Topics in recursion theory: primitive recursion, Kleene normal form, arithmetic definability, arithmetic hierarchy...

Mainly from: [1, 19], the Wikipedia, and Mathoverflow

April 16, 2014

Contents

1 What's this?

This is a collection of transcriptions and personal notes in the area of Recursion Theory. Our goal is the study some specific topics such as (i) the search of natural intermediate problems of the arithmetic hierarchy; that is, problems strictly in a class such as Σ_n^0, but not $\leq_m$-complete in that class. (ii) the need (or not) of non primitive recursive reductions in Σ_1^0, and others.

These rather disorganized notes are for personal use only and include some verbatim transcriptions of other works. Perhaps for the occasional reader, they can also be the source of inspiration and further research. My contribution is Chapter 6: the primitive recursive hierarchy.

2 Notation and preliminaries

Definition 1

> $f(x) \equiv g(x)$ *or* $f = g$*:* $\forall x : f(x) = g(x)$.
>
> ∞*-many: infinitely many.*
>
> $p : \mathbb{N}^2 \mapsto \mathbb{N}$, $l : \mathbb{N} \mapsto \mathbb{N}$, *and* $r : \mathbb{N}^2 \mapsto \mathbb{N}$ *denote the Cantor pairing function and its left and right inverses.*
>
> ϕ_e*: the partial recursive function that corresponds to the Turing machine with index* e*. Another notation:* $\phi_e(x) = \{e\}(x)$.
>
> $H(e, x, t)$*: if* e *is an index of Turing machine* M*, and* x *is its input, the predicate* $H(e, x, t)$ *means "*M *halted after* t *steps of computation".*
>
> $M(x){\downarrow}$*: the computation* $M(x)$ *halts (or "converges"); equivalent to* $\exists t : H(e, x, t)$ *where* e *is an index of* M.
>
> $M(x){\downarrow}^t$*: the computation* $M(x)$ *halts after* t *steps of computation; equivalent to* $H(e, x, t)$ *where* e *is an index of* M.
>
> $M(x){\uparrow}$*: the computation* $M(x)$ *does not halt (or "diverges"); equivalent to* $\forall t : \neg H(e, x, t)$ *where* e *is an index of* M.
>
> $M(x){\uparrow}^t$*: the computation* $M(x)$ *did not halt after* t *steps of computation; equivalent to* $\neg H(e, x, t)$ *where* e *is an index of* M.
>
> $A \leq_m B$*:* A *reduces "many-to-one" to* B.
>
> $A \leq_t B$*:* A *Turing-reduces to* B.
>
> $A \leq_m^{\text{PR}} B$*:* A *"PR-reduces" to* B.
>
> *If* A *and* B *are sets of integers,*
>
> > A *is r.e. in* B*:* $A = L(M^B)$ *for some oracle TM* M *(oracle* B*).*
> >
> > A *is recursive in* B*:* A *is r.e. in* B *with* M^B *total. Same as* $A \leq_t B$*. Same as: membership in* A *is decidable relative to an oracle for* B.

2.1 Summary of notation and some results

This material is essentially from Soare's book [19].

1. The ordinal ω is identified with $\mathbb{N}$.

2. φ is used instead of ϕ.
We write $\varphi_{e,t}(x) = y$ if $x, y, e < t$ and y is the output of $\varphi_e(x)$ in $< t$ steps of the Turing program P_e. If such a y exists we say $\varphi_{e,t}(x)$ converges, which we write as $\varphi_{e,t}(x)\downarrow$, and diverges ($\varphi_{e,t}(x)\uparrow$) otherwise. Similarly, we write $\varphi_e(x)\downarrow$ if $\varphi_{e,t}(x)\downarrow$ for some t and we write $\varphi_e(x)\downarrow = y$ if $\varphi_e(x)\downarrow$ and $\varphi_e(x) = y$, and similarly for $\varphi_{e,t}(x)\downarrow = y$.

3. We let $\widetilde{P_e}$ be the e^{th} oracle program under some effective coding, we let $\Phi_e^A(x)$ denote the partial function computed by $\widetilde{P_e}$ with oracle A, and we let $\phi_e^A(x)$ denote the maximum element used (i.e., examined) on the oracle tape A during the computation on input x.

4. "Computably enumerable" (c.e.) sets instead of "recursively enumerable" (r.e.).
W_e denotes the domain of φ_e, $W_e = \text{dom}(\varphi_e) = \{x : \varphi_e(x)\downarrow\}$.
$W_{e,s} = \text{dom}(\varphi)_{e,s} = \{x : \varphi_e(x)\downarrow$ in s steps$\}$, see item 2.

5. Similarly "computable" and "partial computable" instead of "recursive" and "partial recursive" respectively.

6. Some decision problems

   (a) $K \overset{\text{def}}{=} \{x : \varphi_x(x)$ converges$\} = \{x : x \in W_x\}$ is the SHP ("self-halting") set (or problem). The set K is not computable (recursive), but c.e.
   (b) $K_0 \overset{\text{def}}{=} \{\langle x, y\rangle : x \in W_y\}$. The set K_0 is not computable but c.e. The set K_0 corresponds to the halting problem HP: given x and y, does $\varphi_x(y)$ converge?
   (c) K_1: $\{x : W_x \neq \emptyset\}$, defined for at least one value.
   (d) Fin: FINITE, $\{x : W_x$ is finite$\}$.
   (e) Inf: INFINITE, $\{x : W_x$ is infinite$\}$.
   (f) Cof: CO-FINITE, $\{x : W_x$ is cofinite$\}$.
   (g) Tot: TOTAL, $\{x : \varphi_x$ is a total function$\} = \{x : W_x = \omega\}$.
   (h) Con: $\{x : W_x$ is total and constant$\}$.
   (i) Rec: RECURSIVE, $\{x : W_x$ is recursive (computable)$\}$.
   (j) Ext: $\{x : \varphi_x$ is extendible to a total computable function$\}$.

7. A is one-one reducible (1-reducible) to B ($A \leq_1 B$) if $A \leq_m B$ by a bijective function.

8. It may be proved that it makes no difference whether to use "$\leq_m$" or "$\leq_1$" in the definition of complete problems.

9. Jump operator, page 71. The diagonalization produces a noncomputable c.e. set K. Relativizing this procedure to A we get a set K^A c.e.a. in A such that $A \leq_T K_A$.
   Definition. (i) Let $K^A = \{x : \Phi_x^A(x)\downarrow\} = \{x : x \in W_x^A\}$. K_A is called the *jump of* A and is denoted by A'. (ii) $A(n)$, the nth jump of A, is obtained by iterating the jump n times; i.e., $A(0) = A$, $A(n{+}1) = (A(n))'$. It follows that $K_A \equiv K_0^A \equiv K_1^A$ where $K_0^A = \{\langle x, y\rangle : \Phi_x^A(y)\downarrow\}$ and $K_1^A = \{x : W_x^A \neq \emptyset\}$.

10. The jump is well-defined on degrees, Theorem 11.2.2. Let $0^{(n)} \overset{\text{def}}{=} \deg(\emptyset^{(n)})$. Thus $0 < 0' < 0'' \ldots$

$$\begin{aligned}
0 &= \deg(\emptyset) = \{B : B \text{ is computable}\} \\
0' &= \deg(\emptyset'),\ \text{where } \emptyset' \overset{\text{def}}{=} K^{\emptyset} \equiv K \equiv K_0 \equiv K_1 \\
0'' &= \deg(\emptyset'') = \deg(\text{Fin}) = \deg(\text{Tot}) = \deg(\text{Inf}) \\
0''' &= \deg(\emptyset''') = \deg(\text{Cof}) = \deg(\text{Cput}) = \deg(\text{Ext})
\end{aligned}$$

What is "Cput"?

11. Turing degrees and the jump operator.
    (a) $A \equiv_T B$ if $A \leq_T B$ and $B \leq_T A$ (equivalence relation).
    (b) Turing degree (or degree of unsolvability) of A is the equivalence class $\deg(A) \overset{\text{def}}{=} \{B : B \equiv_T A\}$.
    (c) Degrees $\mathbf{D}$: partially ordered set $(\mathbf{D}, \leq)$ under the relation $\deg(A) \leq \deg(B)$ iff $A \leq_T B$. We write $\deg(A) < \deg(B)$ if $A <_T B$, i.e., if $A \leq_T B$ and $B \not\leq_T A$.

There are (infinitely) many methods for indexing a family of computation "devices" – Turing machines, programs... Let M be a particular device; depending on the circumstances it may be appropriate to talk about "the index of M" or about "an index of M". The index of a PR function refers exclusively to PR functions, and not to a more general model of computation (such as Turing machine).

Definition 2 *A set $A \subseteq N$ is* not trivial *if $A \neq \emptyset$ and $A \neq \mathbb{N}$.*

Definition 3 *In this definition the functions f and s and the relation r are total. The function $f(\overline{x})$ with codomain $\{0,1\}$ represents the relation $\{\overline{x} : f(\overline{x}) = 1\}$.*
The relation *associated with the total function $f(\overline{x})$ is*

$$r(\overline{x}, y) = \begin{cases} 1 & \text{if } f(\overline{x}) = y \\ 0 & \text{otherwise} \end{cases}$$

The step function *associated with the total function $f(\overline{x})$ is*

$$s(\overline{x}, y) = \begin{cases} 1 & \text{if } f(\overline{x}) \geq y \\ 0 & \text{otherwise} \end{cases}$$

Consider the following sequence of integer operators, where all the operators are right associative.

$$\begin{array}{lcll} a \times n & = & a \times n = a + a + \cdots + a & (n\ a\text{'s}) \\ a \overset{1}{\uparrow} n & = & a \uparrow n \overset{\text{def}}{=} a^n = a \times a \times \cdots \times a & (n\ a\text{'s}) \\ a \overset{2}{\uparrow} n & = & a \uparrow a \uparrow \cdots \uparrow a & (n\ a\text{'s}) \end{array}$$

The following definition generalizes these examples.

