5 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 2.6
above.   □ Corollary 2.11

2.3.1. *Subtleties About Proving Universality.* The next two theorems herein (Theorems 2.12
and 2.18) provide two more acceptable programming systems, η, θ, respectively, each defined
(as was ψ above) by respective, unusual φ-programs. The first of these theorems (Theo-
rem 2.12) provides a surprise, part positive, part negative, regarding proving *in* $\mathbf{T}$ that
universality holds for η. The contrast between these last two theorems is also interesting.
Of course, since each of η, θ is acceptable, universality holds for each of them (at least
outside $\mathbf{T}$).

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

Theorem 2.12. *There exists an acceptable programming system η and an e such that*
$\mathbf{PA} \vdash \ll e \text{ is universal for } \eta \gg$, *yet, surprisingly, for each p,*

$$\text{If } \mathbf{T} \vdash \ll p \text{ is universal for } \eta \gg, \text{ then } p = e. \tag{2.25}$$

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

Proof of Theorem 2.12.   The Kleene Second Recursion Theorem provides a φ-program e
and an associated η both such that, for each p, x,

$$\eta_p(x) \overset{\text{def}}{=} \varphi_e(\langle p, x\rangle), \text{ which } = \tag{2.26}$$

$$\begin{cases} p, & \text{if } [p \neq e \ \wedge \ \mathbf{T} \vdash_x \ll (\exists q, r \mid r \neq q)[\eta_q = \eta_r] \gg]; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2.27}$$

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

Claim 2.13. $\mathbf{T} \nvdash \ll (\exists q, r \mid q \neq r)[\eta_q = \eta_r] \gg$.

Proof of Claim 2.13.   Assume for contradiction that

$$\mathbf{T} \vdash \ll (\exists q, r \mid q \neq r)[\eta_q = \eta_r] \gg. \tag{2.28}$$

Let x_0 be the minimum number of steps in any such proof. Since $\mathbf{T}$ does not prove false
sentences of $\mathbf{PA}$ and (2.28), we have

$$(\exists q, r \mid q \neq r)[\eta_q = \eta_r]. \tag{2.29}$$

Then, for $f(p) = \varphi_e(\langle p, x_0\rangle)$, $\rho(f) \supseteq (\mathbb{N} - \{e\})$. Clearly, $\eta_e = \varphi_e$, and η_e has infinite range.
Furthermore, $(\forall p \neq e)(\forall x \geq x_0)[\eta_p(x) = p]$; therefore, $(\forall p \neq e)[\eta_p \text{ has finite range}]$, and,
thus, there is no η-program whose code number is not e whose computed partial 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 partial function, a contradiction
to (2.29).   □ Claim 2.13

Claim 2.14. $(\forall p, x)[\eta_p(x) = \varphi_p(x)]$; hence, η is acceptable.

Proof of Claim 2.14. By Claim 2.13, the if clause of (2.27) is always false, hence, by (2.26, 2.27), $(\forall p, x)[\eta_p(x) = \varphi_e(\langle p, x \rangle) = \varphi_p(x)]$. $\qquad \square$ CLAIM 2.14

Claim 2.15. There does not exist p, q such that $p \neq q$, and $\mathbf{T} \vdash \ll \eta_p = \eta_q \gg$.

Proof of Claim 2.15. Suppose for contradiction otherwise. Then, by Gödel's Lemma followed by existential generalization, the latter in $\mathbf{T}$, we obtain a contradiction to Claim 2.13. $\qquad \square$ CLAIM 2.15

Claim 2.16. $\mathbf{PA} \vdash \ll e$ is universal for $\eta \gg$.

Proof of Claim 2.16. We need *not* prove in $\mathbf{PA}$ that η is a programming system for the 1-argument partial computable functions. Instead, it suffices for us to argue only that

$$\mathbf{PA} \vdash \ll (\forall p, x)[\eta_e(\langle p, x \rangle) = \eta_p(x)] \gg. \tag{2.30}$$

Then, from (2.26) above, the definition of η by e in the φ-system, applied to each side of (2.30), it, then, suffices to show that

$$\mathbf{PA} \vdash \ll (\forall p, x)[\varphi_e(\langle e, \langle p, x \rangle \rangle) = \varphi_e(\langle p, x \rangle)] \gg. \tag{2.31}$$

By the otherwise clause of (2.26, 2.27) above, applied to $\ll \varphi_e(\langle e, \langle p, x \rangle \rangle) \gg$, where p, x are variables (not numerals), we get its provable in $\mathbf{PA}$ value to be $\ll \varphi_e(\langle p, x \rangle) \gg$ — again with p, x variables. This together with universal generalization *inside* $\mathbf{PA}$ on the variables p, x, verifies *in* $\mathbf{PA}$ the sufficient (2.31) just above. $\qquad \square$ CLAIM 2.16

Claim 2.17. For all $p \neq e$, $\mathbf{T} \nvdash \ll p$ is universal in $\eta \gg$.

Proof of Claim 2.17. Immediate from Claims 2.15 and 2.16. $\qquad \square$ CLAIM 2.17

$\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \square$ THEOREM 2.12

Theorem 2.18. *There exists an acceptable programming system θ such that, for each u,*

$$\mathbf{T} \nvdash \ll u \text{ is universal in } \theta \gg. \tag{2.32}$$

Of course, in θ, there are infinitely many universal programs, but none *are provably so in* $\mathbf{T}$. *Furthermore, θ turns out to be φ.*

Proof of Theorem 2.18. Kleene's Recursion Theorem provides a φ-program e and an associated θ both such that, for each p, x,

$$\theta_p(x) \stackrel{\text{def}}{=} \varphi_e(\langle p, x \rangle) \text{ which } = \tag{2.33}$$

$$\begin{cases} p, & \text{if } \mathbf{T} \vdash_x \ll (\exists u)[u \text{ is universal in } \theta] \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2.34}$$

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

Furthermore, it is not the case that $(\exists u)[\mathbf{T} \vdash \ll u$ is universal in $\theta \gg]$, since, if $\mathbf{T}$ proved such a thing, it would immediately follow from Existential Generalization *in* $\mathbf{T}$ that $\ll (\exists u)[u$ is universal in $\theta] \gg$ is provable in $\mathbf{T}$, which has already been shown not to be provable by $\mathbf{T}$. $\qquad\qquad\square$ THEOREM 2.18

We expect that analogs of Theorems 2.12 and 2.18 just above can be obtained for other properties besides universality. In the next section we have an analog for composition.

2.3.2. *Provable Composition with Unprovable Program Equivalence.* The main result of this section (Theorem 2.21 below): there is an acceptable programming system (equivalent to φ) such that $\mathbf{T}$ can prove there exists a specific program which witnesses composition in that system, but $\mathbf{T}$ is *still* unable to prove that there exist two distinct, equivalent programs in that system. It is the last and hardest to prove result in the present paper.

