The Characterization of Computation via Predicates And Some Consequences

Duarte Maia

November 22, 2023

Contents

1 Introduction

In this document, I seek to write down the main results I have been looking into in the summer of 2023, and sketches of their proofs. This may or may not be suitable for readers other than myself.

2 The Basic Systems

In the remainder of this document, we will be working over the language denoted L_A, which consists of the symbols 0, 1, $+$, $\times$, and $\leq$. In this language, we introduce two axiomatic systems.

2.1 Weak Peano Arithmetic

Definition 2.1.1. The axiomatic system PA^- is given by (the translation to FOL of) the following statements:

The operations $+$ and $\times$ are associative, commutative, and satisfy the distributive law,
for which 0 and 1 are the identities, (SRing) {ax:sring}
and moreover $(x + z = y + z) \to (x = y)$, and $(xz = yz \wedge z \neq 0) \to (x = y)$.

The binary relation $\leq$ is a total order, which is respected by the operations
(i.e. if $x \leq y$ then $x + z \leq y + z$ and $xz \leq yz$) (Ord) {ax:ord}

$x \leq y$ iff $\exists_z(x + z = y)$, (Def$\leq$) {ax:defleq}

$\forall_x x \geq 0$ (ZMin) {ax:zmin}

$\forall_x(x \neq 0 \leftrightarrow x \geq 1)$. (Disc) {ax:disc}

□

The main power of this definition is that it allows us to carry out concrete computations, in a sense that will be made precise in propositions 2.1.5 and 2.1.6. Before then, however, we need some preliminary results.

Remark 2.1.2. The axioms above may or may not contain some redundancy. Moreover, we will scarcely need the full power of PA^-, and many authors work with far, far weaker systems. However, since we will mostly be interested in PA itself, and PA^- will be mostly used to identify where induction may be foregone or weakened, we do not care too much about ensuring that it is as weak as possible.

Convention. If the letter n denotes a natural number, its presence in a formula should be replaced by a term in L_A which evaluates to n. Note that, by (SRing), PA^- proves that any two such terms are equal.

Convention. The symbol $x < y$ means the obvious: '$x \leq y$, but $x \neq y$'.

{prop:leq}

Proposition 2.1.3. PA^- proves that $x < y$ iff $x + 1 \leq y$.

Proof: We reason in PA^-. Suppose that $x < y$, i.e. $x \leq y$ but $x \neq y$. Then, by (Def$\leq$) we have $y = x + z$ for some z. Moreover, since $x \neq y$, z cannot be zero. Thus, by (Disc), $z \geq 1$. Finally, applying the fact that (by (Ord)) the order is respected by addition, we have (in PA^-) $x + 1 \leq x + z = y$.

Conversely, given that $x + 1 \leq y$, and moreover $x \leq x + 1$, it suffices to show that $x \neq y$. But indeed, suppose that $x = y$. Since $x + 1 \leq y$, we have $y = x + 1 + z$ for some z. Thus, by (SRing) we have $z + 1 = 0$. However, this is impossible: whenever $a + b = 0$, we have $0 \leq a \leq a + b = 0$, whence $a = 0$ by (Ord), and so in particular from $z + 1 = 0$ we get $1 = 0$. Yet, by (Disc), $1 \neq 0$, a contradiction. ■

{prop:leqrep}

Proposition 2.1.4. If n is a natural number, we have

$$\mathrm{PA}^- \vdash \forall_x(x \leq n \leftrightarrow [x = 0 \vee \cdots \vee x = n]). \tag{1}$$

Proof: We reason by (meta-)induction on n. For $n = 0$, the statement is obvious. On the other hand, if the statement is true for a given $n \in \mathbb{N}$, we have, in PA^-,

$$x \leq n + 1 \text{ iff } (x = n + 1 \vee x < n + 1), \tag{2}$$

and by (2.1.3) $x < n + 1$ iff $x + 1 \leq n + 1$. By (Def$\leq$) and (SRing), this is equivalent to $x \leq n$, and by the induction hypothesis this is equivalent to $[x = 0 \vee \cdots \vee x = n]$, which completes the proof. ■

{prop:wpadelt

Proposition 2.1.5. PA^- is complete for (closed) Δ_0 sentences. In particular, if φ is a closed Δ_0 formula, $\mathrm{PA}^- \vdash \varphi$ iff $\mathbb{N} \vDash \varphi$.

Proof: First, we remark that, by proposition 2.1.4, PA^- has 'closed bounded quantifier elimination', in the following sense. If φ is of the form $\forall_{x\leq t}\varphi_0(x)$, then PA^- proves that φ is equivalent to $\varphi_0(0)\vee\cdots\vee\varphi_0(n)$, where n is whatever natural number t evaluates to. Thus, we may recursively eliminate all bounded quantifiers from a Δ_0 sentence, obtaining at the end an equivalent quantifier-free sentence. Hence, it suffices to show that PA^- is complete for atomic sentences. But indeed, an atomic sentence is either of the form $t_1 = t_2$ or $t_1 \leq t_2$, and these can be decided in PA^- by evaluating the numerical value of t_1 and t_2.

(Details are being swept under the rug here, but the bottom line is that cancellation properties may be used to reduce to sentences of the form $t = 0$, $t \leq 0$, or $t \geq 0$, and PA^- is able to prove that the first two hold iff t evaluates numerically to zero, and the latter holds always.)

The second part of the proposition is a direct consequence of completeness for closed Δ_0 formulas, together with the fact that $\mathbb{N} \vDash \mathrm{PA}^-$. ■

{prop:wpasigm

Proposition 2.1.6. PA^- proves all true Σ_1 sentences. More specifically, if $\varphi \equiv \exists_x\varphi_0(x)$ is a closed Σ_1 formula, $\mathrm{PA}^- \vdash \varphi$ iff $\mathbb{N} \vDash \varphi$.

Proof: Evidently, if $\mathrm{PA}^- \vdash \varphi$, we have $\mathbb{N} \vDash \varphi$, so we prove the opposite implication.

Suppose that $\mathbb{N} \vDash \varphi$. Then, there is some natural number n such that $\mathbb{N} \vDash \varphi_0(n)$. Since $\varphi_0(n)$ is a Δ_0 closed sentence, we conclude by proposition 2.1.5 that $\mathrm{PA}^- \vdash \varphi_0(n)$ and so $\mathrm{PA}^- \vdash \exists_x\varphi_0(x)$. ■

2.2 Peano Arithmetic

The system of axioms of PA is obtained by adding the axiom schema of induction to PA^-.

2.3 Adding Function Symbols

For the time being, I will keep this section short and free of proofs, because they are quite technical and not that interesting to write out. Perhaps someday I will do it, because there is one or another thing that is kind of interesting.

The bottom line is the following: Suppose that T is a set of axioms in the language L and $\varphi(\vec{x}, y)$ is a formula such that T proves $\forall_{\vec{x}}\exists^1_y\varphi(\vec{x}, y)$. Then, this formula may be said to represent a function, and we consider adding a function symbol $f(\vec{x})$ to the language (call the new language L'), together with the axiom $\forall_{\vec{x}}\varphi(\vec{x}, f(\vec{x}))$ (call the new set of axioms T'). The first main result is that this adds no power to the language.

