An Anomaly in Diagonalization

T. Parent (Nazarbayev University)
nontology@gmail.com

0. Introduction

It is known that some diagonal arguments, when formalized, do not demonstrate the impossibility of the diagonal object, but instead just reveal a breakdown in definability or encoding. For example, in a formal setting, Richard's paradox does not yield a contradiction; it instead indicates that one of the relevant sets is ill-defined. (For elaboration and other examples, see Chapter 2 of [10].)

This invites the possibility that other diagonal arguments reflect similar anomalies. The diagonal argument against a universal p.r. function is considered in this light. The impetus is an algorithm that apparently satisfies the criteria for being p.r. while simulating the computation of $f_i(i, n)$ for any index i of a binary p.r. function. The paper does not attempt to explain why this construction apparently survives the usual diagonal objection, but presents it in a form precise enough to support that analysis.

Of course, the algorithm should not be possible, given the standard result that no universal p.r. function exists. But that only reinforces the significance of the construction. It demonstrates that the diagonal argument, at least in this case, does not entail what it is commonly believed to entail. Again, following [10], it is likely that the argument instead shows that certain background assumptions fail. Pursuing this lead is, I hope, a task that the broader community will find worthwhile. But the present paper is focused exclusively on showing that the construction contains no error, and that the standard moral drawn from diagonalization must be incorrect.

One preliminary: The following normally does not discuss p.r. functions directly but rather function symbols $f^n_{\underline{m}}$ within a version of Primitive Recursive Arithmetic (PRA).[1] This is because the algorithm relies crucially on numeric subscripts, and it is unproblematic which numeric subscript a function symbol has. In contrast, it is dubious to assume that, e.g., the factorial function itself has some specific index attached to it (in Plato's heaven, as it were). The discussion thus proceeds primarily by considering function symbols rather than functions per se.

[1] This may suggest a similarity to [7] by Princeton mathematician Edward Nelson, in which PRA was purportedly shown inconsistent. (Tao's response in [12] led to the retraction in [8].) This paper is emphatically not alleging that PRA is inconsistent. Rather, the claim is that the significance of one diagonal argument has been misidentified. Another difference worth mentioning is that Nelson was focused on the induction axiom, whereas this paper concerns a more minimal version of PRA which does not include the induction axiom.

1. Exposition of PRA–

We shall focus on a weakened version of PRA; cf. [1, 4, 11]. Call it PRA–. The weakening is that the induction axiom is replaced with the axiom that any positive integer has a predecessor; cf. [9].[2] But for our purposes, the most important feature of PRA– is the way a p.r. function is assigned a subscripted function symbol; this is detailed later in this section.

First, terms of PRA– are defined inductively as follows:

(a) '0' is a term.

(b) If τ is a term, then so is τ'. ('0' followed by zero or more occurrences of '′' are the numerals. Let $\underline{\tau}$ be the numeral co-referring with τ.)[3]

(c) $v\underline{n}$ is a term (a variable).

(d) If $\tau_1, \ldots, \tau_m$ are terms, then so is $f\underline{nm}(\tau_1 \ldots \tau_m)$ (a non-numeric or nn-term).

But for convenience, Arabic numerals will be used and I revert to using 'x,', 'y', etc., as variables. Also, a nn-term will normally be written as $f_{\underline{n}}(\tau_1, \ldots, \tau_m)$.

The well-formed formulae (wff) of PRA– are defined thus:

(i) If τ_1 and τ_2 are terms, then $\tau_1{=}\tau_2$ is a wff.

(ii) If ϕ is a wff, then so is $\sim\phi$.

(iii) If ϕ and ψ are wff, then so is $(\phi \supset \psi)$.

Assume that '=', '$\sim$', and '$\supset$', have their standard interpretations. Parentheses will be omitted when there is no danger of confusion.

The system has all axioms from the Hilbert schemes:

(H1) $\phi \supset (\phi \supset \phi)$

(H2) $(\phi \supset (\psi \supset \chi)) \supset ((\phi \supset \psi) \supset (\phi \supset \chi))$

(H3) $(\sim \psi \supset \sim \phi) \supset (\phi \supset \psi)$

PRA– also has the standard axiomatic analysis of '=', as per the Law of Universal Identity and the Indiscernability of Identicals:

($U_=$) $x{=}x$

($I_=$) $\tau_1{=}\tau_2 \supset (\phi[x/\tau_1] \supset \phi[x/\tau_2])$

Here and elsewhere, $\phi[x/\tau]$ is the result of uniformly replacing 'x' in ϕ with τ.

[2] Also, PRA may have principles for course-of-values recursion. These are left aside also.

[3] In what follows, I often elide the distinction between use and mention. Quasi-quotes from [8] could be used to distinguish an expression $\ulcorner\tau'\urcorner$ from what it represents. But to avoid clutter, I instead rely on context to disambiguate.

Further, in PRA– each p.r. function is expressed by at least one basic function symbol $f_{\underline{n}}$; axioms that accordingly define functions symbols are included. Per [3], pp. 179-180, these axioms shall be specified inductively.

The inductive specification below is more involved than usual, but when a function symbol has an axiom, its subscript shall denote a code indicating its axiomatic definition,[4] and this will be crucial later. (I assume some standard way of coding symbols, strings of symbols, and sequences of strings; cf. [3] pp. 179.) Moreover, it is important that the coding, and thus, the enumeration of the axioms for the p.r. functions is primitive recursive.[5] Where '*' indicates the concatenation of symbols:

Axioms for base p.r. functions:

The 0^{th} axiom is: $f_0(x)=0$.
The 1^{st} axiom is: $f_1(x)=x'$
If c codes $\underline{j}*\underline{k}$ where $1\leq j\leq k\leq c$, then the c^{th} axiom is:
$f_c(x_1, \ldots, x_k)=x_j$

Axioms for composed p.r. functions:

If c codes the string $\underline{0}*\underline{b}*\underline{c_1}* \ldots *\underline{c_n}*\underline{k}$ (and each of b, $c_1, \ldots, c_n$, and k is less than c), and the b^{th}, $c_1{}^{\text{th}}, \ldots,$ and $c_n{}^{\text{th}}$ axioms define, respectively, $f_{\underline{b}}(x_1, \ldots, x_n)$, $f_{\underline{c1}}(x_1, \ldots, x_k), \ldots,$ and $f_{\underline{cn}}(x_1, \ldots, x_k)$, then the c^{th} axiom is:
$f_c(x_1, \ldots, x_k)=f_b(f_{c1}(x_1, \ldots, x_k), \ldots, f_{cn}(x_1, \ldots, x_k))$