Definition 4 *For m, $n \geq 0$, the function $a \overset{m}{\uparrow} n$ is*

$$\begin{cases} a \overset{-2}{\uparrow} n = a + 1 \\ a \overset{-1}{\uparrow} n = a + n \\ a \overset{m}{\uparrow} n = \underbrace{a \overset{m-1}{\uparrow} a \overset{m-1}{\uparrow} \cdots a \overset{m-1}{\uparrow} a}_{n\ a\text{'s}} & \textit{for } m \geq 1 \end{cases}$$

The case $m = 0$ follows from the case $m = -1$ and the induction rule (last line).

For $m \geq 1$ the operator $\overset{m}{\uparrow}$ is not associative. The Ackermann function [7] can be expressed ([12]) as $a(m, n) = 2 \overset{m-1}{\uparrow} (n + 3) - 3$.

Definition 5 (HP, SHP, and HAS-ZEROS)
HP*(halting problem): the set $\{(e, x) : \phi_e(x)\downarrow\}$; the question "given e, $x \in \mathbb{N}$, does $\phi_e(x)\downarrow$?".*
SHP*(self halting problem): the set $\{e : \phi_e(e)\downarrow\}$; the question "given $e \in \mathbb{N}$, does $\phi_e(e)\downarrow$?".*
HAS-ZEROS*: the set $\{e : \exists x\, \phi_e(x) = 0\}$; the question "given $e \in \mathbb{N}$, is there an integer x such that $\phi_e(x) = 0$?".*

3 Recursion Theorem

Theorem 1 ("Fixed-point" or "first recursion" theorem) *If g is a recursive function, there is an integer n such that $\phi_n = \phi_{g(n)}$.*

The integer n is called a fixed point of g for the indexing system in use.

Proof. For each integer i define the unary function H_i as follows:

```
$\boxed{H_i(x):}$
1) Compute $\phi_i(i)$; // $x$ is ignored here                  $(\star)$
2) If $\phi_i(i)\downarrow$, let $j$ be the output;
3)     Compute $\phi_j(x)$;                                     $(\star)$
4)     If $\phi_j(x)\downarrow$, output the result.
```

Note that the computation may diverge ("loop forever") at the lines marked "($\star$)". The previous family of programs, one for each $i \in \mathbb{N}$, effectively defines a family of indices, one for each H_i. Let the corresponding recursive function be $h(i)$. Thus $\phi_{h(i)}(x) \equiv H_i(x)$.

Consider the recursive function $g(h(x))$ where g is the given recursive function. Let its index be k, $\phi_k(x) \equiv g(h(x))$. As ϕ_k is total, $\phi_k(k)$ halts. Thus, when computing $H_k(x)$, we get

$$\begin{array}{lll} & H_k(x) & \\ = & \phi_{h(k)}(x) & [h(k) \text{ is index of } H_k] \\ = & \phi_{\phi_k(k)}(x) & [\phi_k(k)\downarrow] \\ = & \phi_{g(h(k))}(x) & [k \text{ is an index of } g\cdot h] \end{array}$$

Thus, for every x, $\phi_{h(k)}(x) = \phi_{g(h(k))}(x)$, so that $h(k)$ is a fixed point of g. Notice that $h(k)$ is effectively defined: it is the index of H_k, where k is the index of $g\cdot h$. □

From the Wikipedia – with some repetition...

Rogers' fixed-point theorem

Given a function F, a fixed point of F is, in this context, an index e such that $\phi_e \simeq \phi_{F(e)}$; in programming terms, e is semantically equivalent to $F(e)$.

Rogers' fixed-point theorem. If F is (total) computable, it has a fixed point.

This theorem is Theorem I in (Rogers, 1967: §11.2) where it is described as "a simpler version" of Kleene's (second) recursion theorem.

Proof of the fixed-point theorem

The proof uses a particular total computable function h, defined as follows. Given a natural number x, the function h outputs the index of the partial computable function that performs the following computation:

```
Given an input $y$, first attempt to compute $\phi_x(x)$.
If that computation returns an output $e$, then:
        Compute $\phi_e(y)$ and return its value, if any.
```

Thus, for all x, if $\phi_x(x)$ halts, then $\phi_{h(x)} = \phi_{\phi_x(x)}$, and if $\phi_x(x)$ does not halt then $\phi_{h(x)}$ does not halt; this is denoted $\phi_{h(x)} \simeq \phi_{\phi_x(x)}$. The function h can be constructed from the partial computable function $g(x,y) = \phi_{\phi_x(x)}(y)$ and the S-m-n theorem: for each x, $h(x)$ is the index of a program which computes the function $y \mapsto g(x,y)$.

To complete the proof, let F be any total computable function, and construct h as above. Let e be an index of the composition $F \circ h$, which is a total computable function. Then $\phi_{h(e)} \simeq \phi_{\phi_e(e)}$ by the definition of h. But, because e is an index of $F \circ h$, $\phi_e(e) = (F \circ h)(e)$, and thus $\phi_{\phi_e(e)} \simeq \phi_{F(h(e))}$. By the transitivity of $\simeq$, this means $\phi_{h(e)} \simeq \phi_{F(h(e))}$. Hence $\phi_n \simeq \phi_{F(n)}$ for $n = h(e)$.

Fixed-point free functions
A function F such that $\phi_e \not\simeq \phi_{F(e)}$ for all e is called *fixed point free*. The fixed-point theorem shows that no computable function is fixed point free, but there are many non-computable fixed-point free functions. "Arslanov's completeness criterion" states that the only recursively enumerable Turing degree that computes a fixed point free function is $0'$, the degree of the halting problem.

Kleene's second recursion theorem
An informal interpretation of the second recursion theorem is that self-referential programs are acceptable.

Second recursion theorem
For any partial recursive function $Q(x, y)$ there is an index p such that $\phi_p \simeq \lambda y.Q(p, y)$.

This can be used as follows. Suppose that we have a self-referential program, namely one that evaluates a computable function Q of two arguments where the first is supposed to be the index of that very program, and the second represents input. By the theorem, we have a program p that does exactly that. Note that p only has y as input; it does not have to be supplied with its own index but satisfies the "self referential" equation by construction.

The theorem can be proved from Rogers' theorem by letting $F(p)$ be a function such that $\phi_{F(p)}(y) = Q(p, y)$ (a construction described by the S-m-n theorem). One can then verify that a fixed-point of this F is an index p as required.

4 Arithmetic hierarchy

Definition 6 *An* instance *of a (possibly quantified) predicate P is obtained by assigning to the free variables of P a tuple of integers; thus, an instance has no free variables. The* question *or* statement *associated with an instance of P is "is the sentence P true?".*

Consider a "quantification sequence" $\exists x_1 \exists x_2 \ldots \exists x_n$. In the sequel we will use pairing functions and "compress" the quantifier sequence into a single quantifier $\exists x$, where $x = p(x_1, p(x_2, \ldots p(x_{n-1}, x_n) \ldots))$; here, p is a (recursive) bijection $p : \mathbb{N} \mapsto \mathbb{N}^2$. Similarly, we compress universal quantifiers.

Definition 7 *Let P be a recursive predicate; the variables of P are not shown. A problem belongs to the class Σ_n^0 if the corresponding question has the form*

$$\exists\forall\exists\forall \ldots Q : P$$

where n is the total number of quantifiers, Q is ∃ if and n is odd and ∀ otherwise, and P is a recursive predicate (no variables are shown). A problem belongs to the class Π^0_n if the corresponding question has the form

$$\forall\exists\forall\exists\ldots Q : P$$

where P is a recursive predicate, n is the total number of quantifiers, and Q is ∀ for n is odd and ∃ for n even. An equivalent definition for sets, *instead of* decision problems *is (see for instance [10])*

$$\begin{aligned}
\Delta^0_0 = \Sigma^0_0 = \Pi^0_0 &\overset{\text{def}}{=} \{\textit{recursive sets}\} && (1)\\
\Sigma^0_1 &\overset{\text{def}}{=} \{\textit{r.e. sets}\} && (2)\\
\Sigma^0_{n+1} &\overset{\text{def}}{=} \{L(M^B) : B \in \Sigma^0_n\} && (3)\\
&= \{A : A \textit{ is r.e. in some } B \in \Sigma^0_n\} && (4)\\
\Pi^0_n &\overset{\text{def}}{=} \{\textit{complements of sets in } \Sigma^0_n\} && (5)\\
\Delta^0_{n+1} &\overset{\text{def}}{=} \{L(M^B) : B \in \Sigma^0_n,\ M^B \textit{ total}\} && (6)\\
&= \{A : A \textit{ is recursive in some } B \in \Sigma^0_n\} && (7)\\
&= \{A : A \leq_t B \textit{ for in some } B \in \Sigma^0_n\} && (8)
\end{aligned}$$

((2) is included in (3)/(4)) Thus $\Sigma^0_0 = \Pi^0_0$ is the class of recursive problems; is is also denoted by Δ^0_0 or simply Δ. The problem P is in Σ^0_n iff the problem $\neg P$ is in Π^0_n.

The proof of the following result is easy.