{prop:cons1}

Proposition 2.3.1. With L, T, L', and T' as above, there is a way to transform formulas φ in L' into formulas $\hat{\varphi}$ in L, such that:

- From the perspective of T', $\varphi \leftrightarrow \hat{\varphi}$,
- If φ is already in L, then $T \vdash \varphi \leftrightarrow \hat{\varphi}$,
- T' proves φ iff T proves $\hat{\varphi}$.

As a consequence of the above properties, if φ is a formula in L and $T' \vdash \varphi$, then $T \vdash \hat{\varphi}$, hence $T \vdash \varphi$ as well.

The operator $\hat{\cdot}$ described above operates recursively on a formula, being interesting only when it reaches atomic formulas, e.g. $t_1 = t_2$. In this scenario, it will recursively replace any instance of application of f by an instance of φ, for example:

$$\begin{aligned} g(f(x), y) = f(h(0)) \quad \rightsquigarrow \quad & \forall_{\nu_1|\varphi(x,\nu_1)}\, \forall_{\nu_2|\varphi(h(0),\nu_2)}\, g(\nu_1, y) = \nu_2, \\ \text{or } & \exists_{\nu_1|\varphi(x,\nu_1)}\, \exists_{\nu_2|\varphi(h(0),\nu_2)}\, g(\nu_1, y) = \nu_2. \end{aligned} \tag{3}$$

Note: The choice to use a $\forall$ or an $\exists$ is indifferent. This is useful because it means that if φ is Σ_n or Π_n, for $n \geq 1$, we may choose $\hat{\varphi}$ in the same class.

A useful property is that axiom schema of induction is preserved.

{prop:cons2}

Proposition 2.3.2. Suppose that T contains the axiom schema of induction in L. Then T' proves the axiom schema of induction in L'.

Moreover, this conclusion holds if we restrict ourselves to Σ_n or Π_n induction, for $n \geq 1$.

Proof: If $\psi(x, \vec{y})$ is a formula, then by proposition 2.3.1, from the perspective of T' the axiom of induction for ψ is equivalent to the axiom of induction for $\hat{\psi}$. Since T proves the latter, T' does as well, and so it proves induction for ψ. ■

{rmk:delta0}

Remark 2.3.3. The second part of proposition 2.3.2 does not necessarily hold for Δ_0 formulas, because the hat operator adds a quantifier which is not necessarily bounded. However, if one can find a term $t(\vec{x})$ such that $T \vdash \forall_{\vec{x}} \forall_y (\varphi(\vec{x}, y) \to y \leq t(\vec{x}))$, then the hat operator may be made to add only bounded quantifiers. To me, this motivates the definition of Δ_0 function, as one which is defined by a bounded quantifier formula and is itself bounded by a term in the input.

3 Computing in Arithmetic

3.1 Encoding Sequences

{def:seqenc}

Definition 3.1.1. A *sequence encoding* is a pair of formulas, say $\lambda(x, w)$ and $\theta(x, y, z)$, satisfying the following conditions (in the axiomatic system under consideration).

$$\forall_x \exists^1_w \lambda(x, w) \quad \text{(We say } w \text{ is the length of } x\text{, and denote it } \mathrm{len}(x)) \qquad \text{(Len)}$$

$$\exists_x \mathrm{len}(x) = 0 \quad \text{(We call } x \text{ an empty sequence)} \qquad \text{(Eps)} \quad \{\texttt{ax:eps}\}$$

$$\text{Given } x \text{ and } y\text{, there is a unique } z \text{ such that } \theta(x, y, z).$$
$$\text{This } z \text{ is denoted by } (x)_y\text{, and moreover if } y \geq \mathrm{len}(x) \text{ we have } (x)_y = 0. \qquad \text{(Eval)}$$

$$\text{Given } x \text{ and } a\text{, there exists } x' \text{ such that:}$$
$$\mathrm{len}(x') = \mathrm{len}(x) + 1,\ (x')_{\mathrm{len}(x)} = a\text{, and for } y < \mathrm{len}(x),\ (x')_y = (x)_y. \qquad \text{(Ext)} \quad \{\texttt{ax:ext}\}$$

□

An extremely important fact is the following.

{prop:godelle

Proposition 3.1.2 (Gödel's lemma). There exists a Δ_0 sequence encoding in PA. (Also in the sense of remark 2.3.3.)

As it happens, as long as we have sufficient induction, the actual encoding chosen is irrelevant for our purposes, as we will never use anything about the encoding other than definition 3.1.1 (and possibly the fact that λ and θ are Δ_0). Indeed, one could imagine replacing our language by a two-sorted language, of which one sort is natural numbers and the other is sequences, and adjoining the axioms in definition 3.1.1. Then, proposition 3.1.2 would translate to a conservativity result: This new system proves nothing about the natural numbers that PA did not already.

This process of adding a second sort to our language could lead to some issues, namely when it comes to the notion of bounded quantification. However, I believe that it is already not feasible to bound the representatives of sequences, and indeed I think that quantifiers over (numbers that are intended to represent) sequences are basically never bounded. In any case, the main use for sequences is to represent the execution process of a Turing machine (see proposition **??** below), in which event the quantifier for the sequence is unbounded, and thus we need not worry about what it means for a quantifier over a sequence to be bounded.

Remark 3.1.3. If I'm not mistaken, proposition 3.1.2 does not require the full power of PA, but only Σ_1 induction.

3.2 Recursion in PA

Now that we are equipped with the notion of sequences, we are able to add a lot of new function symbols to PA.

{prop:recursi

Proposition 3.2.1. Let $f(x, y)$ be a function in two variables in PA, by which we mean: Let $\varphi(x, y, z)$ be a formula such that PA $\vdash \forall_{x,y}\exists^1_z\varphi(x, y, z)$, and let f be a function symbol adjoined to represent this function. Moreover, let t_0 be a given term in L_A. Then, PA proves that there exists a unique function $g(x)$ such that

$$g(0) = t_0, \tag{4}$$
$$\forall_x\ g(x + 1) = f(x, g(x)). \tag{5}$$

Proof: We define $\psi(x, z)$ by the formula

$$\psi(x, z): \quad \exists_s(\text{len}(s) = x + 1 \wedge (s)_0 = t_0 \wedge \forall_{k<x}(s)_{k+1} = f(x, (s)_k) \wedge (s)_x = z). \tag{6}$$

Then, one shows by induction (using (Eps) and (Ext)) that $\forall_x\exists_z\psi(x, z)$. Moreover, we have uniqueness of z because, by induction on k, any two sequences s of length $x + 1$ satisfying $(s)_0 = t_0$ and $\forall_{k\leq x}(s)_{k+1} = f(x, (s)_k)$ agree for all k. In particular, they will agree at $k = x$, and thus the value of z is uniquely determined from the value of x. ■

Remark 3.2.2. Proposition 3.2.1 is not nearly as general as it could be; it is intended rather to sketch the kind of thing that is possible, without being overburdened with excessive abstraction. A small modification of it, for example, allows us to prove the existence and uniqueness of a binary function representing exponentiation, which is determined by the property that $x^0 = 1$ and $x^{y+1} = x^y \times x$.

Remark 3.2.3. A possible culmination, by which I mean 'most general yet reasonable possible version', of proposition 3.2.1 is the statement that the set of functions definable in PA are closed under the operations used to generate the primitive recursive functions.