Axioms for p.r. recursions:

If c codes the string '$\prime$'$*\underline{a}*\underline{d}*\underline{k}$ (and each of a, d, and k is less than c), and the a^{th} and d^{th} axioms define, respectively, $f_{\underline{a}}(x_1, \ldots, x_k)$ and $f_{\underline{d}}(x_1, \ldots, x_{k+2})$, then where $(\phi\ \&\ \psi)$ is shorthand for $\sim(\phi\ \supset \sim\psi)$, the c^{th} axiom is:
$f_c(x_1, \ldots, x_k, 0)=f_a(x_1, \ldots, x_k)\ \&$
$f_c(x_1, \ldots, x_k, y')=f_d(f_c(x_1, \ldots, x_k, y), x_1, \ldots, x_k, y)$

In both cases, the parenthetical "and each... is less than c" is eliminable. The index c is independently guaranteed to exceed the arity k, as well as the index for any function used to define $f_{\underline{c}}$. On a standard Gödel numbering, the code for $\underline{n}$ is greater than n; hence, since c codes a string consisting of $\underline{k}$ and the subscripts of the component function symbols, *inter alia*, c will be greater than any of these.[6] This fact about c will be needed at the end of the next section.

[4]While explicit subscript-dependent constructions are not typical, it is standard that primitive recursive enumerations encode the definitions of p.r. functions. See, e.g., Chapter 4 of [2]. The subscripts thus do not introduce a non-standard assumption, but rather make overt a practice normally left implicit.

[5]Since a p.r. enumeration must be total, let n map to 0 when n enumerates no axiom.

[6]Relatedly, with a composed p.r. function, if f_b has arity n, it must be that $n<c$. After all, c codes $n+2$ symbols, and the code for $n+2$ symbols is always greater than n itself.

Moving on, PRA– also has the following arithmetical axioms:

(A1) $\sim 0 = x'$

(A2) $x' = y' \supset x = y$

(A3) $\sim 0 = x \supset \ (\exists y \leq x) x = y'$

Per usual, '$(\exists y \leq x)$' is a bounded existential quantifier.

The rules of inference in PRA– are *modus ponens* and variable substitution:

(MP) From ϕ and $(\phi \supset \psi)$, ψ is derivable.

(VS) From $\phi(x_1, \ldots, x_n)$, $\phi[x/\tau_1], \ldots, [x_n/\tau_n]$ is derivable.

A finite sequence of wff counts as a *derivation* of ϕ in PRA– from a (possibly empty) set of wff Γ iff: The first members are the members of Γ (if any), the last member is ϕ, and any member of the sequence is either a member of Γ, an axiom, or is derivable from previous members via some inference rule in the system. When Γ is empty, we say that the sequence is a *proof* of ϕ in PRA–.

2. Schematics for a universal function

This section begins the construction of the following function u:

$$u(i, n) = m \text{ iff } f_{\underline{i}}(\underline{i}, \underline{n}) = \underline{m} \text{ is a theorem of PRA–.}$$

It should be uncontroversial that some function can compute the ith binary p.r. function on inputs i and n. But as will be argued in section 6, our construction reveals that the function is p.r.

The construction starts with a p.r. proof predicate—but like PRA, PRA– has no predicate that represents *in toto* the relations between theorems and their proofs, as it lacks unbounded quantifiers. Yet, if ϕ has Gödel number $\ulcorner \phi \urcorner$, we can define an "i-bounded" p.r. proof predicate $B(i, n, \ulcorner \phi \urcorner)$, which is satisfied iff ϕ has a proof coded by n where any axiom may occur except those beyond the i^{th} axiom defining a p.r. function symbol.

In defining the i-bounded proof predicate, we skip several details since they are somewhat tedious and are identical to those for Gödel's predicate $B(x, y)$ ([3], p. 186, #45). Or at least, only minor revisions are required. (E.g., unlike Gödel's system, PRA– is first-order only and has no quantifiers; many of Gödel's definitions are thus more complex than we need.) The only revisions which may not be obvious concern the definition of an i-bounded "axiom predicate" $Ax(i, n)$ (cf. #42 in [3]). This is a predicate which represents the axioms for PRA– except those beyond the i^{th} axiomatic definition of a p.r. function symbol.

In defining $Ax(i, n)$, the predicates representing the logical, arithmetical, and '=' axioms are presumed given. What remains is to construct a p.r. predicate $AxPR(i, j)$ which is satisfied iff j codes the i^{th} or earlier axiom for a p.r. function. Recall that we gave earlier a p.r. enumeration of (the codes of) these axioms.

Hence, we know that there is a p.r. enumeration e such that if $e(i) \neq 0$, $e(i)$ codes the axiom defining the i^{th} p.r. function symbol.[7] This enumeration allows us to adequately define the predicate as: $AxPR(i, j)$ iff $(\exists x \leq i)(e(x) = j \;\&\; j \neq 0)$.

We now can claim an i-bounded p.r. proof predicate $B(i, n, \ulcorner \phi \urcorner)$, defined in the manner of [3]. From here, we can then define $U(i, n, m)$ as a predicate that strongly represents the function u. Briefly:

$$U(i, n, m) \text{ iff } B(i, s(i, n), \ulcorner f_{\underline{i}}(\underline{i}, \underline{n}) = \underline{m} \urcorner)$$

This indicates that $U(i, n, m)$ holds iff $s(i, n)$ codes an i-bounded proof in PRA– of $f_{\underline{i}}(\underline{i}, \underline{n}) = \underline{m}$. The suggestion will be that, if this formula has a proof at all, the object coded by $s(i, n)$ will count as one (and otherwise it will not).

Roughly, the idea is to define $s(i, n)$ as follows, where $e(i)$ again enumerates (the codes of) the axioms for the function symbols:

$$s(i, n) = \begin{cases} cp(\ulcorner f_{\underline{i}}(\underline{i}, \underline{n}) = \underline{m} \urcorner) & \text{if } e(i) \text{ codes an axiom defining } f^2_{\underline{i}}, \\ 0 & \text{otherwise.} \end{cases}$$

This is meant to express that, when the defining condition is satisfied, s outputs the code for a "canonical proof" of $f_{\underline{i}}(\underline{i}, \underline{n}) = \underline{m}$, for some $\underline{m}$. Thus, s purports to compute what m is, given only i and n, by generating a proof wherein the term $f_{\underline{i}}(\underline{i}, \underline{n})$ is reduced to a numeral.