Theorem 2 *For $n \geq 1$ the problem P is complete in Σ^0_n iff the problem $\neg P$ is complete in Π^0_n.*

4.1 Arithmetic hierarchy: classification of some sets

Definition 8 *A problem P is* exactly *(or strictly) the class Σ^0_n of the arithmetical hierarchy if it belongs to Σ^0_n but not to $\Sigma^0_{n-1} \cup \Pi^0_{n-1}$. Similar definition for a problem* exactly in the class Π^0_n.

We list some sets and classify them in the arithmetic hierarchy.

Theorem 3 *The set* EMPTY $\overset{\text{def}}{=} \{M : L(M) = \emptyset\}$ *is exactly in Π^0_1.*

Proof. (i) The corresponding statement is $\forall n \forall t : \neg H(e,x,t)$. (ii) Define a reduction $\neg\text{HP} \leq_m \text{EMPTY}$. Let e be an instance of $\neg$HP and let

$$f(x) = \begin{cases} 0 & \textit{if } \phi_e(e)\downarrow \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

We have that $e \in \neg\text{HP}$ iff $f \in \text{EMPTY}$.

The set TOTAL $\overset{\text{def}}{=} \{M : M \textit{ is total}\}$ *is exactly in Π^0_2. Proof (partial). The corresponding statement is $\forall n \exists t : H(e,x,t)$.*

The set FINITE $\overset{\text{def}}{=}$ $\{M : L(M)$ *is finite*$\}$ *is exactly in* Σ_2^0. <u>*Proof (partial).*</u> *The corresponding statement is* $\exists n \forall x \forall t : (x \geq n) \Rightarrow \neg H(e,x,t)$.

The set CO-FINITE $\overset{\text{def}}{=}$ $\{M : \overline{L(M)}$ *is finite*$\}$ *is exactly in* Σ_3^0. <u>*Proof (partial).*</u> *The corresponding statement is* $\exists n \forall x \exists t : (x \geq n) \Rightarrow H(e,x,t)$.

4.2 Arithmetic hierarchy: complete sets

We begin with a question.

Question.

$$\boxed{\text{Most well known sets in } \Sigma_1^0 \setminus \Delta \text{, such as HP, SHP, HAS-ZEROS..., are complete in } \Sigma_1^0 \text{. I don't know examples of such sets that are not complete. Is every set in } \Sigma_1^0 \setminus \Delta \text{ complete in } \Sigma_1^0 \text{? If not so: (i) proof it (is there some Ladner-type Theorem?) (ii) give examples – natural examples, if possible.}}$$

This question was answered by Friedberg and Muchnik (for Turing reductions) in 1956/7: *there are* non complete sets in $\Sigma_1^0 \setminus \Delta$ – but no "natural" is known, see page 15

Theorem 4 *The halting problem* (HP) *is* $\leq_m$*-complete in* Σ_1^0.

<u>Proof</u>. Let A be a set in Σ_1^0. By (3) (page **??**) $A = L(M)$ for some Turing machine M. Define the Turing machine M' with input n as follows: run $M(n)$; if $M(n)\downarrow$ in an accepting state (that is, if $n \in A$) then accept n; if $M(n)\downarrow$ in a rejecting state (that is, if $n \notin A$) then enter a non-halting configuration (and of course, do not halt if $M(n)\uparrow$). Thus $M'(n)\downarrow$ iff $n \in A$. □

Corollary 1 *The non-halting problem is* $\leq_m$*-complete in* Π_1^0.

Corollary 2 *The self-halting problem* (SHP) *is* $\leq_m$*-complete in* Σ_1^0.

<u>Proof</u>. It is obvious that SHP$\in \Sigma_1^0$.
Consider the instance $\langle M, n\rangle$ of HP; the corresponding question is "$M(n)\downarrow$?". Define the Turing machine M' as:

```
$M'(y)$, input $y$ (which is not used):
    run $\phi_n(n)$
```

Note that $M(n)\downarrow$ iff $M'(y)\downarrow$ for every y, and in particular, iff $M'(m')\downarrow$, where m' is an index of M'. Thus, we have reduced HP to SHP. □

Lemma 1 *The set* FINITE *is* Σ_2^0*-hard.*

<u>Proof</u>. We reduce an arbitrary set A of Σ_2^0 to FINITE

$$\{x : \exists y \forall z\, R(x,y,z)\} \leq_m \{M : L(M) \text{ is finite}\}$$

where R is recursive. We need a recursive function f such that for all x, $f(x)$ is an index of a Turing machine M_x such that $\exists y \forall z R(x,y,z) \Leftrightarrow L(M_x)$ is finite. For this purpose, given x, define M_x as

```
$\boxed{M_x(w):}$
  for each $y$ with $|y| \leq |w|$:
      search $z$ such that $\neg R(x,y,z)$
```

It follows that: (i) if $\forall y \exists z \, \neg R(x,y,z)$ then $L(M_x) = \Sigma^\star$; (ii) otherwise we have $\exists y \forall z \, R(x,y,z)$, so that the computation $M(w)$ does not halt for w sufficiently large; that is, $L(M_x)$ is finite. □

Theorem 5 *The* FINITE *problem is* $\leq_m$*-complete in* Σ^0_2.

Proof. Lemma 1 states that FINITE is Σ^0_2-hard. We only have to prove that it is in Σ^0_2; but this is simple because the corresponding statement is

$$\exists y \forall z \forall t : (z \geq y) \Rightarrow \neg H(e,z,t)$$

□

Theorem 6 *If A is* $\leq_m$*-complete in* Σ^0_n*, then* FINITE^A *is* $\leq_m$*-complete in* Σ^0_{n+2}.

Proof. (incomplete) (i) $\text{FINITE}^A \in \Sigma^0_{n+2}$. We have

$$[M \in \text{FINITE}^A] \Leftrightarrow [L(M^A) \text{ is finite}] \Leftrightarrow [\exists y \forall z \forall t (z \geq y) \Rightarrow M^A(z)\uparrow^t]$$

where without loss of generality we consider only Turing machines without reject states.

The predicate $M^A(z)\uparrow^t$ is in Δ^0_{n+1} because it is recursive in A, which is in Σ^0_n; thus it is also in Σ^0_{n+1}, and can be expressed in the form $\overbrace{\forall\exists\ldots\ldots\ldots}^{n+1 \text{ quantifiers}} P$, where P is recursive. Combining with a prefix of the form $\exists\forall$, we get

Define M^A, "jump"

$$\exists y \forall z \forall t : (z \geq y) \Rightarrow M^A(z)\uparrow^t$$

The overall sequence of quantifiers has the form

$$\exists\forall\forall \quad \overbrace{\forall\exists\ldots\ldots Q}^{n+1 \text{ alternations}}$$

where Q is an appropriate quantifier. Thus FINITE^A is in Σ^0_{n+2} (note that the 3 consecutive $\forall$'s count as 1 alternation).

(ii) FINITE^A is $\leq_m$-complete in Σ^0_{n+2}. We can proceed as in the proof of Lemma FINhard.
To complete. □

Corollary 3 $\text{FINITE}^{\text{HP}}$ *is* $\leq_m$*-complete in* Σ^0_3.

4.3 Some complete sets in Σ_3^0

The following problems are $\boxed{\leq_m\text{-complete}}$ in Σ_3^0.

- CO-FINITE $\overset{\text{def}}{=} \{M : L(M) \text{ is cofinite}\} = \{M : \neg L(M) \text{ is finite}\}$
- RECURSIVE $\overset{\text{def}}{=} \{M : L(M) \text{ is recursive}\}$
- REGULAR $\overset{\text{def}}{=} \{M : L(M) \text{ is regular}\}$
- CONTEXT-FREE $\overset{\text{def}}{=} \{M : L(M) \text{ is context-free}\}$

We now prove that the CO-FINITE language is $\leq_m$-complete in Σ_3^0.

Theorem 7 *The language* CO-FINITE *is $\leq_m$-complete in Σ_3^0.*

Proof. The proof has two parts.
(i) CO-FINITE$\in \Sigma_3^0$. Let e be an the index of M. The statement is: $M(x)$ accepts all except a finite number of x's, or

$$\exists y \forall x \exists t : (x \geq y) \Rightarrow H(e,x,t)$$

Thus CO-FINITE $\in \Sigma_3^0$.

(ii) We now prove that CO-FINITE is hard in Σ_3^0. From Lemma 6 (page 11) we have that FINITE$^{\text{HP}}$ is Σ_3^0-hard. Define a recursive function f such that

$$\begin{array}{rcl} M & \xrightarrow{f} & N \\ L(M^{\text{HP}}) \text{ is finite} & \Leftrightarrow & L(N) \text{ is cofinite} \end{array} \quad \begin{array}{r} (9) \\ (10) \end{array}$$

where N is the Turing machine $f(M)$. Given M, let K^{HP} be an oracle machine with oracle HP that works as follows.

$\boxed{K^{\text{HP}}(y), \text{ input } y}$

(1) Search z with $|z| > |y|$ that is accepted by M^{HP}. This is done by the well known "dovetailing technique". If such a z is found, go to step (2).

(2) Verify the YES oracle responses in step (1) by checking that the computation $M(w)$ halts, whenever the oracle answered YES to a query with input $\langle M, w\rangle$. (NO oracle responses are ignored). All these simulations terminate.

This describes a recursive transformation $M \mapsto K$ such that, if $L(M^{\text{HP}})$ is finite, then $L(K^{\text{HP}})$ is finite, and if $L(M^{\text{HP}})$ is infinite, then $L(K^{\text{HP}})$ is $\Sigma^\star$.