I am taking care to avoid the phrasing 'PA proves the existence of all primitive recursive functions', because given a PR function there is more than one way to construct over $\mathbb{N}$ it using the operations. In particular, one may replace $f(x)$ by the (*a posteriori* PR) function $g(x)$ given by 'if x represents a proof that PA is inconsistent, return 0, otherwise return $f(x)$'. Since PA is (hopefully) consistent, f and g agree over the natural numbers. However, *a posteriori* we know that there exist models of PA where such an x does exist, and on those models f and g will furnish two distinct extensions of the same PR function.

Remark 3.2.4. [To do, write out how this looks in PA$^-$.] [Future self says: What?]

Using Gödel's lemma, it is possible to encode the code for a Turing machine as a natural number, and likewise for the state of a Turing machine. Moreover, it is possible to do so in such a way that the function step(*tape*, *code*) is PR. Finally, we may also encode a 'step sequence', and both the predicate 'is this step sequence a valid execution sequence for this code', and the function which extracts the output 'number' from a tape, are also PR. As such, given a Turing machine encoded by the number $n \in \mathbb{N}$, we may attempt to encode the partial computable function corresponding to it by the formula

$$\varphi(x, y) \equiv \exists_s\ (s \text{ is a running sequence for } n, \text{ with input } x \text{ and output } y). \tag{7}$$

In order to better understand the subtleties of this definition, let us establish some notation.

Definition 3.2.5. We will now enumerate some Primitive Recursive functions on the natural numbers, and give them a name. In the sequel, we assume that these names have been added to the language as function symbols, axiomatized via some way to represent them in the Primitive Recursive schema.[1]

[1] In nonstandard models, distinct representations may lead to distinct functions. However, these representation will always agree over $\mathbb{N}$.

- step(n,t): Where t represents a tape state, and n a Turing machine, this function symbol outputs (a number representing) the next state of the tape upon executing a step of the Turing machine n,
- tapefrominput(x): This function symbol outputs (a number representing) a tape containing nothing but x many ones after the pointer,
- halted(t): Where t represents a tape state, this predicate will evaluate as True if t is in the halted state (i.e. computation has terminated) and False otherwise,
- tapetooutput(t): Where t represents a tape state, this function symbol extracts a natural number as output from the tape, in such a manner that tapetooutput(tapefrominput(x)) $= x+1$ holds for all x. The value 0 is reserved as an error code, in the event that no discernible output may be obtained.

Implicit in the above definitions is the notion that n and t represent valid Turing machines and tapes, respectively. This verification is also PR, and we may assume that given invalid input, some fixed error code (e.g. zero) is produced as output. □

{def:phin}

Definition 3.2.6. Let $f\colon \mathbb{N} \rightharpoonup \mathbb{N}$ be a partial computable function, and let n be a code for a Turing machine which computes f. To this Turing machine we associate the Σ_1 formula

$$\varphi(n,x,y) \equiv \exists_s \begin{pmatrix} (s)_0 = \text{tapefrominput}(x) \\ \wedge\, [\forall_{i<\text{len}(s)-1}(s)_{i+1} = \text{step}(n,(s)_i)] \\ \wedge\ \text{halted}((s)_{\text{len}(s)-1}) \\ \wedge\, y+1 = \text{tapetooutput}((s)_{\text{len}(s)-1}) \end{pmatrix} \tag{8}$$

□

Proposition 3.2.7. Let f, n, and φ be as in definition 3.2.6, and let $a \in \mathbb{N}$ be in the domain of f. Then, $\text{PA}^- \vdash \varphi(n,a,f(a))$.

Proof: Direct consequence of proposition 2.1.6. ■

Remark 3.2.8. It is not the case that, as stated, the output (from the POV of PA^-) is unique, as there may be nonstandard execution sequences s which give wildly different results. This can be remedied by ensuring that s is a standard element.

In general it is not possible to make this demand, but in this case we have the following work-around: We demand that s is a minimal valid execution sequence. Indeed, in the scenario where a is in the domain of f, a valid (standard) execution sequence exists by hypothesis, and any nonstandard execution sequence will be greater than it (by 2.1.4), so picking out the minimal sequence ensures that it is standard.

{prop:psin}

Proposition 3.2.9. Define the formula $\psi(\eta,x,y)$ given by

$$\psi(\eta,x,y) \equiv \exists_s \begin{pmatrix} (s)_0 = \text{tapefrominput}(x) \\ \wedge\, [\forall_{i<\text{len}(s)-1}(s)_{i+1} = \text{step}(\eta,(s)_i)] \\ \wedge\ \text{halted}((s)_{\text{len}(s)-1}) \\ \wedge\ s \text{ is minimal under the above conditions} \\ \wedge\, y+1 = \text{tapetooutput}((s)_{\text{len}(s)-1}) \end{pmatrix} \tag{9}$$

Now, let $f\colon \mathbb{N} \rightharpoonup \mathbb{N}$ be a partial computable function, and n a code for a Turing machine computing f. Then, for every natural number a in the domain of f,

$$\left(\text{PA}^- \vdash \psi(n,a,b)\right) \text{ iff } b = f(a). \tag{10}$$

Proof: Let us briefly introduce the Δ_0 formula $\psi_0(\eta, x, y, s)$ such that $\psi(\eta, x, y) \equiv \exists_s \psi_0(\eta, x, y, s)$.

($\leftarrow$) If a is in the domain of f, there is an execution sequence for n, represented by a standard natural number s, which starts in the state tapefrominput(a) and ends in the state corresponding to the output $f(a)$. This execution sequence is (or may be chosen to be, depending on uniqueness properties of our encoding) minimal, hence (by 2.1.5) $\mathrm{PA}^- \vdash \psi_0(n, a, f(a), s)$.

($\rightarrow$) Note that the expression for $\psi_0(\eta, x, y, s)$ contains the line $y = \mathrm{tapetooutput}(s)$. Moreover, PA^- proves that, given η and x, there is at most one value of s making ψ_0 hold. Thus, we conclude that there is at most one value of y making $\psi(n, a, y)$ hold, as we've seen $f(a)$ is such a value, and so b must be the same. ■

{cor:psin}

Corollary 3.2.10. Let f, ψ, n as above. Then, for all a in the domain of f,

$$\mathrm{PA}^- \vdash \forall_y(\psi(n, a, y) \leftrightarrow y = f(a)). \tag{11}$$

Proof: We claim that PA^- proves that $\forall_\eta \forall_x \exists_y^{\leq 1} \psi(\eta, x, y)$. Perhaps this is not true for the s as stated and requires tinkering with the encoding a bit, but if we encode tapes in such a way that tapetooutput is simple enough (this could be as easy as making our encoding of Turing machines be a pair whose second element is the result of tapetooutput) we can see that the line $y + 1 = \mathrm{tapetooutput}((s)_{\mathrm{len}(s)-1})$ uniquely determines y given s (from the perspective of PA^-.