In the proof of this theorem, we require provable in $\mathbf{PA}$ *infinite* padding for just the φ-system — a stronger form of padding than provided by Lemma 2.5, though this *infinite* padding function is only valid for φ, and thus would not have been usable for Corollary 2.11. As such, we introduce the following function pad, which uses the concepts from [28, Chapter 3 but as modified in the associated Errata] of normal and abnormal code numbers of φ-programs. Herein we briefly discuss these concepts. A program in the φ-system is defined as a *non*-empty sequence of instructions for a k-tape Turing machine, where all these instructions have the same k. If a given number is not directly the code for such a sequence, it is defined to be an *abnormal* code; otherwise, it is a *normal* code. The process of checking a given number to see if it is normal or abnormal is computable.[15] By convention, abnormal codes are treated as each encoding the same Turing machine, and thus all compute the same function. It is a consequence of the encoding used for the φ-system that all normal codes are divisible by eight; thus, there are infinitely many even abnormal codes — for instance, anything divisible by two but not eight is an even abnormal code.

$$\mathrm{pad}(p) \overset{\text{def}}{=} \begin{cases} \text{the next even abnormal code,} & \text{if } p \text{ is an abnormal code;} \\ p \text{ with the last instruction repeated,} & \text{if } p \text{ is a normal code.} \end{cases} \qquad (2.35)$$

Claim 2.19. For any input p, the output of $\mathrm{pad}(p)$ is the code number of an even program q such that $q > p$, $\varphi_q = \varphi_p$, and $\mathbf{PA}$ proves this.

Proof of Claim 2.19. As above, determining whether a given number is a normal code is algorithmic. Furthermore, as there are infinitely many even abnormal codes, finding the next such is straightforward. Thus, the first clause in the definition of pad is computably checkable and, if true, pad outputs an even number greater than the input, such that both are abnormal codes. As noted above, each abnormal code is defined (for the φ-system) to compute the same function as each other abnormal code, and, hence, the part of the claim not about $\mathbf{PA}$ holds for such p.

---

[15]In fact, it is *linear-time* checkable although nothing in the proofs herein makes use of that fact.

If the second clause holds, then converting p into a coded sequence of instructions is simple, repeating the last instruction is also simple, and converting that sequence back into a code number is once again simple. By [28], when there are multiple instructions that apply in a given state, the *first* one is the one that applies, thus repeating an instruction can have no impact on the (possibly partial) function computed by a φ program; thus, $\varphi_{\text{pad}(p)} = \varphi_p$ in this case. Furthermore, as noted above, all normal instruction codes *must* be even; thus, $\text{pad}(p)$ is even in this case as well.

Therefore, the part of the claim not about **PA** holds. The proof so far is so simple that it can be carried out in **PA** straightforwardly albeit tediously. $\square$ CLAIM 2.19

In our proof of the next theorem (Theorem 2.21) it is convenient to employ the following lemma (Lemma 2.20), a new recursion theorem which mixes an n-ary (non-parameter) version of the original Kleene Recursion Theorem [25, Page 214] with the Delayed Recursion Theorem [4, Theorem 1].[16]

Lemma 2.20 (A Mixed Recursion Theorem). *Suppose $n > 0$. Suppose $\xi_1, \ldots, \xi_n, \xi$ are partial computable.*

Then there are $e_1, \ldots, e_n, c$ such that φ_c is total, and, for each i with $1 \leq i \leq n$, for all x, y,

$$\varphi_{e_i}(y) = \xi_i(e_1, \ldots, e_n, c, y), \tag{2.36}$$

and

$$\varphi_{\varphi_c(x)}(y) = \xi(e_1, \ldots, e_n, c, x, y). \tag{2.37}$$

In Lemma 2.20 just above, (2.36) expresses the n-ary Kleene Recursion Theorem part, and (2.37) expresses the Delayed Recursion Theorem part.

Theorem 2.21. *There is an acceptable programming system ζ and a w such that $\textbf{PA} \vdash \ll w$ is a witness to composition in $\zeta \gg$, yet $\textbf{T} \nvdash \ll (\exists r, t \neq r)[\zeta_r = \zeta_t] \gg$. Furthermore, ζ turns out to be φ.*

Proof of Theorem 2.21.   We apply the $n = 3$ case of Lemma 2.20 just above to obtain programs e, c, w', w behaving as below in (2.39, 2.40, 2.41, 2.42), respectively.[17] The ζ we need for the theorem is defined in terms of this e thus. For each p, x,

$$\zeta_p(x) \stackrel{\text{def}}{=} \varphi_e(p, x). \tag{2.38}$$

For convenience below, in describing the behavior of e, c, w', w, in many places we'll write this ζ instead of φ_e.

In the following formula we let $\text{prime}(n)$ be the nth prime, where $\text{prime}(0) = 2$, $\text{prime}(1) = 3, \ldots$. Importantly to the combinatorics of the diagonalization below in this proof, any odd prime raised to a power is odd, and we noted before the statement of the present theorem being proved (Theorem 2.21) that our particular infinite padding function pad always outputs even numbers.

---

[16]In [4] the Delayed Recursion Theorem is used to prove the Operator Recursion Theorem [4, 6]. In [5] the proof of its Remark 1 employs a *subrecursive* Delayed Recursion Theorem.

[17] Our application of Lemma 2.20 here does not and does not need to make full use of all the self/other reference available in this lemma.

For each p, x,

$$\varphi_e(p, x) = \begin{cases} \varphi_p(x), & \text{if } p = w \ \lor \ (\exists y < p)[\varphi_w(y) = p] \ \lor \\ & \quad \mathbf{T} \not\vdash_x \ll (\exists r, t \neq r)[\zeta_r = \zeta_t] \gg; \\ \text{prime}(p+1)^x, & \text{if } p \neq w \ \land \ (\forall y < p)[\varphi_w(y){\downarrow} \neq p] \ \land \\ & \quad \mathbf{T} \vdash_x \ll (\exists r, t \neq r)[\zeta_r = \zeta_t] \gg; \\ {\uparrow}, & \text{otherwise.} \end{cases} \qquad (2.39)$$

φ-program c, spelled out just below, outputs a φ-program which computes the composition of two input ζ-programs. It is employed by φ-program w' further below.

For each p, q,

$$\varphi_{\varphi_c(\langle p, q \rangle)}(x) = \zeta_p(\zeta_q(x)). \qquad (2.40)$$

φ-program w', spelled out next, rewrites each output of φ-program c above so that the output of w' computes the same partial function as this output of c, but has combinatorially useful numeric properties regarding evenness.