Let us now describe a recursive transformation $K \mapsto N$, by defining N as the Turing machine that accepts the strings that are not accepting computation histories of K^{HP} (on any input). Apart from the well known usual Turing machines conditions for non correct computation histories, we have now to deal with non valid oracle responses. The correctness proof of valid responses are made by the algorithm above (step (2)), so N has to detect incorrect responses, and these correspond to one or more halting computations when the oracle answer is NO. These can be detected "in parallel" (by the dovetailing technique).

We have thus defined the recursive function $M \xrightarrow{f} N$ as the composition of two recursive functions, $M \mapsto K \mapsto N$. Moreover the condition (10) is satisfied. □

4.4 Arithmetic hierarchy: on closures

Σ_n^0 is not closed under (infinite) intersection.

Let M_i be the i^{th} Turing machine program, and $H(i,k,t)$ be the proposition "at step t, the computation $M_i(k)$ is halted".

For each $m \in \mathbb{N}$, $U_m \stackrel{\text{def}}{=} \{e : (\exists k > m) : \phi_e(k)\downarrow\}$ belongs to Σ_1^0 because the predicate in the definition of the set can be written $\exists k, t : k > i \wedge H(i,k,t)$.

We have

$$\mathrm{Inf} \stackrel{\text{def}}{=} \{\mathrm{e} : \phi_{\mathrm{e}}(\mathrm{k})\downarrow \text{ for } \infty\text{-many } k\text{'s}\} = \bigcup_{\mathrm{m}} \mathrm{U_m}$$

Inf belongs to Π_2^0 once the predicate is $\forall m \exists k, t : H(e,k,t)$; in fact, Inf is Π_2^0-complete. Hence Inf is an intersection of Σ_1^0 sets which is not in Σ_1^0.

5 Non complete $\Sigma_1 \setminus \Delta_0^0$ sets

FROM "MATH**overflow**"

Is every non-recursive set in Σ_1 complete in Σ_1 (relatively to many-to-one reductions)?

Most well known sets in $\Sigma_1 \setminus \Delta_0$, such as the Halting problem, are complete in Σ_1, relatively to the many-to-one reduction. In fact I don't know any example of a (non recursive) set in Σ_1 that is not complete.

Is every set in $\Sigma_1 \setminus \Delta_0$ complete in Σ_1?

– If so, why the trouble of proving completeness for such problems?

– If not (is there some Ladner-type Theorem?), can anyone give natural examples of $\Sigma_1 \setminus \Delta_0$ sets that are not complete?

Of course the question generalizes for other levels of the arithmetic hierarchy.

I apologize if this question was already answered here, or if the answer is well known.

Asked by Armando Matos

Answer I
Not every set in $\Sigma_1 \setminus \Delta_1$ is Σ_1-complete.

For Turing reductions it was known as Post's Problem and resolved by Friedberg and Muchnik around 1957. Interestingly there is still no known natural solution to Post's problem. Perhaps the closest is as follows: let $W_{e,s}$ be the eth Σ_1 set in some standard enumeration, as it looks at stage s; let $K(j)[s]$ be the prefix free Kolmogorov complexity of j as approximated at stage s; and let $A = \bigcup_s A_s$ where the finite sets A_s are uniformly Δ_1 and

$$A = \{\langle e,n\rangle : \exists s (W_{e,s} \cap A_s = \emptyset \wedge \langle e,n\rangle \in W_{e,s} \wedge \Sigma_{\langle e,n\rangle \leq j \leq s} 2^{-K(j)[s]} < 2-(e+2))\}$$

Then A is K-trivial (which implies it is not Σ_1-complete) and not Δ_1 (Downey, Hirschfeldt, Nies, Stephan).

For m-reductions the problem was already resolved by Post. He constructed a set $A \in \Sigma_1 \setminus \Delta_1$ whose complement B contains no infinite Δ_1 set, and showed that such a set A would have to be m-incomplete. In standard terminology A is simple and B is immune.
Bjørn Kjos-Hanssen

There are even conjectures and results to the effect that there cannot be a natural solution: for example, there is no $e \in \omega$ and computable total f such that (1) for all sets X, $X <_T W_e^X <_T X'$, (2) if $X \equiv_T Y$ via Φ_d then $W_e^X \equiv_T W_e^Y$ via Φ_d. I believe even more is known, but I can't recall.
– Noah S 2 days ago 1

Does your remark on the lack of natural complete sets apply to many-one completeness (mentioned in the OP) as well as Turing completeness?
– Joel David Hamkins 2 days ago

Joel David Hamkins, I think it does, right? I suppose the halting probability Ω is a natural example of a Δ^0_2 set of incomplete tt-degree, though.
– Bjorn Kjos-Hanssen

Answer II
Let Ω be the halting probability, and let X be the set of rational numbers less than Ω . I think I can reasonably claim that X is a natural set.

Since Ω is left-c.e., X is Σ^0_1. Also, X is non-computable since it's Turing equivalent to Ω . However, X cannot be many-one complete for Σ^0_1 sets. Indeed, no convex set of rationals (or finite union of convex sets of rationals) can be many-one complete for Σ^0_1 sets.

Suppose X were complete. Note that $\emptyset'$ is uniformly many-one complete: from a Σ^0_1 index for a set, we can compute an index for its reduction to $\emptyset'$. Since X is complete, there is a reduction from $\emptyset'$ to X, and so by composing these we see that X is uniformly many-one complete (this argument works for any complete set).

Now, we'll make a Σ^0_1 set V, and by a standard argument with the recursion theorem we can assume we already know an n with $V = W_n$. Using the uniformity, this means we know the index of a computable reduction f from V to X.

Begin by computing $f(0)$, $f(1)$, and $f(2)$. These are rational numbers, and so are arranged in some order. Without loss of generality, let's assume that $f(0) \leq_Q f(1) \leq_Q f(2)$. Then enumerate 0 and 2 into V, but leave 1 out. By assumption, this means that $f(0)$ and $f(2)$ are in X, but $f(1)$ is not. But this contradicts the convexity of X.
Answered yesterday, Dan Turetsky

Nice although I suppose the m-degree of this set depends on the version of Ω used? Also instead of Ω we could use $0'$ which is even more natural?
– Bjørn Kjos-Hanssen yesterday

Yeah, I'm pretty sure I've got a construction of m-incomparable versions. If you mean the real $0.\emptyset'$', that should also work, but again different numberings will give different m-degrees. Also, I'm stumped on whether these are tt-complete.
– Dan Turetsky yesterday

Because Ω itself is not a canonical number, but is only a arbitrarily chosen member of an infinite class of Ω-numbers, sets defined using Ω will not be natural.
– Carl Mummert 3 hours ago

Well, even Hilbert' 10th problem has to be encoded to get a subset of ω... which can be done in many ways
– Bjørn Kjos-Hanssen 2 hours ago

Often, naturalness makes sense only modulo some notion of equivalence. In the context of m-equivalence, there is only one reasonable encoding of Hilbert's 10th problem, and it is natural. Not so for (left cuts of) Ω .
– Denis Hirschfeldt

From the Wikipedia

Albert Abramovich Muchnik (born 1934) is a Russian mathematician who worked in the field of foundations and mathematical logic.

He received his Ph.D from Moscow State Pedagogical Institute in 1959 under the advisorship of Pyotr Novikov.[1] Muchnik's most significant contribution was on the subject of relative computability. He and Richard Friedberg, independently introduced the priority method which gave an affirmative answer to Post's Problem regarding the existence of re Turing degrees between 0 and 0' . This groundbreaking result, now known as the Friedberg-Muchnik Theorem,[2][3] opened a wide study of the Turing degrees of the recursively enumerable sets which turned out to possess a very complicated and non-trivial structure. He also has a significant contribution in the subject of mass problems where he introduced the generalisation of Turing degrees, called "Muchnik degrees" in his work On Strong and Weak Reducibilities of Algorithmic Problems published in 1963.[4] Muchnik also elaborated Kolmogorov's proposal of viewing intuitionism as "calculus of problems" and proved that the lattice of Muchnik degrees is Brouwerian.

From Soare's book "Computability Theory and Applications" pages 176–179
The Friedberg-Muchnik Theorem that there exist c.e. sets A and B of incom-

parable degree is the canonical finite injury theorem, not only because it was the first example of a finite injury proof, but also because the injury pattern is the most typical, a bit more complex than the low simple set of Theorem 7.1.1 where the injury set I_e satisfies $|I_e| \leq e$, but not as complex as the unbounded injury pattern in Theorem 7.4.1 where we have merely $|I_e| < \infty$. Here we have $|I_e| < 2^e$ which is typical of these arguments.

Theorem 8 (Friedberg (1957), Muchnik (1956)) *There exist c.e. sets A and B of incomparable Turing degree.*

Proof. Proof. Without loss of generality we may assume that our effective numbering of oracle Turing programs satisfies (or can be effectively renumbered to satisfy) that every oracle Turing program $\hat{P}_e$ occurs with both an even and odd index $\hat{P}_e = \hat{P}_{2i} = \hat{P}_{2j+1}$ for some i and j. Hence, to satisfy all the incomparability requirements it suffices to satisfy merely all even requirements $\{R_{2e}\}_{e\in\omega}$ and likewise for odd requirements. (This enables us to have the subscript of R_e match that on Φ_e which makes the proof more perspicuous rather than having $R_{2e} : A \neq \Phi_e$.)