Once this is done, note that (reasoning in FOL) from $\forall_x \exists_y^{\leq 1} P(x, y)$ and $P(a, b)$, one obtains $\forall_y(P(a, y) \leftrightarrow y = b)$. ■

Remark 3.2.11. If a is not in the domain of f, it is unclear to me what PA^- thinks of the sentence $\exists_y \psi(n, a, y)$. By proposition 2.1.6, PA^- certainly cannot prove it true, but in general there is no reason to believe that it can prove it false either.

4 The First Incompleteness Theorem

4.1 Self-Reference

A big ingredient in proving the first incompleteness theorem is the capacity to encode a sentence saying 'this sentence is false', or P: 'P is false'. However, sentences cannot directly be self-referential, so we must find a way to work around this.

A way to get around this issue is to note that, as we've seen before, we can 'encode computer programs inside arithmetic'. Moreover, it is known that we can make computer programs that print themselves, also known as Quines. As such, we might expect to encode inside arithmetic a formula that is able to refer to a code for itself in some indirect way. Formally, this is reflected in the proposition 4.1.2 below.

Definition 4.1.1. We assume a fixed way to encode formulas in the language of Arithmetic as natural numbers in a way amenable to computable operations, e.g. substituting variables for expressions, checking for closedness, etc. The number encoding a formula is called its *Gödel number*. If φ is a formula, its Gödel number is written $\ulcorner \varphi \urcorner$. It will also be useful to have a way to denote the reverse operation: If n is a number, $\llcorner n \lrcorner$ symbolizes the formula encoded by n. □

{prop:selfref}

Proposition 4.1.2. Let $\theta(x)$ be a formula with one free variable in the language of arithmetic. Then, there is a sentence P such that

$$\mathrm{PA}^- \vdash P \leftrightarrow \theta(\ulcorner P \urcorner). \tag{12}$$

Moreover, if θ is Π_1, then P is also Π_1.

Remark 4.1.3. This will allow us to find a sentence P that cannot be proven nor disproven in a given theory T, roughly by setting $\theta(x)$: T thinks $\llcorner x \lrcorner$ is false. Then, if T is able to prove P, it will hold that (according to T) P is false, which is a contradiction, and a similar argument works if T were able to prove $\neg P$.

Of course, the truth is a lot more technical than the above paragraph.

The remainder of this section consists of the proof of proposition 4.1.2.

The main idea to construct P is the following. We shall have a 'template formula' $\tau(x)$, and x being intended as some data from which the formula P will reconstruct itself. More precisely, we will set $P \equiv \tau(\ulcorner\tau\urcorner)$, and the idea is that τ is able to take $x = \ulcorner\tau\urcorner$ and reconstruct itself by considering $\llcorner x \lrcorner(x)$.

To be a little more precise, let $f\colon \mathbb{N} \to \mathbb{N}$ be given by

$$f(n) = \ulcorner \llcorner n \lrcorner(n) \urcorner, \tag{13}$$

where $\llcorner n \lrcorner(n)$ represents replacing some fixed variable in $\llcorner n \lrcorner$, say x, by (some expression evaluating to) the numerical value of n.

{rmk:rug}

Remark 4.1.4. We are sweeping under the rug details such as 'what if x is not the Gödel number of any formula?' In our view, these details are easily dealt with (pick some convention for an error code) and do not add anything to the exposition other than extra technical load.

Now, under any reasonable choice of Gödel encoding, the function f is computable and (see remark 4.1.4) total. Therefore, it is represented in PA^- by some Σ_1 formula $\varphi(x, y)$ as in proposition 3.2.9. We now have all the tools that we need to reverse-engineer the definition of P.

We want that, under PA^-, $P \leftrightarrow \theta(\ulcorner P \urcorner)$. Assuming that P is constructed by feeding the Gödel number of some template formula τ to itself, this is the same as requiring that

$$\tau(\ulcorner\tau\urcorner) \leftrightarrow \theta(\ulcorner \tau(\ulcorner\tau\urcorner) \urcorner), \tag{14}$$

or equivalently $\tau(\ulcorner\tau\urcorner) \leftrightarrow \theta(f(\ulcorner\tau\urcorner))$. This suggests setting $\tau(x)\colon \theta(f(x))$, but this is invalid because f is not part of the language of arithmetic. So we replace f by the representation given by φ, and set

$$\tau(x)\colon \forall_y(\varphi(x, y) \to \theta(y)), \quad P\colon \tau(\ulcorner\tau\urcorner). \tag{15}$$

We now prove that P does what is desired of it. Note that $\ulcorner P \urcorner = f(\ulcorner\tau\urcorner)$, so it is the case that $\mathrm{PA}^- \vdash \varphi(\ulcorner\tau\urcorner, \ulcorner P \urcorner)$. Moreover, by corollary 3.2.10 PA^- proves $\varphi(\ulcorner\tau\urcorner, y) \leftrightarrow y = \ulcorner P \urcorner$, hence

$$\mathrm{PA}^- \vdash \big[\forall_y(\varphi(\ulcorner\tau\urcorner, y) \to \theta(y))\big] \leftrightarrow \big[\forall_y(y = \ulcorner P \urcorner \to \theta(y))\big]. \tag{16}$$

The left-hand side is by definition P, and the right-hand side is evidently equivalent to $\theta(\ulcorner P \urcorner)$, and so the proof of proposition 4.1.2 is complete.

Remark 4.1.5. While in the above exposition we assumed that θ is a formula in the language of arithmetic, this is not a necessity. Indeed, it is actively undesirable when applying this theorem to more general contexts, such as ZF or second order arithmetic.

We remark that θ was effectively only used as a black box. As such, the only effective requirements on the underlying system are: The underlying language $\mathcal{L}$ must be computable in some sense (so that Gödel numbering makes sense), and there must be (possibly composite) predicates: 'isNatural', 'isSuccessor', 'isZero', 'isSumOf', 'isProductOf', and $\leq$, such that the underlying theory T proves (the translation to relational languages of) the axioms of PA^-, relativized to the class of elements satisfying isNatural.

The resulting translation will likely wreak havoc on the quantifier complexity of the formula, though one wonders to what extent complexity is meaningful to the underlying theory. This is strange because complexity, and in particular the completeness of PA^- for bounded quantifier sentences, was essential on the path to where we stand, and it feels like the results we have gotten to should abstract to a context where complexity may be meaningless or useless.

One may try to define the complexity of a formula using only the predicates above, corresponding to a translation of the language of arithmetic to a relational language. This raises the question of how to define complexity such that that the formula $\varphi(x)\colon \exists_y(\mathrm{isSuc}(y, x) \wedge \exists_z(\mathrm{isSuc}(z, y) \wedge \mathrm{isZero}(z)))$ is of the lowest level of complexity, being as simple as $x = S(S(0))$.

(*Post Scriptum:* It is my expectation after some discussion with Denis Hirschfeldt that, while the notion of Δ_0 formula is severely weakened, all other steps on the hierarchy should stay intact.)

Anyhow, according to the above paragraphs, one obtains that the results that follow will hold for any language and theory which is strong enough to 'express arithmetic to the level of PA^-'. This is not shocking for incompleteness, but more surprisingly undefinability of truth (section 4.4) will hold even for languages and theories with additional expressive power beyond that of arithmetic.