For each p, q,

$$\varphi_{w'}(\langle p, q \rangle) = \begin{cases} \text{the first value } v, \text{ if any, found by iteratively applying} \\ \quad \text{pad to } \varphi_c(\langle p, q \rangle) \text{ such that: } v \neq w, \ v \text{ is even,} \\ \quad v > \langle p, q \rangle \ (\text{hence, } v > p, q \text{ by Lemma 1.1}), \\ \quad (\forall x < \langle p, q \rangle)[\varphi_{w'}(x){\downarrow}], \\ \quad [\text{if } \langle p, q \rangle > 0, \text{ then } v > \varphi_{w'}(\langle p, q \rangle - 1) \text{ which } {\downarrow}], \\ \quad \text{and, for } \langle r, s \rangle = v, \\ \quad (\neg \exists x < \langle p, q \rangle)[\varphi_{w'}(x) = r \ \lor \ \varphi_{w'}(x) = s]; \\ {\uparrow}, \text{if no such } v \text{ exists.} \end{cases} \qquad (2.41)$$

As we'll see, φ-program w, spelled out next, is such that $\varphi_w = \zeta_w$ (Claim 2.29 below), *and* it computes the $m = 1$ case control structure of composition from [28, Lemma 3.10] (see Footnote 8 above) — but for the ζ-system (Claim 2.32). This w rewrites the output of w' above so that the resultant ζ-system composition will have strong associativity properties *at the ζ-program code number level*: let $\text{comp}_1 = \zeta_w$; then for all ζ-programs a, b, c, the ζ-program number $\text{comp}_1(\text{comp}_1(a, b), c)$ *will be the same ζ-program number as* $\text{comp}_1(a, \text{comp}_1(b, c))$. This strong property is, in effect, further developed in Claims 2.34 through 2.37, these claims put limits on the nature of *un*equal ζ-program numbers *in* $\rho(\text{comp}_1)$ — so they cannot interfere with the unprovability part of Theorem 2.21, and, then, they are used in proving the difficult to prove Claim 2.40. This latter claim provides most of the desired unprovability — with Claim 2.41 finishing it off.

For each p, q,

$$\varphi_w(\langle p, q \rangle) = \begin{cases} \varphi_w(\langle \varphi_w(\langle p, r \rangle), s \rangle), & \text{if } (\forall \langle r, s \rangle < q)[\varphi_w(\langle r, s \rangle){\downarrow}] \ \land \\ & \quad (\exists \langle r, s \rangle < q)[\varphi_w(\langle r, s \rangle) = q], \\ & \quad \text{then select minimum such } s \\ & \quad \text{and, then, select the minimum } r \\ & \quad \text{corresponding to this } s; \\ \varphi_{w'}(\langle p, q \rangle), & \text{if } (\forall \langle r, s \rangle < q)[\varphi_w(\langle r, s \rangle){\downarrow} \neq q]; \\ {\uparrow}, & \text{otherwise.} \end{cases} \qquad (2.42)$$

Claim 2.22. PA proves $\ll \varphi_c$ is total $\gg$.

Proof of Claim 2.22.   In general proofs of recursion theorems, especially including Lemma 2.20 above, are so simple that their proofs can be carried out in **PA** (albeit tediously).
□ CLAIM 2.22

Claim 2.23. For all $x > 0$, if $\varphi_{w'}(x-1){\downarrow} \ \wedge \ \varphi_{w'}(x){\downarrow}$, then $\varphi_{w'}(x-1) < \varphi_{w'}(x)$.

Proof of Claim 2.23.   This follows directly from (2.41).   □ CLAIM 2.23

Claim 2.24. $\varphi_{w'}$ is total, and **PA** proves that.

Proof of Claim 2.24.   Assume for induction that for arbitrarily-fixed $\langle p, q \rangle$, for all $x < \langle p, q \rangle$, we have $\varphi_{w'}(x){\downarrow}$. By Claim 2.22, $\varphi_c(\langle p, q \rangle){\downarrow}$. By Claim 2.19 the output of pad is always even and greater than the input. Thus, repeatedly applying pad to $\varphi_c(\langle p, q \rangle)$ will result in v having an even value, $v \neq w$, $v > p$, $v > q$, and $v > \langle p, q \rangle$. Furthermore, by the induction assumption, [if $\langle p, q \rangle > 0$, then $\varphi_{w'}(\langle p, q \rangle - 1){\downarrow}$], and padding $\varphi_c(\langle p, q \rangle)$ until $v > \varphi_{w'}(\langle p, q \rangle - 1)$ is certainly possible. Lastly, by the induction assumption, checking whether for $(\langle r, s \rangle = v)$ there does or does not $(\exists x < \langle p, q \rangle)[\varphi_{w'}(x) = r \ \vee \ \varphi_{w'}(x) = s]$ is algorithmically testable.

   Thus, by induction, $\varphi_{w'}$ is total. The above inductive proof is accessible to **PA**, thus **PA** proves it.   □ CLAIM 2.24

Claim 2.25. $\rho(\varphi_w) \subseteq \rho(\varphi_{w'})$. Furthermore, **PA** proves this.

Proof of Claim 2.25.   For each input x, exactly one of the three clauses of (2.42) must hold. If the first clause holds, then whatever value $\varphi_w(x)$ has must be a value that was in the range of φ_w on some other input; by implicit application of Lemma 1.1, this recursion must bottom out at some value, and that value must come from some other clause. If the second clause holds, then $\varphi_w(x) = \varphi_{w'}(x)$, and thus is a value in the range of $\varphi_{w'}$. Lastly, if the third clause holds, then $\varphi_w(x){\uparrow}$, and no value is added to the range of φ_w. Thus, all values in the range of φ_w have some x such that the second clause of (2.42) holds for $\varphi_w(x)$, and therefore, the range of φ_w is a (potentially proper) subset of the range of $\varphi_{w'}$. **PA** can handle the preceding argument.   □ CLAIM 2.25

Claim 2.26. If $q \in \rho(\varphi_w)$, then $(\exists \langle r, s \rangle < q)[\varphi_w(\langle r, s \rangle){\downarrow} = q]$. Furthermore, **PA** proves this.

Proof of Claim 2.26.   Fix arbitrary q, assume that $q \in \rho(\varphi_w)$. Per the *proof* of Claim 2.25 and the second clause of (2.42), there then exists x such that $\varphi_w(x) = q = \varphi_{w'}(x)$. By Claims 2.23 and 2.24, for x such that $\varphi_{w'}(x) = q$, it must be the case that $x \leq q$. By (2.41), $x \neq q$. Thus, $(\exists x < q)[\varphi_w(x) = q]$; as this follows from the assumption that $q \in \rho(\varphi_w)$, it therefore follows that if $q \in \rho(\varphi_w)$, $(\exists x < q)[\varphi_w(x) = q]$. This proves the claim except for the part about **PA**. Lastly, **PA** can handle the just prior reasoning.   □ CLAIM 2.26

Claim 2.27. For all p, if $(\exists \langle r, s \rangle < p)[\varphi_w(\langle r, s \rangle) = p]$, then the minimum such s is *not* in the range of φ_w.