We construct c.e. sets A and B to meet the same requirements as in (6.1) of the Kleene-Post theorem,

$$R_e : A \neq \Phi_e^B \quad \text{for } e \text{ even} \tag{11}$$

$$R_e : B \neq \Phi_e^A \quad \text{for } e \text{ odd} \tag{12}$$

We say that requirement R_i has higher priority than R_j if $i < j$. We first present the *basic module* or *atomic strategy* for meeting a single requirement R_e and then explain how to combine these strategies.

Basic Module for Meeting R_e for e even. Select an integer x not yet in A. Wait for s such that $\Phi^{B_s^u}(x)\downarrow = 0$, which must happen if $\Phi_e^B = A$. (Now $u < s$ by our convention in Chapter 3.)

Action. At stage $s + 1$ enumerate x in A, and define a restraint function $r(e, s + 1) = s + 1$ which prevents lower priority requirements R_j, $j > e$, from enumerating any $z \leq r(e, s + 1)$ into B. If the higher priority requirements R_i, $i < e$, are finished acting by stage s then the restraint function $r(e, s+1)$ ensures that $B \upharpoonright s = B_s \upharpoonright_s s$ so that

$$\Phi B(x)\downarrow = 0 \neq 1 = A(x)$$

and requirement R_e is met. If some R_i, $i < e$, acts at some stage $t > s + 1$ then we reset the R_e basic module with a fresh witness x' and begin all over.

Construction. We now combine all the strategies as follows. Let $\omega^{[y]} = \{\langle x, y\rangle : x \in \omega\}$. To avoid conflict between requirements we choose witnesses $x \in \omega^{[e]}$ to meet R_e. Second we start with all restraint functions $r(e, 0) = -1$ so that $r(e, s) > 0$ indicates that R_e has been satisfied at some stage $t \leq s$ and no R_i, $i < e$, has acted since then. We only allow Re to act at stage $s + 1$ if its indicator $r(e, s) = -1$. Therefore, r acts both as an indicator function and restraint function.

Stage s = 0. Define $r(e, 0) = -1$ for all e.

Stage s+1 even. Choose the least even e such that

$$\begin{array}{rcl} r(e,s) = -1 & \wedge & (\exists x)[x \in \omega^{[e]} - A_s] \\ \wedge \quad \Phi^{B_s}_{e,x}(x)\downarrow = 0 & \wedge & (\forall i < e)[r(i,s) < x]] \end{array}$$

Action. If there is no such e, then do nothing and go to stage $s+2$. Otherwise, choose the least such e and the least corresponding x. We say R_e *acts* at stage $s+1$. Perform the following steps.

Step 1. Enumerate x in A.

Step 2. Define $r(e, s+1) = s+1$ (thereby restraining $B_s \upharpoonright s$).

Step 3. For all $j > e$, define $r(j, s+1) = -1$. We say that these lower priority requirements $\{R_j\}_{j>e}$ are injured at $s+1$ and are reset.

Step 4. For all $i < e$ define $r(i, s+1) = r(i, s)$. (This leaves the previous action performed by these higher priority requirements $\{R_i\}_{i<e}$ untouched).

Stage $s+1$ *odd.* Do the same for odd with the roles of A and B reversed.

□

Lemma 2 *If requirement R_e acts at some stage $s+1$ and is never later injured, then requirement R_e is met and $r(e,t) = s+1$ for all $t \geq s+1$.*

Proof. Suppose R_e acts at stage $s+1$ and say e is even. Then $\Phi^{B_s}(x)\downarrow = 0$ for some $x \in A_{s+1}$. Since no R_i, $i < e$, ever acts after stage $s+1$ it follows by induction on $t > s$ that R_e never acts again and $r(e,t) = s+1$ for all $t > s$. Hence,no R_j, $j > e$, enumerates any $x \leq s$ into B after stage $s+1$. Therefore, $\upharpoonright s = Bs \upharpoonright s$ and $\Phi^{B(x)}\downarrow = 0 \neq A(x)$. □

Lemma 3 *For every e requirement R_e is met, acts at most finitely often, and $r(e) = \lim_s r(e,s)$ exists.*

Proof. Fix e and assume true for all R_i, $i < e$. Let v be the greatest stage when some such R_i acts if ever and $v = 0$ if none exists. Then $r(e,v) = -1$ and this will persist until some stage $s+1 > v$ (if ever) when R_e acts. If R_e acts as some stage $s+1$ then by Lemma 2 that action satisfies R_e which never acts again and $r(e,t) = s+1$ for all $t \geq s+1$.

Either way $r(e)$ exists and R_e acts at most finitely often. Now suppose that R_e is not met. Then $\Phi^B_e = A$. Now by stage v at most finitely many elements $x \in \omega^{[e]}$ have been enumerated in A. Choose the least $x \in \omega^{[e]} - A_v$ such that $x >$v. Eventually there will be a stage s such that $\Phi^{B_s}_{e,s}(x)\downarrow = 0$ since $x \notin A_s$. Hence, R_e acts at stage $s+1$ and by Lemma 2 this action satisfies R_e forever. □

Proposition 1 *In Theorem 7.2.1 $|Ie| < 2^e$ where we define*

$$\text{(InjurySet)} \quad I_e = \{x : x \in A_{s+1} - As \wedge x \leq r(e,s)\} \tag{13}$$

for e even and similarly for e odd and B in place of A.

Proof. Define $f_s(i) = 1$ if $r(i, s) > 0$ and $f_s(i) = 0$ otherwise. If R_e is injured at stage $s+1$ it is only because some R_i acts, and hence $f_s(i) = 0$ and $f_s+1(i) = 1$ causing f to increase lexicographically at stage $s+1$. However, $fs \upharpoonright e$ is a binary string of exactly e bits and can increase lexicographically at most $2e - 1$ times. $\square$

5.1 On jumps

This material is from Li and Vitányi book ([11], exercise 3.6.18).
One can also consider more powerful notions of computability, called relativized computability, such as Turing machines equipped with oracles. Such an oracle is a subset A of the natural numbers, and a Turing machine T equipped with oracle A, denoted by T^A, can ask "is $n \in A$?"

Thus, if A is the set of programs (the binary code considered as a natural number) for which T (without oracle A) halts, then T^A can compute more than T . Let T_1, T_2,... be the standard enumeration of prefix machines, and let U be the reference prefix machine with $U(\langle i, p\rangle) = T_i(p)$ for all i, p.

In recursion theory one defines the *jump* A' of A as $A' = \{x : U^A(x) < \infty\}$. Main jumps are those of the empty set: $\emptyset$, $\emptyset'$, $\emptyset''$,... Clearly, $\emptyset'$ is recursively enumerable, by $U = U^\emptyset$, $\emptyset''$ is recursively enumerable by U' defined as $U' = U^{\emptyset'}$, $\emptyset'''$ is recursively enumerable by $U'' = U^{\emptyset''}$, and so on.

Define the halting probability of U' by $\Omega' = \Sigma_{U'(p)<\infty} 2^{-l(p)}$, and similarly the halting probability U'' by $\Omega'' = \Sigma_{U''(p)<\infty} 2^{-l(p)}$, and so on. Just as Ω is random with respect to the $\emptyset$ jump, every such halting probability is Martin-Löf random with respect to its respective jumps, that is, of the higher orders of randomness.

We are interested in natural examples of infinite binary sequences (equivalently, real numbers) that are of these higher orders of randomness, but are defined without recourse to oracles.

(a) Show that the probability that a program for the reference universal prefix machine U both outputs finitely many symbols and does not halt (has an infinite computation) is Martin-Löf random in the first jump of the halting problem.

(b) Define the probability $\beta = \Sigma 2 - l(p)$, where the summation is taken over the shortest $p \in \{0, 1\}^\star$ such that the set $Q = \{q : U(pq) < \infty\}$ is cofinite ($\{0, 1\}^\star - Q < \infty$). Show that β is as random as Ω'': it is Martin-Löf random in the second jump of the halting problem.

5.2 On Chaitin's Ω number

In the computer science subfield of algorithmic information theory, a Chaitin constant (Chaitin omega number) or halting probability is a real number that informally represents the probability that a randomly constructed program will halt. These numbers are formed from a construction due to Gregory Chaitin.

Although there are infinitely many halting probabilities, it is common to use the letter Ω to refer to them as if there were only one. Because Ω depends on the program encoding used, it is sometimes called Chaitin's construction instead of Chaitin's constant when not referring to any specific encoding.

Each halting probability is a normal and transcendental real number that is not computable, which means that there is no algorithm enumerating its digits.

5.2.1 Background

The definition of a halting probability relies on the existence of prefix-free universal computable functions. Such a function, intuitively, represents a programming language with the property that no valid program can be obtained as a proper extension of another valid program.

Suppose that F is a partial function that takes one argument, a finite binary string, and possibly returns a single binary string as output. The function F is called computable if there is a Turing machine that computes it.

The function F is called universal if the following property holds: for every computable function f of a single variable there is a string w such that for all x, $F(wx) = f(x)$; here "wx" represents the concatenation of the two strings w and x. This means that F can be used to simulate any computable function of one variable. Informally, w represents a "script" for the computable function f, and F represents an "interpreter" that parses the script as a prefix of its input and then executes it on the remainder of input. Note that for any fixed w the function $f(x) = F(wx)$ is computable; thus the universality property states that all computable functions of one variable can be obtained in this fashion.

The domain of F is the set of all inputs p on which it is defined. For F that are universal, such a p can generally be seen both as the concatenation of a program part and a data part, and as a single program for the function F.