4.2 The Gödel Sentence

Now that we are able to construct self-referential sentences (or, perhaps more precisely, sentences that are able to refer to their own Gödel number), we begin to conceive how one may write a sentence P saying 'P is false'. However, in a precise technical sense, it is not possible to write a formula which recovers the truth value of P from its Gödel number; see section 4.4 below. However, there is a 'close enough' analogue: it *is* possible to encode the notion of proof, so we can consider a sentence P: P cannot be proven.

More precisely, let T be a theory in the language of arithmetic. Then, *under the assumption that T can be axiomatized by a recursively enumerable set*, there is a partial computable function $T\mathrm{Proves}(x)$, which halts if $T \vdash \llcorner x \lrcorner$, and runs forever otherwise.

Yet more specifically, there is a total computable predicate $T\mathrm{ProofOf}(q, x)$, which (under some suitable encoding of proof) checks whether q is a proof of $T \vdash \llcorner x \lrcorner$. Then, we set $T\mathrm{Proves}(x)$ to mean $\exists_q T\mathrm{ProofOf}(q, x)$. Remarkably, if we are very particular about the form of the encoding, we can ensure that $T\mathrm{ProofOf}(q, x)$ is a bounded quantifier formula. (This is an application of a rather general principle; Since $T\mathrm{ProofOf}$ is (under a reasonable encoding) recursive, it may be encoded as a Σ_1 formula $\exists_s \varphi(q, x, s)$. If we change our encoding so that q is replaced by $\langle q, s\rangle$, we can now represent $T\mathrm{ProofOf}$ in a bounded quantifier way.)

Remark 4.2.1. Since T is assumed to be axiomatized by a recursively *enumerable* set of axioms, and not necessarily recursive, one may be concerned about whether $T\mathrm{ProofOf}$ is total recursive. A naïve implementation might result in a partial recursive function: Loop over the statements, check for each of them whether it is an application of a logical rule, or a logical axiom, or an axiom of T, and this last step could hypothetically loop forever. However, it is possible to play with the encoding to make the function total.

Let $f\colon \mathbb{N} \to \mathbb{N}$ be an enumeration of the axioms of T. Then, we demand from our encoding of proofs that, when an axiom A of T is invoked, we include the value of n that witnesses that the axiom is included in the enumeration, i.e. some value of n such that $f(n) = \ulcorner A \urcorner$.

That said, we can now encode the sentence P: 'T proves $\neg P$', by applying proposition 4.1.2 to

$$\theta(x)\colon \exists_q T\mathrm{ProofOf}(q, \ulcorner \neg \llcorner x \lrcorner \urcorner) \equiv T\mathrm{Proves}(\ulcorner \neg \llcorner x \lrcorner \urcorner). \tag{17}$$

Remark 4.2.2. There are two obvious ways to rewrite 'P is false' in terms of provability: Either 'T does not prove P', or 'T proves $\neg P$'. We chose the latter alternative. However, I think both alternatives yield similar results, and it does not matter which we take.

{prop:inc1g}

Proposition 4.2.3. Let T be a recursively enumerably axiomatizable theory in the language of arithmetic, and construct P such that $\mathrm{PA}^- \vdash P \leftrightarrow T\mathrm{Proves}(\ulcorner \neg P \urcorner)$. This is called *the Gödel sentence of T*.

Then, assuming that $\mathrm{PA}^- \subseteq T \subseteq \mathrm{Th}(\mathbb{N})$, T can neither prove nor disprove P.

{rmk:omega}

Remark 4.2.4. The statement that $\mathrm{PA}^- \subseteq T$ is a requirement that T is 'at least a little bit strong', in order to be able to reason about itself. This is extremely necessary; theories with weak enough reasoning may very well be complete and computable.

The statement that $T \subseteq \mathrm{Th}(\mathbb{N})$ is a requirement that T doesn't say anything false about the natural numbers. This requirement is much less necessary. For one, we will only use the much weaker statement about T: When $\varphi(x)$ is a Δ_0 formula, T cannot at the same time prove all the following sentences: $\varphi(0), \varphi(1), \varphi(2), \ldots$ and $\exists_x \neg\varphi(x)$. This is known as ω-consistency of T for Δ_0 formulas in one variable. Since T contains PA^-, which is complete for Δ_0 sentences, this property may be rewritten as: If $T \vdash \exists_x \psi(x)$, there is some natural number n such that $T \vdash \psi(n)$, or equivalently that $\mathrm{PA}^- \vdash \psi(n)$ or that $\mathbb{N} \vDash \psi(n)$.

In section 4.3 we will find a different sentence, for which the equivalent of proposition 4.2.3 may be greatly relaxed, requiring only that $\mathrm{PA}^- \subseteq T$ and that T is consistent.

Proof (Proposition 4.2.3): Suppose first for contradiction that $T \vdash \neg P$. Then, if n is a natural number encoding a proof of this fact, we the predicate $T\text{ProofOf}(n, \ulcorner \neg P \urcorner)$ evaluates as true. Therefore, by proposition 2.1.5, $\text{PA}^- \vdash T\text{ProofOf}(n, \ulcorner \neg P \urcorner)$, hence $\text{PA}^- \vdash T\text{Proves}(\ulcorner \neg P \urcorner)$, thus $\text{PA}^- \vdash P$ and so $T \vdash P$, a contradiction.

Suppose now that $T \vdash P$. Then, since $\text{PA}^- \subseteq T$, we obtain that $T \vdash T\text{Proves}(\ulcorner \neg P \urcorner)$, or equivalently $T \vdash \exists_q T\text{ProofOf}(q, \ulcorner \neg P \urcorner)$. Now, by remark 4.2.4, we obtain that there is some natural number n such that $\mathbb{N} \vDash T\text{ProofOf}(n, \ulcorner \neg P \urcorner)$, and so n encodes an 'honest' proof that $T \vdash \neg P$. This is a contradiction. ■

Corollary 4.2.5. PA is incomplete.

4.3 The Rosser Sentence

{sec:rosser}

The main place for improvement in proposition 4.2.3 is the stringent requirement that (as per remark 4.2.4) if the theory T proves $\exists_x \varphi(x)$, for φ in Δ_0, then it really is the case (from the perspective of $\mathbb{N}$) that there exists someone satisfying $\varphi(x)$. This is necessary to go from 'T proves that there is a proof of $\neg P$' to 'There is really a proof of $\neg P$'. If T does not satisfy this, it may be the case that T 'sees' a proof of $\neg P$, encoded as an element of its universe n, but that from our outside perspective n is a non-standard number and therefore cannot be turned into a proof of $T \vdash \neg P$.

The idea behind the Rosser sentence is to somehow ensure that the proof n is really a standard integer. However, as we will see later in section 6, there is no first-order way to express that x is 'standard'. Therefore, we have to resort to imperfect ways of ensuring that x is standard, and one such way is the following: Require that x be less than some y, which is known to be standard. This works by proposition 2.1.4.

The Rosser sentence is now in sight. Suppose that T proves P, where P is the sentence 'T proves $\neg P$', plus some bells and whistles. Then, we want to ensure that this proof of $\neg P$ is a standard number, so we better say that it is smaller than something known to be standard. Well, we do have a 'standard' proof m of $T \vdash P$. So how about we let P be 'There is a proof of $T \vdash \neg P$, and it is smaller (as a number) than any proof of $T \vdash P$'? Then in particular this proof of $\neg P$ is smaller than m, and hence standard.