Proof of Claim 2.27.   Assume by way of contradiction otherwise. Fix the least counterexample p to the claim, and fix the minimum s corresponding to that p. By the assumption by way of contradiction, $(\exists \langle r', s' \rangle)[\varphi_w(\langle r', s' \rangle) = s]$. Fix the minimum such s' and the corresponding r' such that $\langle r', s' \rangle$ is minimum. By Claim 2.26, $\langle r', s' \rangle < s$. By the first clause of (2.42), $\varphi_w(\langle r, s \rangle) = \varphi_w(\langle \varphi_w(\langle r, r' \rangle), s' \rangle)$. Since s is minimum as indicated above and $s' < s$ (by Lemma 1.1), it follows that $\langle \varphi_w(\langle r, r' \rangle), s' \rangle \geq p$.

Case one: The first clause of (2.42) holds for input $\langle \varphi_w(\langle r, r' \rangle), s' \rangle$. Then s' is also a counterexample to Claim 2.27, which, since $s' < p$, contradicts p's minimality.

Case two: The second clause of (2.42) holds for input $\langle \varphi_w(\langle r, r' \rangle), s' \rangle$. Then

$$\varphi_w(\langle \varphi_w(\langle r, r' \rangle), s' \rangle) = \varphi_{w'}(\langle \varphi_w(\langle r, r' \rangle), s' \rangle).$$

Then, by (2.41), it follows that

$$\varphi_w(\langle \varphi_w(\langle r, r' \rangle), s' \rangle) > \langle \varphi_w(\langle r, r' \rangle), s' \rangle,$$

and thus, by substitution, it follows that $p > \langle \varphi_w(\langle r, r' \rangle), s' \rangle$; this contradicts

$$\langle \varphi_w(\langle r, r' \rangle), s' \rangle \geq p.$$

Case three: The third clause of (2.42) holds for input $\langle \varphi_w(\langle r, r' \rangle), s' \rangle$; this contradicts $\varphi_w(\langle r, s \rangle)\downarrow = p$.

All cases lead to a contradiction; therefore, the claim holds.   □ CLAIM 2.27

Claim 2.28. φ_w is total; furthermore, **PA** proves this.

Proof of Claim 2.28.   Assume for induction that for arbitrarily-fixed $\langle p, q \rangle$, for all $x < \langle p, q \rangle$, $\varphi_w(x)\downarrow$. It follows from inequalities about pairing (Lemma 1.1) and the induction assumption that $(\forall \langle r, s \rangle < q)[\varphi_w(\langle r, s \rangle)\downarrow]$. Thus, for input $\langle p, q \rangle$, the first clause of (2.42) holds if $(\exists \langle r, s \rangle < q)[q = \varphi_w(\langle r, s \rangle)]$ and the second clause of (2.42) holds if $(\neg \exists \langle r, s \rangle < q)[q = \varphi_w(\langle r, s \rangle)]$; thus, the third clause of (2.42) does not hold for input $\langle p, q \rangle$.

If the second clause of (2.42) holds for input $\langle p, q \rangle$, then by Claim 2.24, $\varphi_w(\langle p, q \rangle)\downarrow$.

If the first clause of (2.42) holds for input $\langle p, q \rangle$, then

$$\varphi_w(\langle p, q \rangle)\downarrow \Leftrightarrow \varphi_w(\langle \varphi_w(\langle p, r \rangle), s \rangle)\downarrow,$$

where $\langle r, s \rangle < q$, $q = \varphi_w(\langle r, s \rangle)$, s is minimum such that that is the case, and r is minimum corresponding to s. It follows that both $r < q$ and $s < q$. From the fact that $r < q$, it follows that $\varphi_w(\langle p, r \rangle)\downarrow$ by the inductive assumption. As $s < q$, and $s \notin \rho(\varphi_w)$ (by Claim 2.27), it then follows from the induction hypothesis that $(\forall \langle r', s' \rangle < s)[\varphi_w(\langle r', s' \rangle)\downarrow \neq s]$; thus, the second clause of (2.42) holds for $\varphi_w(\langle \varphi_w(\langle p, r \rangle), s \rangle)$, which is therefore defined by Claim 2.24.

From the inductive assumption, then, $\varphi_w(\langle p, q \rangle)\downarrow$. By induction, φ_w is total. **PA** can prove this as well.   □ CLAIM 2.28

Claim 2.29. $\zeta_w = \varphi_w$, and **PA** proves this.

Proof of Claim 2.29.   The equality follows immediately from the first disjunct of the disjunction in the first clause of (2.39); **PA** can prove this as well.   □ CLAIM 2.29

Claim 2.30. $(\forall p, q)[\varphi_{\varphi_{w'}(\langle p, q \rangle)} = \zeta_p \circ \zeta_q]$; furthermore, **PA** proves this.

Proof of Claim 2.30.   The claim follows from (2.41, 2.40) and Claim 2.24 above.

□ Claim 2.30

Claim 2.31. $(\forall p, q)[\varphi_{\varphi_{w'}(\langle p,q\rangle)} = \varphi_{\varphi_{w}(\langle p,q\rangle)}]$; furthermore, **PA** proves this.

Proof of Claim 2.31.   Assume for induction that for arbitrary p, q, for all $q' < q$, for all p', $\varphi_{\varphi_{w'}(\langle p',q'\rangle)} = \varphi_{\varphi_{w}(\langle p',q'\rangle)}$.