The function F is called prefix-free if there are no two elements p, p' in its domain such that p' is a proper extension of p. This can be rephrased as: the domain of F is a prefix-free code (instantaneous code) on the set of finite binary strings. A simple way to enforce prefix-free-ness is to use machines whose means of input is a binary stream from which bits can be read one at a time. There is no end-of-stream marker; the end of input is determined by when the universal machine decides to stop reading more bits. Here, the difference between the two notions of program mentioned in the last paragraph becomes clear; one is easily recognized by some grammar, while the other requires arbitrary computation to recognize.

The domain of any universal computable function is a computably enumerable set but never a computable set. The domain is always Turing equivalent to the halting problem.

5.2.2 Definition

Let P_F be the domain of a prefix-free universal computable function F. The constant Ω_F is then defined as

$$\Omega_F = \sum_{p \in P_F} 2^{-|p|},$$

where $|p|$ denotes the length of a string p. This is an infinite sum which has one summand for every p in the domain of F. The requirement that the domain be prefix-free, together with Kraft's inequality, ensures that this sum converges to a real number between 0 and 1. If F is clear from context then Ω_F may be denoted simply Ω, although different prefix-free universal computable functions lead to different values of Ω.

5.2.3 Relationship to the halting problem

Knowing the first N bits of Ω, one could calculate the halting problem for all programs of a size up to N. Let the program p for which the halting problem is to be solved be N bits long. In dovetailing fashion, all programs of all lengths are run, until enough have halted to jointly contribute enough probability to match these first N bits. If the program p hasn't halted yet, then it never will, since its contribution to the halting probability would affect the first N bits. Thus, the halting problem would be solved for p.

Because many outstanding problems in number theory, such as Goldbach's conjecture are equivalent to solving the halting problem for special programs (which would basically search for counter-examples and halt if one is found), knowing enough bits of Chaitin's constant would also imply knowing the answer to these problems. But as the halting problem is not generally solvable, and therefore calculating any but the first few bits of Chaitin's constant is not possible, this just reduces hard problems to impossible ones, much like trying to build an oracle machine for the halting problem would be.

5.2.4 Interpretation as a probability

The Cantor space is the collection of all infinite sequences of 0s and 1s. A halting probability can be interpreted as the measure of a certain subset of Cantor space under the usual probability measure on Cantor space. It is from this interpretation that halting probabilities take their name.

The probability measure on Cantor space, sometimes called the fair-coin measure, is defined so that for any binary string x the set of sequences that begin with x has measure $2^{-|x|}$. This implies that for each natural number n, the set of sequences f in Cantor space such that $f(n) = 1$ has measure $1/2$, and the set of sequences whose nth element is 0 also has measure $1/2$.

Let F be a prefix-free universal computable function. The domain P of F consists of an infinite set of binary strings

$$P = \{p_1, p_2, \ldots\}$$

Each of these strings p_i determines a subset S_i of Cantor space; the set S_i contains all sequences in cantor space that begin with p_i. These sets are disjoint because P is a prefix-free set. The sum

$$\sum_{p \in P} 2^{-|p|}$$

represents the measure of the set

$$\bigcup_{i \in \mathbb{N}} S_i$$

In this way, Ω_F represents the probability that a randomly selected infinite sequence of 0s and 1s begins with a bit string (of some finite length) that is in the domain of F. It is for this reason that Ω_F is called a halting probability.

5.2.5 Properties

Each Chaitin constant Ω has the following properties:

- It is algorithmically random. This means that the shortest program to output the first n bits of Ω must be of size at least $n - O(1)$. This is because, as in the Goldbach example, those n bits enable us to find out exactly which programs halt among all those of length at most n.
- It is a normal number, which means that its digits are equidistributed as if they were generated by tossing a fair coin.
- It is not a computable number; there is no computable function that enumerates its binary expansion, as discussed below.
- The set of rational numbers q such that $q < \Omega$ is computably enumerable; a real number with such a property is called a left-c.e. real number in recursion theory.
- The set of rational numbers q such that $q > \Omega$ is not computably enumerable.
- Ω is an arithmetical number.
- It is Turing equivalent to the halting problem and thus at level Δ_2^0 of the arithmetical hierarchy.

Not every set that is Turing equivalent to the halting problem is a halting probability. A finer equivalence relation, Solovay equivalence, can be used to characterize the halting probabilities among the left-c.e. reals.

5.2.6 Uncomputability

A real number is called computable if there is an algorithm which, given n, returns the first n digits of the number. This is equivalent to the existence of a program that enumerates the digits of the real number.

No halting probability is computable. The proof of this fact relies on an algorithm which, given the first n digits of Ω, solves Turing's halting problem for programs of length up to n. Since the halting problem is undecidable, Ω can not be computed.

The algorithm proceeds as follows. Given the first n digits of Ω and a $k \leq n$, the algorithm enumerates the domain of F until enough elements of the domain have been found so that the probability they represent is within $2 - (k + 1)$ of Ω. After this point, no additional program of length k can be in the domain, because each of these would add $2-k$ to the measure, which is impossible. Thus the set of strings of length k in the domain is exactly the set of such strings already enumerated.

5.2.7 Incompleteness theorem for halting probabilities

For each specific consistent effectively represented axiomatic system for the natural numbers, such as Peano arithmetic, there exists a constant N such that no bit of Ω after the Nth can be proven to be 1 or 0 within that system. The constant N depends on how the formal system is effectively represented, and thus does not directly reflect the complexity of the axiomatic system. This incompleteness result is similar to Gödel's incompleteness theorem in that it shows that no consistent formal theory for arithmetic can be complete.

5.2.8 Super Omega

As mentioned above, the first n bits of Gregory Chaitin's constant Omega are random or incompressible in the sense that we cannot compute them by a halting algorithm with fewer than $n - O(1)$ bits. However, consider the short but never halting algorithm which systematically lists and runs all possible programs; whenever one of them halts its probability gets added to the output (initialized by zero). After finite time the first n bits of the output will never change any more (it does not matter that this time itself is not computable by a halting program). So there is a short non-halting algorithm whose output converges (after finite time) onto the first n bits of Ω. In other words, the enumerable first n bits of Ω are highly compressible in the sense that they are limit-computable by a very short algorithm; they are not random with respect to the set of enumerating algorithms. Jürgen Schmidhuber (2000) constructed a limit-computable "Super Omega" which in a sense is much more random than the original limit-computable Omega, as one cannot significantly compress the Super Omega by any enumerating non-halting algorithm.

6 A primitive recursive arithmetic hierarchy

In construction.
A different form of arithmetic hierarchy, namely the hierarchy relative to a oracle or set has been studied, see for instance the definition 4.1.2 of [19].

We follow another direction and define a *restricted* form of arithmetic hierarchy in which both the relations and the reductions must be primitive recursive, that is, defined in terms of PR relations. The interest of this definition is twofold. First, the collection of sets in every class of the hierarchy is recursively enumerable (r.e.). For instance, the class of PR-sets (Definition 9 below) is r.e. while the class of recursive sets is not. Second, most (decidable or undecidable) mathematical sets of interest that can be many-to-one reduced can also be reduced by a PR-reduction, that is, a reduction whose transformation function is PR. For instance, the problems HP (halting problem), SHP (self-halting problem) and HAS-ZEROS (at least one zero) *can be PR-reduced to each other.*

6.1 Basic concepts and results

We begin by studying the lowest class of the hierarchy, the class of PR-sets. How can we represent a set by a PR function? We will use the following convention

Definition 9 *A set T is a* PR-set *if there is a PR function $\chi_T : \mathbb{N} \mapsto \mathbb{N}$ such that*

$$\chi_T(x) = \begin{cases} 0 & \textit{if } x \notin T \\ 1 & \textit{if } x \in T \end{cases}$$

The function χ_T is called the characteristic function[1] *of T.*
A PR-relation *$R \subseteq \mathbb{N}^n$ is a relation that can be represented by a PR function $r : \mathbb{N}^n \mapsto \mathbb{N}$ as*

$$r(x_1, \ldots, x_n) = \begin{cases} 0 & \textit{if } (x_1, \ldots, x_n) \notin R \\ 1 & \textit{if } (x_1, \ldots, x_n) \in R \end{cases}$$

The function r is the characteristic function *of T.*

The characteristic function χ_A of a set A could have been defined as any function such that: $\chi_A(x) = 0$ if $x \notin A$ and $\chi_A(x) \geq 1$ if $x \in A$. The two definitions are equivalent.

It it easy to check that PR-sets and relations are closed for union, intersection, and complement.

Theorem 9 *If f is a given PR function, $\{p(x, f(x)) : x \in \mathbb{N}\}$ a PR-set.*

Proof. A characteristic function of the set is obtained by the following algorithm: compute $x = l(z)$ and $y = r(z)$. If $y = f(x)$, output 1, otherwise output 0. This function is PR. □

Definition 10 *A set A is PR-many-to-one-reducible (or simply PR-reducible) to a set B and we write $A \leq_m^{\text{PR}} B$, if there is a PR function f such that for every x we have $x \in A$ iff $\chi_B(f(x)) \in B$. The definition of a PR-reduction between two relations is similar.*

Theorem 10 *If $A \leq_m^{\text{PR}} B$ and B is a PR-set, then A is a PR-set.*

Proof. Let f be the PR function associated with the reduction. Then, $\chi_B \cdot f$ is a *characteristic function* of A. □

[1]which is of course the same as $\{(x, y) : y = f(x)\}$.

There are non-PR functions f with $\mathrm{cod}(()f)=2$

Some recursive functions, like the Ackermann function, are not PR because they "grow too fast".

However, the growth rate is not the only reason for non-PR-ness. In fact there are also recursive functions with codomain $\{0,1\}$ that are not PR.