Our concern, however, is whether this proof generator can be p.r. Again, diagonal arguments suggest a negative answer. Yet the peculiarity is that it seems one can describe such an algorithm. Here is a brief, intuitive gloss. (The detailed description starts in section 4.) First, the algorithm checks whether $e(i)$ codes an axiom defining $f^2_{\underline{i}}$. If not, it outputs 0 and halts. Otherwise, it starts a proof with this axiom, and then instantiates it on i and n; this leads to an equality $f_{\underline{i}}(\underline{i}, \underline{n}) = \phi_i(\underline{i}, \underline{n})$ (unless $f_{\underline{i}}$ expresses a basic binary projection, which the algorithm can handle separately). The proof now adds a line which is the same as the preceding except that, to the right of '=', the rightmost, innermost nn-term τ is replaced by a term that defines it. (The defining term is recovered from the information in τ's subscript.) The algorithm then repeats this as needed, replacing the rightmost, innermost term by a term indicated by its subscript, until a line is reached where '=' is followed only by a numeral.

At a glance, however, the algorithm seems to require an unbounded search, for it is unknown how many replacements are needed to reduce a given term. Yet the algorithm can halt at the right time without this information. After each

[7] This does not imply the existence of a p.r. function $g(i)$ that outputs the i^{th} p.r. function. (That alone would allow a contradictory diagonal function.) Our function does not have functions in its range, but rather codes for various syntactic strings. These happen to code definitions of p.r. functions. But a malignant diagonal function would need to *compute* the i^{th} p.r. function on input i. And that may require an unbounded search of, e.g., the proofs of PRA–, to find one ending with an equality that identifies what $f_i(i)$ is.

reduction, the algorithm simply checks whether zero 'f's are found to the right of '=' in the latest line of the proof (where this check is bounded by the length of the wff). If zero 'f's are found, the algorithm halts; otherwise, not. By this means, the process will halt exactly when it should halt. So effectively, the algorithm is able to compute any binary p.r. term at the requisite (primitive) recursive depth.

One might ask whether a well-defined algorithm is even able to "access" the subscript on a function symbol. Officially, however, a nn-term has no subscript; it is of the form $f\underline{nm}(\tau_1 \ldots \tau_m)$. More importantly, operations defined on such indicies is standard practice, as seen in the definitions of $AxPR(i,\ n)$ and of T-predicate from [5].

If all this is correct, then since a p.r. function symbol is here defined only by function symbols with lower subscripts, a proof of $f_{\underline{i}}(\underline{i},\ \underline{n})=\underline{m}$ yielded by the algorithm would count as an "i-bounded" proof. In which case, $s(i,\ n)$ would render true the sentence $B(i, s(i, n), \ulcorner f_{\underline{i}}(\underline{i}, \underline{n}) = \underline{m} \urcorner)$, as intended.

3. Preparatory remarks, re: function symbols within proofs

This section illustrates how the subscripts on function symbols can guide the generation of a proof, to give a clearer sense of how the algorithm works. This is not strictly necessary, however, and so some readers may prefer to skip to the next section.

As an illustration of the information coded by the indices, consider the term $f_{\underline{c}}(\underline{1},\ \underline{2})$, where c codes the string '$\prime$'$*\underline{q}*\underline{d}*\underline{1}$. The 1st member of the string tells us that $f_{\underline{c}}$ expresses a p.r. recursion, whereas the 2nd and 3rd members indicate that $f_{\underline{c}}$ is defined by $f_{\underline{q}}$ and $f_{\underline{d}}$. So we can recover that the axiom for $f_{\underline{c}}$ is $f_{\underline{c}}(x,\ \underline{0})=f_{\underline{q}}(x)$ & $f_{\underline{c}}(x,\ y')=f_{\underline{d}}(f_{\underline{c}}(x,\ y),\ x,\ y)$.

Suppose also that q codes $\underline{1}*\underline{1}$, whence it expresses simple identity. And suppose d codes $\underline{0}*\underline{p}*\underline{1}*\underline{q}*\underline{q}*\underline{3}$. Since the string begins with $\underline{0}$, it indicates that d indexes a composed p.r. function—further, that $f_{\underline{d}}$ is defined by $f_{\underline{p}}$, $f_{\underline{1}}$, and two occurrences of $f_{\underline{q}}$. So we can recover that its axiom is $f_{\underline{d}}(z,\ x,\ y)=f_{\underline{p}}(f_{\underline{1}}(z),\ f_{\underline{q}}(x),\ f_{\underline{q}}(y))$. (It may seem unnecessary for the axiom to include $f_{\underline{q}}$, but this is so that it has the right form for a composed p.r. function.)

Suppose now that p codes $\underline{1}*\underline{3}$ so that $f_{\underline{p}}$ expresses the ternary 1st-projection function. Then, $f_{\underline{c}}$ expresses addition and that $f_{\underline{c}}(\underline{1},\ \underline{2})=\underline{3}$. The algorithm will yield a proof of this by accessing in this way the information encoded in c.

For expediency's sake, the proofs shall utilize inference rules corresponding to standard evaluation rules for p.r. terms. Where $\underline{n}$, $\underline{n_1}$, $\underline{n_2}$, ... are numerals:

(Z) From $\phi(f_{\underline{0}}(\underline{n}))$, $\phi(\underline{0})$ is derivable.

(S) From $\phi(f_{\underline{1}}(\underline{n}))$, $\phi(\underline{n'})$ is derivable.

(P) If $\underline{c}$ codes $\underline{j}^*\underline{k}$ and $1 \leq j \leq k \leq c$, then from $\phi(f_{\underline{c}}(\underline{n}_1, \ldots, \underline{n}_k))$, $\phi(\underline{n}_j)$ is derivable.

(C) If c codes the string $\underline{0}*\underline{b}*\underline{c_1}*\ldots*\underline{c_n}*\underline{k}$, and the b^{th}, ${c_1}^{\text{th}}$, ..., and ${c_n}^{\text{th}}$ axioms define $f_{\underline{b}}(x_1, \ldots, x_k)$, $f_{\underline{c}_1}(x_1, \ldots, x_k)$, ..., and $f_{\underline{c}_n}(x_1, \ldots, x_k)$, respectively, then from $\phi(f_{\underline{c}}(\underline{n}_1, \ldots, \underline{n}_k))$, $\phi(f_{\underline{b}}(f_{\underline{c}_1}(\underline{n}_1, \ldots, \underline{n}_k), \ldots, f_{\underline{c}_n}(\underline{n}_1, \ldots, \underline{n}_k)))$ is derivable.