  Case one: The first clause of (2.42) holds for $\langle p, q\rangle$. Then, there exists $(\langle r, s\rangle < q)[q = \varphi_w(\langle r, s\rangle)]$; fix r and s in accordance with that clause. Then fix r and s in accordance with that clause; thus, $\langle r, s\rangle < q = \varphi_w(\langle r, s\rangle)$. It follows that $\varphi_w(\langle p, q\rangle) = \varphi_w(\langle \varphi_w(\langle p, r\rangle), s\rangle)$. Since $\langle r, s\rangle < q$, it follows by Lemma 1.1 that $r < q$ and $s < q$; thus, by the induction assumption, $\varphi_{\varphi_w(\langle \varphi_w(\langle p,r\rangle),s\rangle)} = \varphi_{\varphi_{w'}(\langle \varphi_w(\langle p,r\rangle),s\rangle)}$ and $\varphi_{\varphi_w(\langle p,r\rangle)} = \varphi_{\varphi_{w'}(\langle p,r\rangle)}$. By Claim 2.30, $\varphi_{\varphi_{w'}(\langle p,r\rangle)} = \zeta_p \circ \zeta_r$, and $\varphi_{\varphi_{w'}(\langle \varphi_w(\langle p,r\rangle),s\rangle)} = \zeta_{\varphi_w(\langle p,r\rangle)} \circ \zeta_s$. By Claim 2.26 and the second disjunct of the first clause of (2.39), $\zeta_{\varphi_w(\langle p,r\rangle)} = \varphi_{\varphi_w(\langle p,r\rangle)}$. By repeated substitutions, it follows that $\varphi_{\varphi_w(\langle p,q\rangle)} = \varphi_{\varphi_w(\langle \varphi_w(\langle p,r\rangle),s\rangle)} = \varphi_{\varphi_{w'}(\langle \varphi_w(\langle p,r\rangle),s\rangle)} = \zeta_{\varphi_w(\langle p,r\rangle)} \circ \zeta_s = \varphi_{\varphi_w(\langle p,r\rangle)} \circ \zeta_s = \varphi_{\varphi_{w'}(\langle p,r\rangle)} \circ \zeta_s = \zeta_p \circ \zeta_r \circ \zeta_s$. It follows from Claim 2.30 that $\varphi_{\varphi_{w'}(\langle p,q\rangle)} = \zeta_p \circ \zeta_q$. As $q = \varphi_w(\langle r, s\rangle)$ and is thus in $\rho(\varphi_w)$, it follows from (2.39) that $\zeta_q = \varphi_q$, and by substitution it follows that $\varphi_q = \varphi_{\varphi_w(\langle r,s\rangle)}$. From the induction assumption, it follows that $\varphi_{\varphi_w(\langle r,s\rangle)} = \varphi_{\varphi_{w'}(\langle r,s\rangle)}$, and then from Claim 2.30 it follows that $\varphi_{\varphi_{w'}(\langle r,s\rangle)} = \zeta_r \circ \zeta_s$; and thus it follows that $\varphi_{\varphi_{w'}(\langle p,q\rangle)} = \zeta_p \circ \zeta_r \circ \zeta_s$. It then immediately follows that $\varphi_{\varphi_{w'}(\langle p,q\rangle)} = \varphi_{\varphi_w(\langle p,q\rangle)}$.

  Case two: The second clause of (2.42) holds for $\langle p, q\rangle$. Then, it immediately follows from that clause that $\varphi_{\varphi_{w'}(\langle p,q\rangle)} = \varphi_{\varphi_w(\langle p,q\rangle)}$.

  Case three: The third clause of (2.42) holds for $\langle p, q\rangle$. By Claim 2.28, this case is impossible.

  In all possible cases, $\varphi_{\varphi_{w'}(\langle p,q\rangle)} = \varphi_{\varphi_w(\langle p,q\rangle)}$; therefore, by induction,

$$(\forall p, q)[\varphi_{\varphi_{w'}(\langle p,q\rangle)} = \varphi_{\varphi_w(\langle p,q\rangle)}].$$

PA can handle the above reasoning, and thus the claim follows.   □ Claim 2.31

Claim 2.32. PA $\vdash$ $\ll w$ is a witness to composition in $\zeta \gg$.

Proof of Claim 2.32.   This claim follows directly from Claims 2.28, 2.29, 2.30, and 2.31.

□ Claim 2.32

Claim 2.33. All values in the range of φ_w are even.

Proof of Claim 2.33.   By (2.41), the range of $\varphi_{w'}$ consists of only even numbers. The claim follows from that fact and Claim 2.25.   □ Claim 2.33

  In the remainder of this proof, we will need the ability to treat programs in the range of φ_w — which are now known to be compositions of other programs — as a sequence of such compositions. Furthermore, we need the ability to take potentially-long such sequences and pull off a single element from the front or back, and recompose the rest of the sequence to get another ζ-program. In order to do this, we introduce chain and unchain; chain takes the code number of a ζ-program and outputs a *sequence* of ζ-programs such that if the sequence is composed in order, the computed partial function is equivalent to the partial function computed by the input ζ-program; while unchain takes a non-empty sequence of

ζ-programs and outputs a single ζ-program which is equivalent to composing the sequence in order.

$$
\text{chain}(p) \stackrel{\text{def}}{=}
\begin{cases}
\text{chain}(r), s, & \text{if } (\exists \langle r, s \rangle < p)[\varphi_w(\langle r, s \rangle) = p], \\
& \qquad \text{then select the minimum such } s \\
& \qquad \text{and, then, select the minimum such } r \\
& \qquad \text{corresponding to this selected s;} \\
p, & \qquad \text{otherwise.}
\end{cases}
\tag{2.43}
$$

When $\mathbf{v}$ is a sequence of more than one element, the terminology employed just below, all-but-last($\mathbf{v}$) and last($\mathbf{v}$), is self-explanatory.

$$
\text{unchain}(\mathbf{v}) \stackrel{\text{def}}{=}
\begin{cases}
\text{that element,} & \text{if } \mathbf{v} \text{ is a sequence} \\
& \qquad \text{of one element;} \\
\varphi_w(\langle \text{unchain}(\text{all-but-last}(\mathbf{v})), \text{last}(\mathbf{v}) \rangle), & \text{otherwise.}
\end{cases}
\tag{2.44}
$$

Claim 2.34. For $p_0, p_1, \ldots, p_n = \text{chain}(x)$, $\zeta_x = \zeta_{p_0} \circ \zeta_{p_1} \circ \ldots \circ \zeta_{p_n}$.

Proof of Claim 2.34. Let x be an arbitrary value, and assume for induction that for all $y < x$, the claim holds for chain(y).

Case one: chain(x) $= x$. It immediately follows that the claim holds for chain(x) in this case.

Case two: chain(x) $=$ chain(p), q. Then $\varphi_w(\langle p, q \rangle) = x$, $x > p$, and $x > q$.

Then p and q are such that the second clause of (2.42) holds on input $\langle p, q \rangle$, per Claim 2.27. Then $x = \varphi_{w'}(\langle p, q \rangle)$, from which it follows that $\varphi_x = \zeta_p \circ \zeta_q$. Furthermore, since x is in the range of φ_w, $\zeta_x = \varphi_x$ by the second disjunct of the first clause of (2.39). From the just previous reasoning and the induction assumption, the claim also holds for chain(x) in this case.

By the induction, the claim holds for all x. $\qquad\qquad\square$ CLAIM 2.34

Claim 2.35. For any p, none of the elements of chain(p) are in $\rho(\varphi_w)$.