{prop:inc1r}

Proposition 4.3.1. Let T be a recursively enumerably axiomatizable theory in the language of arithmetic, and construct P such that

$$\text{PA}^- \vdash P \leftrightarrow \left(\exists_q \left(T\text{ProofOf}(q, \ulcorner \neg P \urcorner) \wedge \forall_{p<q} \neg T\text{ProofOf}(p, \ulcorner P \urcorner)\right)\right). \tag{18}$$

This is called *the Rosser sentence of* T.

Then, assuming that $\text{PA}^- \subseteq T$ and that T is consistent, T can neither prove nor disprove P.

Proof: Suppose that T proves $\neg P$. Then, there is some proof (encoded by) n of $T \vdash \neg P$, and since T is consistent there is no (standard) natural number m such that $\mathbb{N} \vDash T\text{ProofOf}(m, \ulcorner P \urcorner)$. In particular, since $T\text{ProofOf}$ is Δ_0, $\text{PA}^- \vdash \forall_{p<n} \neg T\text{ProofOf}(p, \ulcorner P \urcorner)$, and from here one easily proves $T \vdash P$, a contradiction.

Suppose now that T proves P; let n encode such a proof. Then, $\text{PA}^- \vdash T\text{ProofOf}(n, \ulcorner P \urcorner)$, and therefore

$$\text{PA}^- \vdash \left(\forall_{p<q} \neg T\text{ProofOf}(p, \ulcorner P \urcorner)\right) \to q \leq n. \tag{19}$$

Therefore, since $T \vdash P$, it proves $\exists_q \left(T\text{ProofOf}(q, \ulcorner \neg P \urcorner) \wedge q \leq n\right)$, and so, since T contains PA^-, there is some $m \leq n$ such that $\mathbb{N} \vDash T\text{ProofOf}(m, \ulcorner \neg P \urcorner)$, and so $T \vdash \neg P$, a contradiction. ■

The following proposition justifies why we care about this, perhaps at first mild-seeming, generalization of proposition 4.2.3. After all, why should we care about theories which extend PA^- but contain falsehoods about the natural numbers?

Corollary 4.3.2. Any computable theory T in the language of arithmetic which contains PA^- has $2^{\aleph_0}$ consistent extensions.

As a consequence, PA has continuum many distinct countable models.

Proof: Build an infinite binary tree as follows: The base node contains T. For each node, which corresponds to an extension T_s of T by finitely many axioms (and therefore is recursively enumerably axiomatizable), pick a sentence P which T_s neither proves nor disproves. Then, create two branches for this node, one by adding P and the other by adding $\neg P$ to the list of axioms.

The resulting tree has at each node a consistent extension of T, and any two nodes at the same depth will explicitly disagree on some statement. Therefore, the continuum-many paths on the tree will each provide a distinct consistent extension of T. ■

4.4 Undefinability of Truth

{prop:tarski}

Proposition 4.4.1 (Tarski's Undefinability of Truth). Let M be a model of PA^- over the language of arithmetic. Then, there is no formula $\theta(x)$ (in the language of arithmetic) such that, for all sentences φ, φ holds in M iff $\theta(\ulcorner\varphi\urcorner)$ holds in M.

Proof: Suppose that θ were such a formula. Apply proposition 4.1.2 to obtain a sentence P such that $\mathrm{PA}^- \vdash P \leftrightarrow \neg\theta(\ulcorner P\urcorner)$. This provides an obvious contradiction, regardless of whether or not P holds in M.■

5 Definability of (Some) Truth

As we saw in proposition 4.4.1, there is no formula which successfuly 'interprets the truth of' arbitrary Gödel numbers of formulas. However, as we will see in this section, under PA, for any finite level of complexity (more precisely, for any finite place in the arithmetic hierarchy) there is indeed a formula that embodies truth in that complexity.

5.1 Bounded Quantifier Truth

The goal of this section is to establish the existence of a predicate $\mathrm{Sat}_{\Delta_0}(x, y)$ which represents bounded quantifier truth in the following way.

{thm:satdelta0

Theorem 5.1.1. There exists a $\Delta_1(\mathrm{PA})$ formula $\mathrm{Sat}_{\Delta_0}(x, y)$, in the language of arithmetic, such that, whenever φ is a bounded quantifier formula in the language of arithmetic in n free variables, say $y_1, \ldots, y_n$, we have

$$\mathrm{PA} \vdash \forall_y(\mathrm{Sat}_{\Delta_0}(\ulcorner\varphi\urcorner, y) \leftrightarrow \varphi((y)_1, \ldots, (y)_n)). \tag{20}$$

{eq:satdelta0

The main idea should not come across as a surprise: We want to define Sat_{Δ_0} inductively in the (Gödel name of the) formula φ. However, it is not evident that this would map to a well-defined formula in the language of arithmetic.

We recall from proposition 3.2.1 that PA is able to express functions (and hence predicates) by primitive recursion on pre-existing functions. Moreover, while in principle the principle of primitive recursion allows you only to use $g(x)$ when computing $g(x+1)$, if you make your function output not just the result you want, but instead a sequence encoding all the results obtained thus far, you may actually assume that you have access to all prior results. This is quite useful when defining functions by induction on formula structure, as the results you will need to refer to when computing the result for $\ulcorner\varphi\urcorner$ are in general much smaller than $\ulcorner\varphi\urcorner - 1$.

To be a bit more precise, here is a modified version of proposition 3.2.1 to take the previous paragraph into account.

{prop:recursi

Proposition 5.1.2. Let $f(x, y)$ be a function in two variables in PA, as in proposition 3.2.1. Moreover, let t_0 be a given term in L_A. Then, PA proves that there exists a unique function $g(x)$ such that

$$g(0) = t_0, \tag{21}$$

$$\forall_z\ g(z+1) = f(z, a(z)), \tag{22}$$

where $a(z)$ is any sequence of length at least $z+1$ such that, for $y \leq z$, $(a(z))_y = g(y)$.

Moreover, if f is represented by a Σ_1, Π_1 or Δ_1(PA) predicate, so is g.

The goal is now to find the appropriate recursion function $f(z, a)$. The input z represents the pair $\langle x, y\rangle$, with x 'representing the Gödel number of a formula', and y the list of inputs to it. Moreover, $\langle\rangle$ is an appropriate pairing function, e.g. $(x+y)^2 + y$. For our purposes, we need only that it and the projections are not complex (in our case, Δ_0), and that it is monotone in each of its arguments.

Now, we begin describing the behavior of $f(\langle x, y\rangle, a)$. First, we check if x is the Gödel number of a formula; there is a PR way to do so for standard values of x, and we will never care about what happens for nonstandard values.

Now, if x is indeed the Gödel number of some φ, we do cases on its structure [depending on the encoding]. For example, if φ is $\varphi_0 \wedge \varphi_1$, we define f via (in this case) $a_{\langle \ulcorner \varphi_0 \urcorner, y\rangle} \times a_{\langle \ulcorner \varphi_1 \urcorner, y\rangle}$. There are only two tricky cases to consider: The case where φ is an atomic formula, and the case where φ starts with a (bounded) quantifier.