Theorem 11 *There are total, non-PR functions, with codomain* $\{0,1\}$.

Proof. By diagonalization. Consider an effective enumeration of PR functions ϕ_0^{PR}, $\phi_1^{\mathrm{PR}}\ldots$. Define

$$f(n)=\begin{cases} 0 & \text{if } \phi_n^{\mathrm{PR}}(n)\geq 1 \\ 1 & \text{if } \phi_n^{\mathrm{PR}}(n)=0 \end{cases}$$

□

6.2 Comments and results on PR-reductions

Most of the (many to one) reductions that are usually defined in recursion theory, that is, "in practice", are PR-reductions. The corresponding functions often transform Turing machine instructions (or equivalently indices) in Turing machine instructions (or indices) and these transformations are often primitive recursive. Moreover PR reductions are used in many proofs of completeness.

Theorem 12 *There are sets A and B in $\Sigma_1^0\backslash\Delta_0^0$ such that $A\leq_m B$, but $A\not\leq_m^{\mathrm{PR}}B$.*

Proof. Let A be the HP set, that is, $\mathrm{HP}\overset{\mathrm{def}}{=}\{i:\phi_i(i)\downarrow\}$, and let $f(n)$ be an increasing function that grows faster than any PR function. Define $B=\{f(i): i\in\mathrm{HP}\}$. Clearly the transformation $n\to f(n)$ defines a many-to-one reduction between HP and B, and there is no PR reduction between HP and B. □

Definition 11 *A set A in a class $\mathcal{C}$ is* PR-complete in $\mathcal{C}$ *if, for every $B\in\mathcal{C}$, we have $B\leq_m^{\mathrm{PR}}A$.*

The following is a consequence of Theorem 12 above.

Theorem 13 *That there are no PR-complete sets in Σ_1^0.*

Definition 12 *The set A is* PR-similar *to the set B, and we write $A\sim^{\mathrm{PR}}B$, if $A\leq_m^{\mathrm{PR}}B$ and $B\leq_m^{\mathrm{PR}}A$. The relation $\sim^{\mathrm{PR}}$ is an equivalence relation. If A is not PR-similar to the set B, we write $A\not\sim^{\mathrm{PR}}B$. When there is no possibility of confusion we omit the superscript* PR.

6.3 PR-sets: the class Δ^{PR} of the PR hierarchy

The class of PR-sets is denoted by Δ^{PR}.

Theorem 14 *If A is a not trivial PR-set, then the class of sets $\{B : B \leq_m^{\text{PR}} A\}$ is the class of PR-sets. In other words, every not trivial PR-set is complete in the class of PR-sets.*

Proof. Let B be a PR-set and χ_B its (PR) characteristic function. Consider the integers $a \notin A$ and $a' \in A$. Use the following function for the PR-reduction

$$f(n) = \begin{cases} a & \text{if } \chi_B(n) = 0 \\ a' & \text{if } \chi_B(n) = 1 \end{cases}$$

□

6.4 The PR-hierarchy

Definition 13 *The PR-hierarchy is constituted by the classes of sets Δ^{PR}, Σ_i^{PR} ($i \geq 0$), and Π_i^{PR} ($i \geq 0$), which are similar to the corresponding arithmetic hierarchy classes (denoted by the superscript 0), see Definition 7 (page 8). The only differences are that P is a PR predicate or a PR-set (instead of a recursive predicate or a recursive set) and that the functions associated with the reductions are PR (instead of recursive).*

Theorem 15 *Every recursive relation (in the sense that its characteristic function is recursive) belongs to Σ_1^{PR}. Every recursive relation that is not PR strictly belongs to $\Sigma_1^{\text{PR}} \setminus \Delta^{\text{PR}}$.*

Proof. We exemplify for relations of arity 1 (PR-sets). Given a recursive relation $R(x)$, there is a PR binary relation implemented by a Turing machine M that does the following

$\boxed{M(x,t):}$

- During at most t steps, search for the integer x in the relation R. This is possible because the relation is recursive.
- If, at step t, the integer x was not found, output FALSE.
- If, at step t, the integer x is found, output TRUE.

The total (but possibly not PR) relation $r(x)$ can be expressed as

$$R = \{x : \exists t \, M(x,t)\}$$

As $M(x,t)$ is a PR-relation, r is in Σ_1^{PR}. □

Definition 14 (Strictly in a class of the hierarchy) *A set A is* strictly *in Σ_1^0 if it belongs to $\Sigma_1^0 \backslash \Delta$. A set A is* strictly *in Π_1^0 if it belongs to $\Pi_1^0 \backslash \Delta$. For $n \geq 2$: the set A is* strictly *in Σ_n^0 if it belongs to $\Sigma_n^0 \backslash (\Sigma_{n-1}^0 \cup \Pi_{n-1}^0)$; the set A is* strictly

in Π^0_n *if it belongs to* $\Pi^0_n \setminus (\Sigma^0_{n-1} \cup \Pi^0_{n-1})$.
We similarly define "strictly in a class of the PR-hierarchy".

Theorem 16 *There are problems A and B strictly in Σ_1^{PR} and $A \not\sim B$.*

<u>Proof</u>. Let A be a non PR-recursive set, whose existence is guaranteed by Theorem 11, page 24. From Theorem 15 (page 25), we see that A is strictly in Σ_1^{PR}. On the other hand it is easy to see that SHP is also strictly in Σ_1^{PR} (to see it, use Kleene normal form). But we can not have SHP$\leq_m^{\text{PR}} A$, because otherwise SHP would be recursive (every PR-reduction is also a many-to-one reduction). □

7 On Kleene's normal form Theorem

The Theorems 17 and 18 below expressing the Kleene normal form do not mention Turing machines or any other computation model – only things related to the logical classification and representation of functions, such as "partial recursive functions", "primitive recursive functions", the "minimization operator", are mentioned. However we will find convenienr to define partial recursive functions as those computable by a Turing machine.

We think that that this is not a limitation. Although (we think) "purely logical proofs" of those theorems are possible, every purely logical definition of a partial recursive function (say) always hides a "computation mechanism": in principle it must be possible to compute the function from its definition – the value of a function must be uniquely defined and computable given its definition. The computational character of the minimization operator "μ"is particularly clear: given x, to find $\mu_y f(x,y)$ we compute in succession $f(x,0)$, $f(x,1)\ldots$ until some of these computations halts and outputs 0 (*if some computation does not halt, the value is undefined*); the value of the expression $mu_y f(x,y)$ is the smallest y, if any, such that $f(x,y)=0$. When defining a partial recursive function, we may not specify arbitrary conditions on the definitions; for instance the definitions

– "... is the *second smallest* y such that $f(x,y)=0$.

– "... is the smallest y such that $f(x,y)$ diverges.

are certainly ilegal, in the sense of not being computable. Essentially there must be a "semantic" coincidence between "the function is logically undefined" (by some computational method) and "the corresponding computation does not halt".

———

The main burden in proving Kleene's normal form Theorem is to show that there are certain functions (related with "universal" or "general purpose" models of computation) that are primitive recursive. These proofs are neither difficult nor deep, but are often rather laborious. For instance, there is a *primitive recursive function* that accepts an integer y and checks things like

1. y represents a computation history of a certain Turing machine T – essentially a succession of machine configurations.
2. The state in the first configuration is the initial state.
3. The application of transition rules of T to each configuration produces the next configuration, except the first, is obtained from the previous one with the application

Kleene's normal form Theorem is a consequence of the following result, where the representation of partial recursive functions by Turing machines is assumed.

Lemma 4 *Let e be the index of a Turing machine and x its input. The function $C(e, x, t)$ that returns the configuration of the Turing machine after t steps of computation is primitive recursive.*

An equivalent result is: let e be the index of a Turing machine and c its configuration. The function $C(e, x)$ that returns the next configuration is primitive recursive. This corresponds essentially to the case $t = 1$ in the previous result.

Let us state the Kleene's normal form Theorem from [15], page 90.

Theorem 17 (KNF Theorem, Odifreddi) *There is a PR function $\mathcal{U}$ and (for each $n \geq 1$) PR predicates $\mathcal{T}_n$, such that for every recursive (total) function f of n variables there is a number e (called the* index *of f) for which the following hold:*

1. $\forall x_1 \ldots x_n \exists y : \mathcal{T}(e, x_1, \ldots, x_n, y)$

2. $\mathcal{U}(\mu_y \mathcal{T}_n(e, x_1, \ldots, x_n, y))$.

Theorem 17 is in fact a special case for it deals only with total functions. Also note that criterium 1. above, which characterizes the *total* character of the function, is not effective.

Most formulations of the Theorem are "general" in the sense that they give a representation or computation for any partial recursive function. A part of the statement which is often emphasised is that, to represent any partial recursive function, the operator μ needs to be used *at most once.* For instance, the theorem is stated in [1], page 94, is:

Theorem 18 (KNF Theorem, Boolos, Burgess and Jeffrey) *Every recursive total or partial function can be obtained from the basic functions (zero, successor, identity) by composition, primitive recursion, and minimization,* using this last process no more than once.

Sometimes, a register language, such as WHILE, is used as the general purpose computation model. In this case, Kleene's normal form Theorem can be stated as follows. For any partial recursive function $f(\overline{x})$ there is a program of the following form (which can be slightly simplified) that computes it.

```
% Input in registers x1,...xn, e
% Output in register x0
PR function (no WHILE's)
WHILE $y \neq 0$
        PR function (no WHILE's)
ENDWHILE
PR function (no WHILE's)
```

A more detailed information about a "normal form" WHILE program can be seen for instance in [3].