Proof of Claim 2.35. Assume by induction that for all $p' < p$, the elements of chain(p') are each not in $\rho(\varphi_w)$.

Case one: The first clause of (2.43) holds for p. Let r, s be per that clause; by Lemma 1.1, it follows that $r < p$. Therefore, by the induction assumption, the elements of chain(r) are each not in $\rho(\varphi_w)$. By Claim 2.27, s is also not in $\rho(\varphi_w)$; thus, each element of chain(p) is not in $\rho(\varphi_w)$.

Case two: The second clause of (2.43) holds for p. By Claim 2.26, if p is in the range of φ_w, then $(\exists \langle r, s \rangle < p)[\varphi_w(\langle r, s \rangle) \downarrow = p]$; thus, $p \notin \rho(\varphi_w)$, and the only element of chain(p) is not in $\rho(\varphi_w)$.

By induction, the claim holds for all p. $\qquad\qquad\square$ CLAIM 2.35

Claim 2.36. For all x, unchain(chain(x)) $= x$.

Proof of Claim 2.36. If chain(x) is a sequence of one element, unchain(chain(x)) is just x per the first clause of (2.44). Let n be some number greater than 1. Assume for induction that for all x such that chain(x) is a sequence of length no more than $n-1$, unchain(chain(x)) = x. Then let x be such that chain(x) is a sequence of length n. That sequence, for some r, s, dependent on x, is chain(r), s. Furthermore, by the first clause of (2.43), $x = \varphi_w(\langle r, s \rangle)$. In such a sequence, for unchain(chain(r), s), the second clause of (2.44) holds, and outputs the result of $\varphi_w(\langle \text{unchain}(\text{chain}(r)), s \rangle)$ — by inductive assumption, it follows that unchain(chain(s)) is s, from which it follows that the output of unchain(chain(x)) is $\varphi_w(\langle r, s \rangle)$ — which is already known to be x. Thus, the claim follows by induction on the length of chain(x). □ CLAIM 2.36

Claim 2.37. For any sequence $p_0, p_1, .., p_n$ of length at least two such that there exists p such that chain(p) = $p_0, p_1, \ldots, p_n$, $\zeta_p = \zeta_{p_0} \circ \zeta_{\text{unchain}(p_1,...,p_n)}$.

Proof of Claim 2.37. Let $p' = \text{unchain}(p_1, .., p_n)$. By (2.43, 2.44) above and by Claim 2.35, it follows that chain(p') = $p_1, .., p_n$. Therefore, by Claim 2.34, $\zeta_{p'} = \zeta_{p_1} \circ \ldots \circ \zeta_{p_n}$. Thus, $\zeta_{p_0} \circ \zeta_{\text{unchain}(p_1,...,p_n)} = \zeta_{p_0} \circ \zeta_{p_1} \circ \ldots \circ \zeta_{p_n}$; by Claim 2.34, this is exactly ζ_p. □ CLAIM 2.37

Claim 2.38. For all x such that [x is odd *or x in $\rho(\varphi_w)$*], $\zeta_w(x) > x$, and, if x is in $\rho(\varphi_w)$, then there does not exist $y \neq x$ such that $\zeta_w(y) = \zeta_w(x)$.

Proof of Claim 2.38. Suppose x is odd. For $\langle p, q \rangle = x$, q must be odd — per Lemma 1.1. As such, because φ_w only outputs even values, q cannot be in the range of φ_w, and thus $\varphi_w(x) = \varphi_{w'}(x)$ by (2.42). By (2.41), $\varphi_{w'}(x) > x$. By Claim 2.29, $\zeta_w = \varphi_w$, and thus $\zeta_w(x) > x$.

Suppose $x \in \rho(\varphi_w)$. For $\langle p, q \rangle = x$, by Lemma 1.1, $p \leq x$ and $q \leq x$. By (2.41) and Claim 2.24, there does not exist $y < x$ such that $\varphi_{w'}(y) =$ either p or q. By (2.41) and Claim 2.24, $\varphi_{w'}(0) > 0$; from this and Claims 2.23 and 2.24, it follows that there does not exist y such that $\varphi_{w'}(y) =$ either p or q. Therefore, by Claim 2.25, $q \notin \rho(\varphi_{w'})$, from which it follows, by (2.42), $\varphi_w(x) = \varphi_{w'}(x)$. As $p \notin \rho(\varphi_{w'})$ and thus $p \notin \rho(\varphi_w)$, it follows by pairing being 1-1 that there does not exist any p', r, s such that $\langle \varphi_w(\langle p', r \rangle), s \rangle = x$.

Therefore, there exists no y such that both the first clause of (2.42) holds on input y and $\varphi_w(y) = \varphi_w(x)$. By Claims 2.23 and 2.24, there exists no $y \neq x$ such that $\varphi_{w'}(x) = \varphi_{w'}(y)$; from this and the fact that $\varphi_w(x) = \varphi_{w'}(x)$, it follows that for $y \neq x$, if the second clause of (2.42) holds on input y, $\varphi_w(y) \neq \varphi_w(x)$. Therefore, there does not exist $y \neq x$ such that [$\varphi_w(x) = \varphi_w(y)$]. Furthermore, from (2.41) and the fact that $\varphi_w(x) = \varphi_{w'}(x)$, it follows that $\varphi_w(x) > x$.

Thus, the claim follows. □ CLAIM 2.38

Claim 2.39. If, for some x_0, $\mathbf{T} \vdash_{x_0} \ll (\exists r, t \neq r)[\zeta_r = \zeta_t] \gg$, then $(\forall p)(\forall x \geq x_0 \mid x$ is odd $\lor x \in \rho(\varphi_w))[\zeta_p(x)\downarrow \land [\zeta_p(x)$ odd $\lor \zeta_p(x) \in \rho(\varphi_w)] \land \zeta_p(x) > x]$.

Proof of Claim 2.39. Assume by way of contradiction that the claim does not hold. Fix least p and least corresponding $x \geq x_0$ such that [x is odd $\lor x \in \rho(\varphi_w)$] so that either $\zeta_p(x)\uparrow$, or $\zeta_p(x)$ even and not in the range of φ_w, or $\zeta_p(x) \leq x$.

Case one: p is w. Then, by Claims 2.28, 2.38 and 2.29, a contradiction follows immediately.

Case two: p is not w and not in $\rho(\varphi_w)$. Then, by (2.39), $\zeta_p(x) = \mathrm{prime}(p+1)^x$, which is an odd value greater than x. A contradiction follows immediately.