(R1) If c codes the string $\text{`}'\text{'}*\underline{a}^*\underline{d}^*\underline{k}$ and the a^{th} and d^{th} axioms define $f_{\underline{a}}(x_1, \ldots, x_k)$ and $f_{\underline{d}}(x_1, \ldots, x_{k+2})$, respectively, then from $\phi(f_{\underline{c}}(\underline{n}_1, \ldots, \underline{n}_k, \underline{0}))$, $\phi(f_{\underline{a}}(\underline{n}_1, \ldots, \underline{n}_k))$ is derivable.

(R2) Under the same antecedent as (R1), from $\phi(f_{\underline{c}}(\underline{n}_1, \ldots, \underline{n}_k, \underline{n}'))$, $\phi(f_{\underline{d}}(f_{\underline{c}}(\underline{n}_1, \ldots, \underline{n}_k, \underline{n}), \underline{n}_1, \ldots, \underline{n}_k, \underline{n}))$ is derivable.

These are in fact more restrictive than the usual evaluation rules, for they apply to terms loaded with numerals only. (This is to simplify things later.) Regardless, we prove in section 5 that the rules are sound. Soundness means, moreover, that they are mere shortcuts for what could be proven otherwise in PRA–. Our remarks will therefore bear on PRA–, even though the shortcut rules are not officially part of that system.

We now give an example to illustrate how the subscripts on function symbols guide the application of these rules. To save time, I make use of the standard elimination rule for '&'; also, some occurrences of '$f_{\underline{q}}(\ )$' are omitted as trivial, although a few are included since they clarify how (R1) is applied. Apart from these omissions, the example is in conformity with the algorithm given later. (To reduce clutter, I omit most underlines, but take heed that numerals appear as such in the proof.) Where $\ulcorner c \urcorner^{-1}$ is the string coded by c:

1.	$f_c(x,0) = f_q(x)$ & $f_c(x,y') = f_d(f_c(x,y),x,y)$	[Axiom for $f_{\underline{c}}$]
2.	$f_c(0',0) = f_q(0')$ & $f_c(0',0'') = f_d(f_c(0',0'),0',0')$	[(VS), 1: $x/0', y/0'$]
3.	$f_c(0',0'') = f_d(f_c(0',0'),0',0')$	[(&E), 2]
4.	$f_c(0',0'') = f_d(f_d(f_c(0',0),0',0),0',0')$	[(R2), 3: $\underline{d}$ = 3rd in $\ulcorner c \urcorner^{-1}$]
5.	$f_c(0',0'') = f_d(f_d(f_q(0'),0',0),0',0')$	[(R1), 4: $\underline{q}$ = 2nd in $\ulcorner c \urcorner^{-1}$]
6.	$f_c(0',0'') = f_d(f_d(0',0',0),0',0')$	[(P), 5: q codes $\underline{1}*\underline{1}$]
7.	$f_c(0',0'') = f_d(f_p(f_1(0'),0',0),0',0')$	[(C), 6: $\underline{p}$, $\underline{1}$ are 2nd, 3rd in $\ulcorner d \urcorner^{-1}$]
8.	$f_c(0',0'') = f_d(f_p(0'',0',0),0',0')$	[(S), 7: subscript is $\underline{1}$]
9.	$f_c(0',0'') = f_d(0'',0',0')$	[(P), 8; p codes $\underline{1}*\underline{3}$]
10.	$f_c(0',0'') = f_p(f_1(0''),0',0')$	[(C), 9: $\underline{p}$, $\underline{1}$ are 2nd, 3rd in $\ulcorner d \urcorner^{-1}$]
11.	$f_c(0',0'') = f_p(0''',0',0')$	[(S), 10: subscript is $\underline{1}$]
12.	$f_c(0',0'') = 0'''$	[(P), 11: p codes $\underline{1}*\underline{3}$]

This example elucidates how subscripts on function symbols enable the construction of the proofs, and in particular, the application of the shortcut rules.

4. An algorithm for canonical proofs

We now present the algorithm for canonical proofs of $f_{\underline{i}}(\underline{i}, \underline{n})=\underline{m}$ (when the formula has a proof at all). Throughout, the algorithm is often described as operating on linguistic strings rather than on the codes for these strings. This is for simplicity's sake, and since coding and decoding is p.r., it is of no import.

Terminology: The indices for some functions will code a string $\underline{0}^*\underline{b}^*\underline{c_1}^* \ldots {}^*\underline{c_n}^*\underline{k}$, where the b^{th}, ${c_1}^{\text{th}}$, ..., and ${c_n}^{\text{th}}$ axioms define $f_{\underline{b}}(x_1, \ldots, x_k)$, $f_{\underline{c1}}(x_1, \ldots, x_k)$, ..., and $f_{\underline{cn}}(x_1, \ldots, x_k)$, respectively. Call such an index a "composition index." Other indices will code a string $'^*\underline{a}^*\underline{d}$, where the a^{th} and d^{th} axioms define $f_{\underline{a}}(x_1, \ldots, x_k)$ and $f_{\underline{d}}(x_1, \ldots, x_{k+2})$, respectively. Call such an index a "recursion index." Also, let us jointly refer to composition indices and recursion indices as "complex indices." The other indices we shall call "simple indices."

Suppose now that $e(i)$ enumerates the axiom for $f^{\underline{2}}_{\underline{i}}$. Then, $s(i,n)$ is the code of the sequence that is determined as follows:

Step 1. [Basic Binary Projections] If i does not code $\underline{j}^*\underline{2}$, where j equals 1 or 2, then go to Step 2. Otherwise:

A. If j=1, output the following sequence and then halt:

1. $f_{\underline{i}}(x, y)=x$	[Axiom for $f_{\underline{i}}$]
2. $f_{\underline{i}}(\underline{i}, \underline{n})=\underline{i}$	[(VS), 1: $x/\underline{i}$, $y/\underline{n}$]

B. If j=2, output the following sequence and then halt:

1. $f_{\underline{i}}(x, y)=y$	[Axiom for $f_{\underline{i}}$]
2. $f_{\underline{i}}(\underline{i}, \underline{n})=\underline{n}$	[(VS), 1: $x/\underline{i}$, $y/\underline{n}$]

Step 2. [Getting the Baseline Term for P.R. Compositions] If i is not a composition index, go to Step 3. Otherwise, start the sequence as follows and then go to Step 4:

1. $f_{\underline{i}}(x, y)=\phi_i(x, y)$	[Axiom for $f_{\underline{i}}$]
2. $f_{\underline{i}}(\underline{i}, \underline{n})=\phi_i(\underline{i}, \underline{n})$	[(VS), 1: $x/\underline{i}$, $y/\underline{n}$]

Step 3. [Getting the Baseline Term for P.R. Recursions]

A. If n=0, start the sequence as follows and then go to Step 4:

1. $f_{\underline{i}}(x, 0)=f_{\underline{a}}(x)$ & $f_{\underline{i}}(x, y')=f_{\underline{d}}(f_{\underline{i}}(x, y), x, y)$	[Axiom for $f_{\underline{i}}$]
2. $f_{\underline{i}}(\underline{i}, 0)=f_{\underline{a}}(\underline{i})$ & $f_{\underline{i}}(\underline{i}, \underline{n}')=f_{\underline{d}}(f_{\underline{i}}(\underline{i}, \underline{n}), \underline{i}, \underline{n})$	[(VS), 1: $x/\underline{i}$, $y/\underline{n}$]
3. $f_{\underline{i}}(\underline{i}, 0)=f_{\underline{a}}(\underline{i})$	[(&E), 2]

B. If $n \neq 0$, start the sequence as follows and then go to Step 4:

1. $f_{\underline{i}}(x, 0)=f_{\underline{a}}(x)$ & $f_{\underline{i}}(x, y')=f_{\underline{d}}(f_{\underline{i}}(x, y), x, y)$	[Axiom for $f_{\underline{i}}$]
2. $f_{\underline{i}}(\underline{i}, 0)=f_{\underline{a}}(\underline{i})$ & $f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}')=f_{\underline{d}}(f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}), \underline{i}, \underline{n\text{-}1})$	[(VS), 1: $x/\underline{i}$, $y/\underline{n\text{-}1}$]
3. $f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}')=f_{\underline{d}}(f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}), \underline{i}, \underline{n\text{-}1})$	[(&E), 2]

Step 4. [Entering the Main Loop.] The latest line of the sequence is an equality; call the right-hand term the "baseline term" for the sequence. Check whether 'f' occurs in the baseline term. (This check is bounded by the total number of symbols in the wff.[8]) If there are no occurrences of 'f', halt. Otherwise, (re)start the Main Loop: For the present iteration of the Loop, let ϵ be the equation on the latest line. To the right of '=', find the nn-term embedded in the most parentheses (a.k.a., the "innermost" nn-term). When there is a tie, choose the rightmost one. Let τ be the rightmost, innermost nn-term (to the right of '=') for the current iteration of the Loop. (N.B., τ will be loaded with numerals only; we prove this later.) Go to Step 5.

Step 5. [Computing τ] Check the index c for τ. It is either 0, 1, or codes $\underline{j}^*\underline{k}$, where $1 \leq j \leq k \leq c$—alternatively, it is a composition or recursion index.

A. If c=0, apply (Z): Add a line where τ is replaced in ϵ with $\underline{0}$. Go back to Step 4.

B. If c=1, then τ is loaded with some $\underline{m}$. Apply (S): Add a line where τ is replaced in ϵ with $\underline{m}'$. Go back to Step 4.

C. If c codes $\underline{j}^*\underline{k}$ where $1 \leq j \leq k \leq c$, then apply (P): Add a line where τ is replaced in ϵ with the numeral in the j^{th} position of τ. Go back to Step 4.

D. If c is a composition index, then apply (C): Add a line where τ is replaced in ϵ with $f_{\underline{b}}(f_{\underline{c1}}(\underline{n_1}, \ldots, \underline{n_k}), \ldots, f_{\underline{cl}}(\underline{n_1}, \ldots, \underline{n_k}))$, where $\underline{n_1}, \ldots, \underline{n_k}$ are the same as in τ, and $\underline{b}$ is the 2^{nd} member of the string coded by c, $\underline{c_1}$ is 3^{rd} member of the coded string coded by c, ..., and $\underline{c_l}$ is l+2^{th} member of the string coded by c. Go back to Step 4.

E. If c is a recursion index, then:

   i. If τ is loaded with $\underline{n_1}, \ldots, \underline{n_k}, \underline{0}$, then apply (R1): Add a line where τ is replaced in ϵ with $f_{\underline{a}}(\underline{n_1}, \ldots, \underline{n_k})$, where $\underline{a}$ is the 2^{nd} member of the string coded by c. Go back to Step 4.

   ii. If τ is loaded with $\underline{n_1}, \ldots, \underline{n_k}, \underline{m \neq 0}$, then apply (R2): Add a line where τ is replaced in ϵ with $f_{\underline{d}}(f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m\text{-}1}), \underline{n_1}, \ldots, \underline{n_k}, \underline{m\text{–}1})$, where $\underline{c}$ is the same as in τ and $\underline{d}$ is the 3^{rd} member of the string coded by c. Go back to Step 4.

[8] We make crucial use here of the p.r. function $l(x)$ in [3], p. 182, #7

Again, the algorithm generates a proof where $f_{\underline{i}}(\underline{i}, \underline{n})$ is computed for any n, provided that the i^{th} axiom for the function symbols defines a binary symbol. The next section proves this, and in the section after that, an argument is given for why the algorithm is p.r.

5. Proof that the algorithm is correct

In precise terms, the claim to be proved is:

> **Theorem**: If $e(i)$ codes the axiom defining $f^{\underline{2}}_{\underline{i}}$, then given i and any n, there is an m such that the algorithm generates a unique proof of $f_{\underline{i}}(\underline{i}, \underline{n})=\underline{m}$, at which point the algorithm halts.

Note that here and elsewhere in this section, our statements employ unbounded quantification; however, this alone does not undermine that the algorithm is p.r. **Theorem** is a claim about the algorithm, not part of the algorithm itself.

Establishing **Theorem** is best approached by considering p.r. terms with simple indices first, and then considering separately those with complex indices.

Simple Indices: Vacuously, **Theorem** holds if the index i=0; after all, the 0^{th} function symbol is not binary, and a binary function-symbol with subscript '0' has no axiom in the system. Similar remarks apply with every other simple index, except an index that codes $\underline{1}^*\ \underline{2}$ or $\underline{2}^*\ \underline{2}$. But in those cases, the relevant function is a basic binary projection, and the algorithm clearly handles these under Step 1 in a way that satisfies **Theorem**.