Let us first consider the case where φ is of the form $\forall_{x<t(y_1,\ldots,y_n)}\psi(x, y_1, \ldots, y_n)$. Without loss of generality, assume that x is y_{n+1}; this is because there is a PR way to replace every instance of x by y_{n+1} and vice versa. Then, we would like to set $f(\langle \ulcorner\varphi\urcorner, y\rangle, a)$ to be given by (this should be defined via an appropriate primitive recursion as a function of a, t, y, and $\ulcorner\psi\urcorner$)

$$f(\langle \ulcorner\varphi\urcorner, y\rangle, a) \stackrel{?}{=} \prod_{i=0}^{\mathrm{eval}(t,y)} (a)_{\langle \ulcorner\psi\urcorner, (y,i)\rangle}. \tag{23}$$

The first problem the reader might notice is the undefined expression $\mathrm{eval}(t, y)$, which we'll define soon. The second problem, and most crucial one, is that we have no guarantee that $\langle \ulcorner\psi\urcorner, (y, i)\rangle$ is less than $\langle \ulcorner\varphi\urcorner, y\rangle$.

To fix the latter issue, we apply the following trick. Given a bounded quantifier formula, as well as parameters for it, it is possible to compute in a primitive recursive way a universal upper bound for all variables that will be iterated over in the course of checking the veracity of the formula. Indeed, for every quantifier $\forall_{y_{k+1}<t}$, where t is a term depending on previously seen variables $y_1, \ldots, y_k$, we evaluate t on the maximal possible value of the variables we've seen so far (still pending the definition of eval). This uses the fact that S, $+$, and $\times$ are monotone in their arguments, and thus any term in the language of arithmetic will be as well.

Now we may proceed with our definition of $\mathrm{Sat}_{\Delta_0}(x, y)$. We begin by processing the formula represented by x, and constructing a tuple $\bar{y} = (y, M, \ldots, M)$, where M is an upper bound for all the variables as per the previous paragraph, and it is added once for every new variable that will be considered in the subsequent iteration. Then, when recursing on the formula structure and coming across a new quantifier $\forall_{y_{n+1}<t(y_1,\ldots,y_n)}\psi$, we recurse down from $p_0 = \langle \ulcorner\varphi\urcorner, (y, M, M, \ldots, M)\rangle$ to $p_1 = \langle \ulcorner\psi\urcorner, (y, y_{n+1}, M, \ldots, M)\rangle$, for all admissible values of y_{n+1}. Note: This may require changing the encoding of sequences to ensure that our encoding is monotone in each coordinate. An example of such an encoding would be one via prime factorization. We also assume that our Gödel numbering is monotone in formula structure, but this assumption will hold of any reasonable encoding of formulas. Once this is done, we are guaranteed by monotonicity that $p_1 < p_0$, hence the recursion will function all the way down until the base case.

So, let us finally tackle the case where φ is an atomic formula, so either $t_1 < t_2$ or $t_1 = t_2$. It is evident that to solve either case it suffices to have a way to evaluate a term t on a given assignment of parameters. This boils down to an easier version of the recursion we were doing before: Given a pair $\langle \ulcorner t\urcorner, y\rangle$, we recurse on the structure of t, applying the evaluation function to subterms. This is guaranteed to work because (assumption) the Gödel numbering decreases when passing to subformulas, and we will never need to modify y.

There are two things left to do. First, we would like to control the complexity of the predicate we have wound up defining. Second, we would like to sketch a proof that the resulting predicate satisfies (25).

For the complexity: It should be noted that all functions we have defined above were done so in a primitive-recursive-like way. We hesitate to say that they're *actually* primitive recursive, because we're taking care to make them work with nonstandard numbers, but for sure there are only three big tools

we're using: Primitive recursion as in Theorem 5.1.2, definitions 'by cases', and compositions of previously defined things. Moreover, none of these will make the complexity of previously defined things jump up above $\Delta_1(\mathrm{PA})$, so we conclude that $\mathrm{Sat}_{\Delta_0}(x,y)$ as we've defined it is a $\Delta_1(\mathrm{PA})$ predicate.

Now, let's verify that equation (25) holds. This proof is performed by induction on the formula structure of φ, boiling down to proving rules like $\mathrm{PA} \vdash \mathrm{Sat}_{\Delta_0}(\ulcorner \varphi_1 \wedge \varphi_2 \urcorner, y) \leftrightarrow (\mathrm{Sat}_{\Delta_0}(\ulcorner \varphi_1 \urcorner, y) \wedge \mathrm{Sat}_{\Delta_0}(\ulcorner \varphi_2 \urcorner, y))$, most of which turn out to be true by virtue of how we set up the recursion for Sat_{Δ_0}. The least trivial ones are the rule for quantifiers, and the rule for atomic formulas.

For quantifiers, we note that our definition allows us to prove (something like)

$$\mathrm{PA} \vdash \mathrm{Sat}_{\Delta_0}(\ulcorner \forall_{x<t(\vec{y})} \psi(x,\vec{y}) \urcorner, y) \leftrightarrow \forall_{x<\mathrm{eval}(\ulcorner t \urcorner, y)} \mathrm{Sat}_{\Delta_0}(\ulcorner \psi \urcorner, (y,x)). \tag{24}$$

Finally, and crucial for both quantified formulas and for atomic formulas, we can show by induction on the structure of the term t that $\mathrm{PA} \vdash \mathrm{eval}(\ulcorner t \urcorner, y) = t((y)_1, \ldots, (y)_n)$.

Anyway, a lot of details are being omitted or even misrepresented. But going the whole five miles would be a lot of work. Maybe someday I'll improve this exposition. If the reader is interested, [1] goes into great detail on this subject in chapter 9. I understood very little of it.

5.2 Σ_n and Π_n Truth

Fortunately, defining bounded quantifier truth was the hard part. We can now define Σ_n and Π_n truth inductively.

{thm:satgeq1}

Theorem 5.2.1. Let $n \geq 1$. Then, there exists a $\Sigma_n(\mathrm{PA})$ formula $\mathrm{Sat}_{\Sigma_n}(x,y)$, in the language of arithmetic, such that, whenever φ is a Σ_n formula in the language of arithmetic in n free variables, say $y_1, \ldots, y_n$, we have

$$\mathrm{PA} \vdash \forall_y(\mathrm{Sat}_{\Sigma_n}(\ulcorner \varphi \urcorner, y) \leftrightarrow \varphi((y)_1, \ldots, (y)_n)). \tag{25}$$

{eq:satdelta0

A similar theorem holds if Σ is replaced by Π.

Proof: We simply show it for Σ_1. The remainder, as well as Π, is obtained by an analogous inductive process.

Set $\mathrm{Sat}_{\Sigma_1}(x,y)$ to be something of the following form. First, check (in a primitive recursive way) if x is the Gödel number of some formula $\exists_z \varphi(\vec{y}, z)$, where φ is a bounded quantifier formula. Then, evaluate $\exists_z \mathrm{Sat}_{\Delta_0}(\ulcorner \varphi \urcorner, (y,z))$. Since Sat_{Δ_0} is $\Delta_1(\mathrm{PA})$, the result is (mod logical equivalence) a Σ_1 formula. ■

6 The Complexity of Nonstandard Theorems

{sec:nonstd}

In the following, we study nonstandard models of PA. That is, we look at countable models $M \models \mathrm{PA}$ which are not (necessarily) the natural numbers.