Case three: p is in $\rho(\varphi_w)$. Let $p_0, \ldots, p_n$ be chain(p) and let $p' = \mathrm{unchain}(p_0, \ldots, p_{n-1})$.[18] By Claim 2.35, $p_n \notin \rho(\varphi_w)$. By Claim 2.36 and (2.43), $\varphi_w(\langle p', p_n \rangle) = p$. From (2.43) we have that $p', p_n, \langle p', p_n \rangle < p$. Then by Claims 2.30 and 2.31 and (2.39), it follows that $\zeta_p = \zeta_{p'} \circ \zeta_{p_n}$. By the second clause of (2.42), $\varphi_w(\langle p', p_n \rangle) = \varphi_{w'}(\langle p', p_n \rangle)$, thus $p > p'$ by (2.41).

> Subcase one: p_n is not w. Then, $\zeta_{p_n}(x) = \mathrm{prime}(p_n+1)^x$, which is both odd and greater than x; thus, by the assumption that p is the least value such that the claim does not hold, $\zeta_{p'}(\mathrm{prime}(p_n+1)^x)$ is greater than $\mathrm{prime}(p_n+1)^x$, which is greater than x. $\zeta_{p'}(\mathrm{prime}(p_n+1)^x)$ is also either odd or in $\rho(\varphi_w)$; as $\zeta_p(x)$ is that value, contradiction follows immediately.
> Subcase two: p_n is w. Then, by (2.39), $\zeta_{p_n}(x)$ is in the range of φ_w, moreover, by Claim 2.38, it is $> x$. By the assumption that p is the least value such that the claim does not hold, $\zeta_{p'}(\zeta_{p_n}(x))$ is greater than $\zeta_{p_n}(x)$, which, as previously shown, is greater than x. $\zeta_{p'}(\zeta_{p_n}(x))$ is also either odd or in $\rho(\varphi_w)$; as that value is equal to $\zeta_p(x)$ — a contradiction follows immediately.

In all cases, a contradiction follows; thus, the claim holds.   □ CLAIM 2.39

Claim 2.40. If, for some x_0, $\mathbf{T} \vdash_{x_0} \ll (\exists r, t \neq r)[\zeta_r = \zeta_t] \gg$, then $(\forall r', t' \neq r')(\overset{\infty}{\exists} x)[\zeta_{r'}(x) \neq \zeta_{t'}(x)]$.

Proof of Claim 2.40. Assume by way of contradiction otherwise. Then by the assumption that $\mathbf{T}$ does not prove false things, $(\exists p', q' \neq p')[\zeta_{p'} = \zeta_{q'}]$. Thus, $(\exists p', q' \neq p')(\forall x \geq 0)[\zeta_{p'}(x) = \zeta_{q'}(x)]$. Let p be the least number such that chain(p) is of minimum length so that there are q and x'' with $q \neq p$ and x'' so that $(\forall x \geq x'')[\zeta_p(x) = \zeta_q(x)]$; then let q be the least corresponding q. Let x_0 be the least number such that $\mathbf{T} \vdash_{x_0} \ll (\exists r, t \neq r)[\zeta_r = \zeta_t] \gg$. Let x' be the least value such that each of the following hold: x' is in the range of φ_w, $x' \geq x_0$, and $x' \geq x''$. By Claim 2.38, there is a value in the range of φ_w greater than any fixed odd number, and thus such an x' must exist.

Case 1: $p = w$.

> Subcase 1.1: $q = w$. This immediately contradicts our assumption that $p \neq q$.
> Subcase 1.2: $q \neq w$ and $q \notin \rho(\varphi_w)$. Then, by (2.39), $\zeta_q(x') = \mathrm{prime}(q+1)^{x'}$, which is an odd value, and by Claim 2.33, $\zeta_w(x')$ is even; a contradiction follows immediately.
> Subcase 1.3: $q \in \rho(\varphi_w)$. Let $q_0, \ldots, q_n = \mathrm{chain}(q)$. By Claim 2.35, q_0 is not in $\rho(\varphi_w)$. Let $q' = \mathrm{unchain}(q_1, \ldots, q_n)$. By Claim 2.37, $\zeta_q = \zeta_{q_0} \circ \zeta_{q'}$. By Claim 2.39, $\zeta_{q'}(x') > x'$ and $\zeta_{q'}(x')$ is either odd or in the range of φ_w.
>> Sub-subcase 1.3.1: $q_0 \neq w$. Then by (2.39), $\zeta_q(x) = \zeta_{q_0}(\zeta_{q'}(x')) = \mathrm{prime}(q_0+1)^{\zeta_{q'}(x')}$, which is an odd value, while, by Claim 2.33, $\zeta_w(x')$ is even; therefore, $\zeta_q(x') \neq \zeta_p(x')$; a contradiction.

_____

[18] By (2.43), for all x in the range of φ_w, the length of chain(x) ≥ 2.

Sub-subcase 1.3.2: $q_0 = w$. Therefore, since $\zeta_{q_0}(\zeta_{q'}(x')) = \zeta_q(x') = \zeta_p(x')$, $\zeta_{q_0}(\zeta_{q'}(x')) = \zeta_w(\zeta_{q'}(x'))$, and $\zeta_p(x') = \zeta_w(x')$, it follows that $\zeta_q(x') = \zeta_w(x')$. However, $\zeta_{q'}(x') > x'$, and by Claim 2.38, it follows from the fact that $\zeta_w(x') = \zeta_w(\zeta_{q'}(x'))$ that $x' = \zeta_{q'}(x')$ — a contradiction.

Case 2: $p \neq w$ and $p \notin \rho(\varphi_w)$.

Subcase 2.1: $q = w$. The same argument holds as for Case 1.2, interchanging p and q.

Subcase 2.2: $q \neq w$ and $q \notin \rho(\varphi_w)$. Then by (2.39), $\zeta_p(x') = \mathrm{prime}(p + 1)^{x'}$ and $\zeta_q(x') = \mathrm{prime}(q + 1)^{x'}$, thus $\mathrm{prime}(p + 1)^{x'} = \mathrm{prime}(q + 1)^{x'}$, which implies that $p = q$, a contradiction.

Subcase 2.3: $q \in \rho(\varphi_w)$. Let $q_0, \ldots, q_n = \mathrm{chain}(q)$; then let $q' = \mathrm{unchain}(q_1, \ldots, q_n)$. By Claim 2.37, $\zeta_q = \zeta_{q_0} \circ \zeta_{q'}$. By Claim 2.39, for all $x \geq x'$, $\zeta_{p'}(x)$ is greater than x' and either odd or in the range of φ_w. By Claim 2.35, $q_0 \notin \rho(\varphi_w^{TM})$.

Sub-subcase 2.3.1: $q_0 = w$. Then $\zeta_q(x') = \zeta_w(\zeta_{q'}(x'))$, which is even per Claim 2.33. By the second clause of (2.39), $\zeta_p(x')$ is odd; a contradiction to $\zeta_p(x') = \zeta_q(x')$ follows.