Complex Indices: We want to show that **Theorem** holds when i is a complex index. This part of the proof will require three lemmas. The first is as follows:

> Lemma 1: If $f^{\underline{2}}_{\underline{i}}$ has a complex index, then for any n, the algorithm starts with a *proof* of $f_{\underline{i}}(\underline{i}, \underline{n})=\beta$, where β is a baseline term and has no free variables.

Proof: If i is complex, there are three kinds of case to consider. The first is where i is a composition index. The other two are where i is a recursion index and where n=0 and $n \neq 0$, respectively. In the first case, the algorithm starts:

1. $f_{\underline{i}}(x, y)=\phi_i(x, y)$	[Axiom for $f_{\underline{i}}$]
2. $f_{\underline{i}}(\underline{i}, \underline{n})=\phi_i(\underline{i}, \underline{n})$	[(VS), 1: $x/\underline{i}$, $y/\underline{n}$]

The rule used is obviously sound; also,$\phi_i(\underline{i}, \underline{n})$ has no free variables and is by definition a baseline term. So the second line verifies Lemma 1 in this first case. In the second case, the algorithm begins:

1. $f_{\underline{i}}(x, 0)=f_{\underline{a}}(x)$ & $f_{\underline{i}}(x, y')=f_{\underline{b}}(f_{\underline{i}}(x, y), x, y)$	[Axiom for $f_{\underline{i}}$]
2. $f_{\underline{i}}(\underline{i}, 0)=f_{\underline{a}}(\underline{i})$ & $f_{\underline{i}}(\underline{i}, \underline{n}')=f_{\underline{b}}(f_{\underline{i}}(\underline{i}, \underline{n}), \underline{i}, \underline{n})$	[(VS), 1: $x/\underline{i}$, $y/\underline{n}$]
3. $f_{\underline{i}}(\underline{i}, 0)=f_{\underline{a}}(\underline{i})$	[(&E), 2]

The rules used here are sound; also, $f_{\underline{a}}(\underline{i})$ has no free variables and is by definition a baseline term. So the third line verifies Lemma 1 in the second case. In the third case, the algorithm starts as follows:

1. $f_{\underline{i}}(x, 0)=f_{\underline{a}}(x)$ & $f_{\underline{i}}(x, y')=f_{\underline{b}}(f_{\underline{i}}(x, y), x, y)$	[Axiom for $f_{\underline{i}}$]
2. $f_{\underline{i}}(\underline{i}, 0)=f_{\underline{a}}(\underline{i})$ & $f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}')=f_{\underline{b}}(f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}), \underline{i}, \underline{n\text{-}1})$	[(VS), 1: $x/\underline{i}$, $y/\underline{n\text{-}1}$]
3. $f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}')=f_{\underline{b}}(f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}), \underline{i}, \underline{n\text{-}1})$	[(&E), 2]

The same rules are used as before, and again, $f_{\underline{b}}(f_{\underline{i}}(\underline{i}, \underline{n\text{-}1}), \underline{i}, \underline{n\text{-}1})$ has no free variables and is by definition a baseline term. So the third line verifies Lemma 1 in the third case, which completes the proof of Lemma 1.

Remark: Like the latter two cases, the baseline term in the first case will have at least one occurrence of 'f'. After all, $f_{\underline{i}}$ in the first case expresses a composed p.r. function, and the axiom for $f_{\underline{i}}$ therefore will contain at least one 'f' on the right-hand side (and the baseline term is an instantiation of the right-hand side).

The second lemma required is the following:

> Lemma 2: [Soundness] If the algorithm applies (Z), (S), (P), (C), (R1), or (R2) to a line in order to produce a new line of a sequence, then in the standard model, if the former line is true, so is the latter.

We prove this by considering the use of each of the rules.

Preliminary: The algorithm is designed to apply the shortcut rules only to the rightmost, innermost nn-term τ on a line. So, τ will be loaded with numerals only. For if τ were loaded with a nn-term, it would not be the innermost. (Also, we also know from Lemma 1 that a baseline term has no free variables, and none of the inference rules introduce free variables into the picture.)

Applying (Z): Suppose the algorithm applies (Z) so to produce $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$. Then, per the instructions on Step 5A, τ_1 must be of the form $f_{\underline{0}}(\underline{m})$, for some numeral $\underline{m}$. Also, per those instructions, τ_2 must be $\underline{0}$. Since $f_{\underline{0}}$ expresses the constantly-zero function, Lemma 2 is verified in the case of (Z).

Applying (S): Suppose the algorithm applies (S) so to produce $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$. Then, per the instructions on Step 5B, τ_1 must be of the form $f_{\underline{1}}(\underline{m})$, for some $\underline{m}$. Also, per those instructions, τ_2 must be $\underline{m}'$. Thus, since $f_{\underline{1}}$ expresses the successor function, Lemma 2 is verified in the case of (S).

Applying (P): Suppose the algorithm applies (P) so to produce $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$ Then, per the instructions on Step 5C, τ_1 must be of the form $f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k})$, where c codes $\underline{j}^*\underline{k}$ and $1 \leq j \leq k \leq c$. Also, per those instructions, τ_2 must be $\underline{n_j}$. Thus, since $f_{\underline{c}}$ expresses the k-ary j^{th}-projection function, Lemma 2 is verified in the case of (P).

Applying (C): Suppose the algorithm applies (C) so to produce $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$. Then, per the instructions on Step 5D, τ_1 must be of the

form $f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k})$, where c codes $\underline{0}{}^*\underline{b}{}^*\underline{c_1}{}^*\ldots{}^*\underline{c_l}{}^*\underline{k}$. Also, per those instructions, τ_2 must be $f_{\underline{b}}(f_{\underline{c1}}(\underline{n_1}, \ldots, \underline{n_k}), \ldots, f_{\underline{cl}}(\underline{n_1}, \ldots, \underline{n_k}))$. Observe that, given how the indices for composed functions are assigned, $f_{\underline{c}}$ expresses a function where the l-ary function expressed by $f_{\underline{b}}$ is composed with the k-ary functions expressed by $f_{\underline{c1}}, \ldots,$ and $f_{\underline{cl}}$. The consequence is that $f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k})$ is co-referential with $f_{\underline{b}}(f_{\underline{c1}}(\underline{n_1}, \ldots, \underline{n_k}), \ldots, f_{\underline{cl}}(\underline{n_1}, \ldots, \underline{n_k}))$. Thus, the transition from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$ to $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ is sound, and Lemma 2 is verified in the case of (C).