It is worth noting that any model of PA, even the nonstandard ones, will contain a copy of $\mathbb{N}$ as an initial segment. In fact, this holds of any model of PA^-: The set of elements which may be expressed as $S^k(0)$ for some natural number k forms a subset of any model of PA^-, and PA^- is strong enough to ensure that this subset behaves as the natural numbers expect it to, in the sense that e.g. $\mathrm{PA}^- \vdash S^k(0) + S^\ell(0) = S^{k+\ell}(0)$. We will often identify this initial segment of any such model with $\mathbb{N}$ without further comment.

6.1 Overspill and The Standard System

Overspill is a very useful technique that allows us to extend (usually trivial) finite results to nonstandard elements, thereby recovering infinite information.

{lem:overspil

Lemma 6.1.1 (Overspill). Let M be a nonstandard element of PA. Then, there is no formula $\varphi(x)$ which identifies the natural numbers. In particular, if $M \models \varphi[n]$ for all natural numbers n, then $M \models \varphi[a]$ for some $a \in M \setminus \mathbb{N}$.

This result also holds with parameters: If $\vec{b} \in M$ and $M \models \varphi[n, \vec{b}]$ for all $n \in \mathbb{N}$ then there is some nonstandard $a \in M$ such that $M \models \varphi[a, \vec{b}]$.

Proof: Suppose by contradiction that M thinks that φ holds exactly on the natural numbers. In particular it does so for zero, and for every such $a \in M$ for which φ holds, it also holds for $a+1$. Thus, by the principle of induction, so we apply induction to get that it holds for all elements of M, and in this case, looking at any $a \in M \setminus \mathbb{N}$ will lead to the desired contradiction. ■

{cor:nsrep}

Corollary 6.1.2. Let M be a nonstandard model of PA, and $A \subseteq \mathbb{N}$ a subset of $\mathbb{N}$ definable in PA, i.e. there is a formula φ such that $n \in A$ iff $\mathrm{PA} \vdash \varphi(S^n(0))$. Then, there is some $a \in M$ representing a sequence of zeros and ones (see section 3.1) such that

$$A = \{\, n \in \mathbb{N} \mid M \vDash (a)_n = 1 \,\}. \tag{26}$$

Proof: Consider the formula

$$\psi(x)\colon \exists_y \forall_{z<x} (\varphi(z) \leftrightarrow (y)_z = 1). \tag{27}$$

Then, ψ evidently holds for all natural numbers x, hence by Overspill it must hold for some nonstandard element b. We conclude by setting a to be the y obtained by knowing $M \vDash \psi[a]$, and noting that every natural number is less than a.[2] ■

(write a bit about ssys?)

6.2 Tennenbaum's Theorem

Tennenbaum's theorem states that every nonstandard model of PA is 'complicated'.

{thm:tennenba

Theorem 6.2.1 (Tennenbaum). Let M be a countable nonstandard model of PA, and assume that its domain is $\mathbb{N}$. Then, the operations S^M, $+^M$, $\times^M$, and $<^M$ are not all computable. (For a more precise result, see corollary 6.2.2 below.)

Proof: The first step of the proof is the following. By applying Corollary 6.1.2 together with the definability of Σ_1 truth, we obtain that there is some $a \in M$ such that, for any Σ_1 formula φ of Gödel number k, $(a)_{S^k(0)} = 1$ if φ holds in M, and 0 otherwise. Thus, if we are able to compute the successor function and indexation of sequences, we will be able to compute Σ^1 truth in M. This is a contradiction, as we explain in the next paragraph.

Let T be the set of Σ_1 sentences that hold in M. It should be noted that these contain PA^-, and moreover T is complete for Σ^1 sentences. If T is computable, we may apply proposition 4.3.1 to obtain a Rosser sentence for T, but it should be noted that P is equivalent (in PA^-) to a Σ_1 sentence. Thus, per the beginning of this paragraph, we conclude that T *must* make a decision on the truth of P, which is a contradiction.

Now all to remains is to investigate what it takes to evaluate sequence indexation. It is worthy of note that we have been very noncommittal about the particular schema we use to encode sequences. Recall that such a schema amounts to a formula in three free variables, say $\sigma(x, y, z)$, satisfying certain properties in PA. A very common encoding will have x encode a triple of numbers, say N, a_0, b_0, and compute z as the remainder of division of N by $a_0 y + b_0$. So, with this encoding, all that is required to index a sequence is to decode pairs, and to perform Euclidean division. Both of these may be performed by looping through all elements of M, and at each stage performing a certain check. For example, to decode a pair $p = \langle i, j \rangle$, loop through all pairs $x, y \in M$ and for each of them compute $(x +^M y) \times^M (x +^M y) +^M y$ and compare the output with p. If p does represent a pair, this is guaranteed to eventually halt, and once it does the pair (x, y) is the pair (i, j). Euclidean division is slightly more complex, in that it requires a comparison to check that the (guessed) remainder is as small as it should be.

This concludes the proof. Note that indeed we used all four operations in the language of arithmetic, so all we've shown is that they cannot all be computable at the same time. ■

[2] Proof: In any model of PA^-, by proposition 2.1.4, any element which is less than or equal to a standard element is itself standard. Thus, a must be bigger than all standard elements.

{cor:tennenba

Corollary 6.2.2. Let M be a countable nonstandard model of PA, and without loss of generality assume that its domain is $\mathbb{N}$. Then, neither $+^M$ nor $\times^M$ is computable.

Proof: The proof boils down to noticing that the bottleneck for the previous proof was the work it takes to index a sequence. Thus, if we are able to provide a sequence encoding that (for our purposes) requires only addition or multiplication to decode, we will have shown that neither of these operations can be computable.

First, a surprising result. In order to perform Euclidean division *by a standard element*, we need only know how to add. Indeed, suppose we are attempting to perform the division of m, a nonstandard element, by $S^n(0)$, a standard element. To do so, we iterate over all nonstandard numbers x and over all $0 \leq r < n$, and ask whether $m = (x +^M \cdots +^M x) +^M (1 +^M \cdots +^M 1)$, where x is added to itself n times, and 1 is added to itself r times. Since PA shows that Euclidean division works, and that any element less than n is a sum of less-than-n many ones, this algorithm will eventually halt.

As a particular instance of the above paragraph, given a *standard* prime number p, it is possible to check whether a *nonstandard* number m is divisible by p. Thus, if we encode a sequence of zeros and ones as a product of primes (the i-th primes divides a iff $(a)_i = 1$) we are able to index it over standard elements (because the standard n-th prime agrees with the n-th prime in M), and hence by considering an encoding of the Σ_1 sentences which hold in M we get a contradiction as in theorem 6.2.1. This shows that addition *cannot* be computable.

The case for multiplication consists of noticing that, say, the map $x \mapsto 2^x$ is an isomorphism between $(M, +)$ and a submonoid of $(M, \times)$. Thus, by replacing a by 2^a (which is definable in any model of PA) we are able to encode our sequence in a way that requires only multiplication to index over standard elements. ■

7 Miscellaneous Curiosities

7.1 Parikh's Theorem

References

[1] Richard Kaye. *Models of Peano arithmetic.* Oxford University Press, 1991.