Sub-subcase 2.3.2: $q_0 = p$. Then, as $\zeta_q(x') = \zeta_{q_0}(\zeta_{q'}(x'))$, it follows that $\zeta_q(x') = \mathrm{prime}(p+1)^{\zeta_{q'}(x')}$. Likewise, $\zeta_p(x') = \mathrm{prime}(p + 1)^{x'}$. Therefore, $\zeta_{q'}(x') = x'$, which is a contradiction to Claim 2.39.

Sub-subcase 2.3.3: $q_0 \neq w$ and $q_0 \neq p$. Then, by (2.39) and the fact that $\zeta_q(x') = \zeta_{q_0}(\zeta_{q'}(x'))$, $\zeta_q(x')$ is a power of $\mathrm{prime}(q_0 + 1)$ and $\zeta_p(x')$ is $\mathrm{prime}(p + 1)^{x'}$; thus $\zeta_q(x') \neq \zeta_p(x')$, a contradiction.

Case 3: $p \in \rho(\varphi_w)$.

Subcase 3.1: $q = w$. The same argument holds as for Case 1.3, interchanging p and q.

Subcase 3.2: $q \neq w$ and $q \notin \rho(\varphi_w)$. The same argument holds as for case two subcase three, interchanging p and q.

Subcase 3.3: $q \in \rho(\varphi_w)$. Let $p_0, \ldots, p_m = \mathrm{chain}(p)$, let $q_0, \ldots, q_n = \mathrm{chain}(q)$. Let $p' = \mathrm{unchain}(p_1, \ldots, p_m)$ and $q' = \mathrm{unchain}(q_1, \ldots, q_n)$. By Claim 2.37, $\zeta_q = \zeta_{q_0} \circ \zeta_{q'}$, and $\zeta_p = \zeta_{p_0} \circ \zeta_{p'}$. By Claim 2.39, for all $x \geq x'$, $\zeta_{p'}(x)$ and $\zeta_{q'}(x)$ are both greater than x' and either odd or in the range of φ_w.

Sub-subcase 3.3.1: $p_0 = q_0 \neq w$. Then, by (2.39), for each $x \geq x'$, $\zeta_{p_0}(x) = \zeta_{q_0}(x) = \mathrm{prime}(p_0 + 1)^x$. Thus, for each $x \geq x'$, for each $y \geq x' \wedge y \neq x$, $\zeta_{p_0}(x) \neq \zeta_{q_0}(y)$. As it is the case that for all $x \geq x'$, $\zeta_{p'}(x)$ and $\zeta_{q'}(x)$ are both greater than x', it then follows from the previous equalities and inequalities about p_0 and q_0, as well as the facts that $\zeta_q = \zeta_{q_0} \circ \zeta_{q'}$, and $\zeta_p = \zeta_{p_0} \circ \zeta_{p'}$, that $(\forall x \geq x')[\zeta_{p'}(x) = \zeta_{q'}(x)]$. But since $\mathrm{chain}(p')$ is shorter than $\mathrm{chain}(p)$, this contradicts our assumption that p had the

shortest chain such that there exists $q \neq p$ and x'' so that $(\forall x \geq x'')[\zeta_p(x) = \zeta_q(x)]$.

Sub-subcase 3.3.2: $p_0 = q_0 = w$. By Claim 2.38 and the fact that for all $x \geq x'$, $\zeta_{p'}(x)$ and $\zeta_{q'}(x)$ are both greater than x' and either odd or in the range of φ_w, it follows from reasoning similar to that of sub-subcase 3.3.1, that for all $x \geq x'$, $\zeta_{p'}(x) = \zeta_{q'}(x)$, which again contradicts our assumption that p had the shortest chain such that there exists $q \neq p$ and x'' so that $(\forall x \geq x'')[\zeta_p(x) = \zeta_q(x)]$.

Sub-subcase 3.3.3: $p_0 \neq q_0$, and neither p_0 nor q_0 is w. Then, by (2.39) and the facts that $\zeta_p = \zeta_{p_0} \circ \zeta_{p'}$ and $\zeta_q = \zeta_{q_0} \circ \zeta_{q'}$, it follows that $\zeta_p(x')$ is a power of $\mathrm{prime}(p_0 + 1)$, and $\zeta_q(x')$ is a power of $\mathrm{prime}(q_0+1)$; thus, $\zeta_p(x') \neq \zeta_q(x')$, a contradiction.

Sub-subcase 3.3.4: $p_0 \neq q_0$, and $p_0 = w$. Then, by Claim 2.33 and the fact that $\zeta_p = \zeta_{p_0} \circ \zeta_{p'}$, it follows that $\zeta_p(x')$ is even. By Claim 2.35, $q_0 \notin \rho(\varphi_w)$ and $q_0 \neq w$; thus by (2.39), the fact that $\zeta_q = \zeta_{q_0} \circ \zeta_{q'}$, and the fact that $\zeta_{q'}(x') \geq x'$, it follows that $\zeta_q(x')$ is odd; a contradiction to the assumption that $\zeta_p(x') = \zeta_q(x')$.

Sub-subcase 3.3.5: $p_0 \neq q_0$, and $q_0 = w$. The same reasoning holds as for sub-subcase 3.3.4, with p and q interchanged.

Every case leads to a contradiction; thus the assumption is false, and the claim follows. $\square$ CLAIM 2.40

Claim 2.41. $\mathbf{T} \nvdash \ll (\exists r, t \neq r)[\zeta_r = \zeta_t] \gg$.

Proof of Claim 2.41. By Claim 2.40, if $\mathbf{T} \vdash \ll (\exists r, t \neq r)[\zeta_r = \zeta_t] \gg$, then $(\forall r', t' \neq r')(\overset{\infty}{\exists} x)[\zeta_{r'}(x) \neq \zeta_{t'}(x)]$. Thus, there does not exist $r, t \neq r$ such that $\zeta_r = \zeta_t$. Since $\mathbf{T}$ does not prove false things, the claim follows. $\square$ CLAIM 2.41

Claim 2.42. ζ is acceptable.

Proof of Claim 2.42. By Claim 2.41, the third disjunct of the first clause of (2.39) is true for all x. Thus, for all p and x, $\zeta_p(x) = \varphi_p(x)$; that is, $\zeta = \varphi$, which is known to be acceptable. $\square$ CLAIM 2.42

The theorem follows immediately from Claims 2.32, 2.41, and 2.42 $\square$ THEOREM 2.21

References

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

[2] 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.

[3] 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).

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

[5] 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.

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

[7] 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.

[8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

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

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

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

[12] 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.

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

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

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

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

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

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

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

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

[21] 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.

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

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

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

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

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

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

[28] 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.

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