Applying (R1): Suppose the algorithm applies (R1) so to produce $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$. Then, per the instructions on Step 5Ei, τ_1 must be of the form $f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k}, \underline{0})$ where c codes $\text{`}'\text{'}{}^*\underline{a}{}^*\underline{d}{}^*\underline{k}$. Also, per those instructions, τ_2 must be $f_{\underline{a}}(\underline{n_1}, \ldots, \underline{n_k})$. Observe that, given how the indices for composed functions are assigned, $f_{\underline{c}}$ expresses a function which is recursively defined by $f_{\underline{a}}$ and $f_{\underline{d}}$. The consequence is that $f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k}, \underline{0})$ co-refers with $f_{\underline{a}}(\underline{n_1}, \ldots, \underline{n_k})$. So the transition from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$ to $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ is sound, and Lemma 2 is verified in the case of (R1).

Applying (R2): Suppose the algorithm applies (R2) so to produce $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$. Then, per the instructions on Step 5Eii, τ_1 must be of the form $f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m})$ for $m>0$, where c codes $\text{`}'\text{'}{}^*\underline{a}{}^*\underline{d}{}^*\underline{k}$. Also, per those instructions, τ_2 must be $f_{\underline{d}}(f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m\text{-}1}), \underline{n_1}, \ldots, \underline{n_k}, \underline{m\text{–}1})$. Observe that, given how the indices for composed functions are assigned, $f_{\underline{c}}$ expresses a function which is recursively defined by $f_{\underline{a}}$ and $f_{\underline{d}}$. The consequence is that $f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m})$ co-refers with $f_{\underline{d}}(f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m\text{-}1}), \underline{n_1}, \ldots, \underline{n_k}, \underline{m\text{–}1})$. So the transition from $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_1)$ to $f_{\underline{i}}(\underline{i}, \underline{n})=\phi(\tau_2)$ is sound, and Lemma 2 is verified in the case of (R2). This completes the proof of Lemma 2.

The third lemma is now stated as follows:

> Lemma 3: If i is a complex index, then for any n, if the algorithm derives $f_{\underline{i}}(\underline{i}, \underline{n})=\beta$, where β is a baseline term, it will continue until it derives $f_{\underline{i}}(\underline{i}, \underline{n})=\underline{m}$, for some $\underline{m}$, after which the algorithm halts.

Proof: Given any complex i and n, assume the algorithm derives $f_{\underline{i}}(\underline{i}, \underline{n})=\beta$, where β is a baseline term. As we saw, a baseline term has at least one occurrence of 'f'—so at this point, the algorithm will iterate the Main Loop at least once. Now the Main Loop is defined in such a way that, on a given iteration, it looks at the equation ϵ (on the latest line) and to the right of '=', it finds the rightmost, innermost nn-term τ, if any. If it does not find such a τ, then the right-hand term of the equation is a numeral and the algorithm halts—in which case, Lemma 3 is verified. Otherwise, if finds such a τ, it adds a line, whereby τ is replaced in ϵ with a term that is a p.r. reduction of τ. The algorithm then checks whether there are zero occurrences of 'f'. If so, then the algorithm halts and we know that the term right of '=' on the latest line is a numeral. In which case, Lemma 3 is verified. Otherwise, The Main Loop will be restarted to further reduce the right-hand term on the latest line.

So in brief, when the algorithm locates a rightmost, innermost nn-term (to the right of '='), it adds a line where that term is removed and replaced with a

p.r. reduction of the term. The replacement may not be a numeral, but no matter: The algorithm will continue iterating and the replacements are made in a determinate order, until a line is reached where the right-hand term is a numeral. And there will be such a line, given that each p.r. term is ultimately reducible to a numeral in finitely many steps, as per the definitional rules for p.r. functions. So the algorithm will produce a derivation from $f_{\underline{i}}(\underline{i}, \underline{n})=\beta$ to $f_{\underline{i}}(\underline{i}, \underline{n})=\underline{m}$, for some $\underline{m}$, by iterating the Main Loop a sufficient number of times, after which it will stop—which is what Lemma 3 says.

From Lemma 1, Lemma 2, and Lemma 3, we know that if i is a complex index, the algorithm yields a unique *proof* of $f_{\underline{i}}(\underline{i}, \underline{n})=\underline{m}$, for some $\underline{m}$. So **Theorem** holds in the case of complex indices, which completes the proof of **Theorem**.

6. Why the algorithm is p.r.

We have just shown that the algorithm behaves as advertised, and in this section, we present the argument that the algorithm is p.r.

A key concern should be cleared away first. It will be objected that, since we have no explicit bound on the number of iterations of the Main Loop, the algorithm cannot be p.r. But the premise is correct only if the algorithm requires such a number to count down the iterations when reducing a term. Yet that is not how our algorithm works. It instead performs a simple check before each iteration to verify whether a numeral has been reached. If a numeral appears on the right of the latest line of the proof, then the algorithm halts; otherwise, it restarts the Loop. As Lemma 3 assures us, this alone makes the algorithm halt exaclty when it should. And so, the algorithm is able to function without knowing how many iterations are necessary to reduce a given p.r. term. A bound for such a number is accordingly unneeded.

It remains to be shown, however, that all steps which *are* part of the algorithm are finitely bounded. We show this by going through each of the steps, verifying how no unbounded searches are undertaken.

However, it is obvious that steps 1–3 are finitely bounded; we need only confirm this for steps 4 and 5 (the Main Loop). In the case of step 4, consider that the algorithm first applies Gödel's "length" operation (cf. note 8 above) to determine the number w of symbols in the latest line of the proof. This number w is set as a bound to check whether zero 'f's occur to the right of '=' in the latest line, thus making the check p.r. If no 'f's occur, the algorithm halts. Otherwise, the algorithm can find the rightmost, innermost nn-term in the latest line of the proof by scanning the term's parentheses structure in a single left-to-right pass to determine nesting depth. This is a bounded operation on a finite string and thus is primitive recursive. If more detail is required, consider that the algorithm counts the parenthesis-pairs—not greater than w—which enclose each 'f' to the right of '=' (of which there are also less than w). An occurrence of 'f'

that yields the greatest count begins an innermost nn-term.[9] If there is more than one such 'f', the rightmost one is the last one in the wff. (If the innermost terms in the wff are enumerated left-to-right, it will be the one enumerated by the greatest number, not greater than w.) Again, the relevant maneuvers are p.r.; they amount to bounded arithmetic operations on finite sequences.[10]

As for step 5, the algorithm adds a new line to the proof, which is identical to the previous line, except that the rightmost, innermost nn-term τ has been replaced. Importantly, replacement is p.r.; cf. the function Sb, #31 in [3]. In more detail, the algorithm first identifies the index for τ as 0, 1, or as coding a specific sort of string. (If the last, the first element of the string indicates if the index is for projection function, or if it is a composition or a recursion index.) From this determination, the algorithm decides on one of the replacements listed below. (Since there are only finitely many replacement templates, this decision process may be implemented by a p.r. predicate.)

> Per (Z): If τ is $f_{\underline{0}}(\underline{m})$, replace it with $\underline{0}$.
>
> Per (S): If τ is $f_{\underline{1}}(\underline{m})$, replace it with $\underline{m'}$.
>
> Per (P): If τ is $f_{\underline{c}}(\underline{n_1}, \ \ldots, \underline{n_k})$ where c codes $\underline{j}^*\underline{k}$ and $1 \leq j \leq k \leq c$, replace it with $\underline{n_j}$.
>
> Per (C): If τ is $f_{\underline{c}}(\underline{n_1}, \ \ldots, \underline{n_k})$, and c is a composition index, replace it with $f_{\underline{b}}(f_{\underline{c1}}(\underline{n_1}, \ldots, \underline{n_k}), \ldots, f_{\underline{cl}}(\underline{n_1}, \ldots, \underline{n_k}))$.
>
> Per (R1): If τ is $f_{\underline{c}}(\underline{n_1}, \ \ldots, \underline{n_k}, \underline{0})$, and c is a recursion index, replace it with $f_{\underline{a}}(\underline{n_1}, \ \ldots, \underline{n_k})$.
>
> Per (R2): If τ is $f_{\underline{c}}(\underline{n_1}, \ \ldots, \underline{n_k}, \underline{m'})$, and c is a recursion index, replace it with $f_{\underline{d}}(f_{\underline{c}}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m}), \underline{n_1}, \ldots, \underline{n_k}, \underline{m})$.

In each case, the replacement operation is bounded by the length of the term, and the output string can be constructed explicitly using only primitive recursive operations. In more detail, given c, the algorithm extracts $\underline{j}$, $\underline{k}$, $\underline{b}$, $\underline{c_1}$, $\ldots$, $\underline{c_l}$, $\underline{a}$, or $\underline{d}$, as needed, and such extraction is p.r. And the replacing term for τ is constructed in a p.r. manner since this at most requires replacement into finite term-schemes (which are found in the axiom-schemes for the function symbols).

It is thus verifiable that the algorithm never engages in an indefinite search. The proof procedure is implemented by direct pattern-matching against a fixed finite set of templates, and the algorithm always operates over terms of bounded length. In short, while the algorithm computes a function that would ordinarily be excluded by a diagonal argument on its standard interpretation, the procedure conforms to the criteria for primitive recursion.

---

[9] Note that codes for terms in PRA– can be identified by the first-order term operation, #18 in [3].

[10] My thanks to [redacted] for assistance with the argument of this paragraph.

7. Closing

The algorithm presented in this paper is primitive recursive while enabling the behavior of a universal p.r. function. This challenges the standard view where diagonalization demonstrates the non-existence of such a function. Other diagonal arguments are known to demonstrate erroneous background assumptions rather than the impossibility of a diagonal object [10]. The present construction indicates that the diagonalization at issue is of this sort. Still, this leaves open why the diagonal argument does not foreclose the possibility of a universal p.r. function. Clarifying this could provide deeper insight into primitive recursion or diagonalization within formal arithmetic.

To reiterate, nothing here suggests that PRA is inconsistent, or that it proves more than it should. On the other hand, the construction might suggest that PRA does not entirely 'avoid universality'—although of course it still lacks universal quantification. And it is far from clear whether the alleged universality affects the familiar use of PRA as a methodological 'safe base' for metamathematical reasoning. Yet some may desire further discussion on the issue, and this is another possible avenue for future research.[11]

References

[1] Curry, H. (1941). A formalization of recursive arithmetic. *American Journal of Mathematics* 63: 263-282.

[2] Cutland, N. (1980) *Computability: an introduction to recursive function theory*. Cambridge: Cambridge University Press.

[3] Gödel, K. (1931). Über Formal Unentscheidbare Sätze der *Principia Mathematica* und Verwandter Systeme I,' *Monatshefte für Mathematik Physik* 38: 173–198. Pagination is from Gödel, K. (1986). *Collected works I. Publications 1929–1936*. S. Feferman et al. (eds.), Oxford: Oxford University Press, pp. 144–195.

[4] Hilbert, D. & Bernays, P. (1934). *Grundlagen der Mathematik*, *Vol. I*, Berlin: Springer.

[5] Kleene, S.C. (1952). *Introduction to metamathematics*. Amsterdam: North Holland Publishing Co.

[6] Nelson, E. (2011). *Elements*. https://arxiv.org/abs/1510.00369

[7] Nelson, E. (2011a). Re: the inconsistency of arithmetic. *The n-Category Café*. https://golem.ph.utexas.edu/category/2011/09/the_inconsistency_of_arithmeti.html#c039590.

[11] My thanks to [redacted] for discussion of issues relevant to this paper. I also express gratitude to an audience at the 2022 meeting of the Australasian Association of Philosophy.

[8] Quine, W. (1951). *Mathematical logic*, revised edition. Cambridge, MA: Harvard University Press.

[9] Robinson, R.M. (1950). An essentially undecidable axiom system. *Proceedings of the International Congress of Mathematics*: 729-730.

[10] Simmons, K. (1993). *Universality and the liar*. Cambridge: Cambridge University Press.

[11] Skolem, T. (1923). Begründung Der Elementaren Arithmetik Durch Die Rekurrierende Denkweise Ohne Anwendung Scheinbarer Veranderlichen Mit Unendlichem Ausdehnungsbereich, *Videnskapsselskapets Skrifter, I. Matematisk-Naturvidenskabelig Klasse*, 6: 1–38. Translated by S. Bauer-Mengelberg as 'The foundations of elementary arithmetic established by the recursive mode of thought, without the use of apparent variables ranging over infinite domains.' In Heijenoort, J. van (ed.). (1967), *From Frege to Gödel: a source book in mathematical logic, 1879–1931*. Cambridge, MA: Harvard University Press, pp. 302-333.

[12] Tao, T. (2011) Re: the inconsistency of arithmetic. *The n-Category Café*. https://golem.ph.utexas.edu/category/2011/09/the_inconsistency_of_arithmeti.html#c039553.