It is also interesting to mention Kleene's original version of the normal form Theorem, as stated in [9], page 288. The notation was slightly adapted.

Theorem 19 (KNF Theorem, Kleene) *For each $n \geq 0$: given any general recursive function $\phi(x_1, \ldots, x_n)$, a number e can be found such that*

$$\forall x_1 \ldots x_n \exists y : \mathcal{T}_n(e, x_1, \ldots, x_n, y), \tag{14}$$

$$\phi(x_1, \ldots, x_n) = \mathcal{U}(\mu_y \mathcal{T}_n(e, x_1, \ldots, x_n, y)), \tag{15}$$

$$\forall x_1 \ldots x_n y : \mathcal{T}_n(e, x_1, \ldots, x_n, y) \Rightarrow \mathcal{U}(y) = \phi(x_1, \ldots, x_n) \tag{16}$$

where $\mathcal{T}_n(e, x_1, \ldots, x_n, y)$ and $\mathcal{U}(y)$ are the particular primitive recursive predicate and function defined above.

Again, this result mentions only total (recursive) functions. Condition 14 says that the function represented by the index e is total. Condition 15 says that $\mathcal{U}(\mu_y \mathcal{T}_n(\ldots))$ is in fact the function ϕ. Finally, condition 16 says that, for any halting computational history y, the value $\mathcal{U}(y)$ is correct, that is, equal to $\phi(x_1, \ldots, x_n)$.

Regarding condition 16, we recall that the computational models or definitional systems may be non-deterministic, so that this is a kind of "normal form" theorem. Particular computations or definition sequences may of course diverge, even for total functions.

It seems that the equality of two partial recursive functions is not expressed by this result. Write as usual $f(\overline{x}) \equiv g(\overline{x})$ if, for any input $\overline{x}$ the following holds

Either $f(\overline{x})$ and $f(\overline{x})$ are both undefined
...or both are defined and have the same value

and, of course, $\mu_y(P(\overline{x}, y))$, where P is a total predicate, is undefined if there is no y such that $P(\overline{x}, y)$ holds. Then (part of) Theorem 19 could be rephrased as the following function equality

$$\phi(\overline{x}) \equiv \mathcal{U}(\mu_y \mathcal{T}_n(e, \overline{x}, y))$$

8 Primitive recursive functions: "positive" results

The closure under substitution and composition can be considered positive results. Another positive result is

Theorem 20 *Every nonempty recursively enumerable set can be enumerated by a primitive recursive function.*

Proof. (from [5]) Let $T(e, n, x)$ be Kleene's T-predicate, which is primitive recursive. Suppose that the e-th computably enumerable set

$$W_e = \{n \in \mathbb{N} : \exists x\, T(e, n, x)\}$$

is nonempty; say $n_0 \in W_e$. Let $l, r : \mathbb{N} \mapsto \mathbb{N}$ be the two primitive recursive projections associated with the Cantor pairing function. Define the primitive recursive function $f : \mathbb{N} \mapsto \mathbb{N}$ by

$$f(m) = \begin{cases} l(m) & \text{when } T(e, l(m), r(m)) \\ n_0 & \text{otherwise} \end{cases}$$

It is clear that f enumerates W_e. $\square$

Theorem 21 *If a PR function $t(x)$ bounds the execution time of a partial recursive (total) function $f(x)$ then $f(x)$ is PR.*

Proof. ??? $\square$

Theorem 22 *If $f(x)$ is PR, it is possible to use the definition of f to characterize a PR function $t(x)$ that is PR and bounds the execution time of $f(x)$.*

Proof. See the discussion of the execution time of a LOOP program in [13]. $\square$

9 Primitive recursive functions: "negative" results

9.1 A PR function can not simulate PR functions

Theorem 23 *There is no binary PR function ϕ_i such that $\phi_i(e, x)$ is $\phi_e(x)$. Equivalently there is no PR function with index i such that $\phi_i(y)$ decomposes y in e (the index of a PR function) and x (its input) and runs $\phi_e(x)$.*

Proof. By contradiction. If there is such PR function ϕ_i, there exists also be a PR function with some index i' that works as follows

```
$\boxed{\phi_{i'}(x)}$
    run $\phi_x(x)$;
    output 1+the result of $\phi_x(x)$
```

But then $\phi_{i'}(i') = \phi_{i'}(i') + 1$ $\square$

Theorem 23 also holds for any (indexable) class of functions that is included in the total (computable) class. Thus,

Corollary 4 *There is no universal PR function.*

This makes it difficult to use diagonalization and prove (directly) that some PR existential problem (such as, "given the PR function f, is there some x such that $f(x) = 0$?") is undecidable and derive from that the undecidability of the halting problem.

When we say "there is no universal PR function" this means that PR functions do not include one of the following capabilities. 1) giving an integer x (loop program Gödel number) translate it to a convenient form p (integer code of a convenient program) and store it in a register. 2) Run the program(integer coded) in p with input p.

Only part 2) is impossible within the class of PR functions. Why? One reason may be that the program p may of course mention the register p and say "run myself", which could result in an infinite computation (thus not in PR).

Theorem 24 *There are (total) recursive functions with codomain* $\{0, 1\}$ *that are not primitive recursive.*

Proof. Consider the following function, where ϕ_e is the LOOP program with index e.

$$f(e) = \begin{cases} 0 & \text{if } \phi_e(e) \geq 1 \\ 1 & \text{if } \phi_e(e) = 0 \end{cases}$$

The function f is total, its codomain is $\{0, 1\}$. We show that is not primitive recursive. Define $g(e) = 1 - f(e)$. Note that for all e $g(e) \neq f(e)$. Let e' be an index of g. We have

$$g(e') = 1 - f(e') \tag{17}$$

But by definition of f, $f(e') = 0$ if $g(e') = 0$, and $f(e') = 1$ if $g(e') \geq 1$. From (17)

$$\begin{cases} g(e') = 1 & \text{if} \quad f(e') = 0 & \text{if} \quad g(e') = 0 \\ g(e') = 0 & \text{if} \quad f(e') = 1 & \text{if} \quad g(e') = 1 \end{cases}$$

This is a contradiction. □

Corollary 5 *The relation*

$$r(i, e, y) = \begin{cases} 1 & \textit{if } \phi_i(e) = y \\ 0 & \textit{otherwise} \end{cases}$$

is not PR.

Proof. Otherwise we could define $f(e)$, where f is the function used in the proof of Theorem 24, as

$$f(e) = \begin{cases} 0 & \text{if } r(e, e, 1) \text{ holds} \\ 1 & \text{if } r(e, e, 0) \text{ holds} \end{cases}$$

and f would be PR. □

9.2 The inverse of a PR function need not be PR

The inverse of a total function $f(x)$ can only be total (and thus can only be PR) if the codomain is $\mathbb{N}$. However it is not in general possible to know if a given PR function is onto? (this problem is unsoluble).

Review this section!

Theorem 25 *There are unary PR functions with codomain in* $\{0,1\}$ *whose inverse is not PR.*

Proof. Let $T(e,x,t)$ be the Kleene T-predicate and define $T'(p(e,t)) \overset{\text{def}}{=} T(e,e,t)$, where $p(e,t)$ is a Cantor pairing function. The inverse of $T'(x)$ is not PR. $T(e,x,t)$ is PR. Let $T'(p(e,t)) \overset{\text{def}}{=} T(e,e,t)$, where $p(e,t)$ is a Cantor pairing function with inverse functions l and r. □

10 Unary primitive recursive functions

Let us consider the problem HAS-ZEROS, Definition 5 (page 6). The instance of the problem is the pair $\langle f, \overline{x}\rangle$, where f is a PR function and $\overline{x}$ is the tuple of input values. This problem is related to the halting problem HP and also to Kleene's normal form. However, in HP the function f may be considered to be fixed – corresponding to the universal Turing machine – and has an extra argument, an index or Gödel number.

Thus, it seems that there are several variants of the HAS-ZEROS problem. However, if we use the $\mathrm{S_{mn}}$ Theorem it is easy to see that

– To specify "a primitive recursive function $f(\overline{x},y)$, and the value of $\overline{x}$" is equivalent to specify "a primitive recursive function of the form $f_{\overline{x}}(y)$" which is less general than (or as general as) to specify "a primitive recursive function f".

Thus, for simplicity, the decision problems can and will often be rephrased in the unary form. This simpler form is used in most of the problems studied in this work. In other words, most of the time we will be dealing with problems whose instance is a *general unary PR function.*

We now prove that, in terms of Recursion Theory, the following 3 problems are equivalent:

(1) Problem HAS-ZEROS for arbitrarily arity: given the unary PR function $f(\overline{x})$, is there some tuple $\overline{x}$ such that $f(\overline{x})=0$?

(2) Unary problem HAS-ZEROS: given the unary PR function $f(x)$, is there some x such that $f(x)=0$?

(3) T-HAS-ZEROS.
Instance: Index e and value of $\overline{x}$
Question: Is there some y such that $T(e,\overline{x},y)=0$?
(T is Kleene's predicate)

For simplicity we denote the problems by (1), (2), and (3). The following reductions prove the equivalence of the three problems for each arity n, where $\overline{x}=\langle x_1,\ldots,x_n\rangle$.

(1)$\leq_m$(2): Let $f(\overline{x}, y)$ be the instance of (1). Using the $\mathrm{S_{mn}}$ Theorem, define $f_{\overline{x}}(y) \equiv f(\overline{x}, y)$. The transformation (1) $\rightarrow$ (2) is $\langle f, \overline{x}\rangle \rightarrow f_{\overline{x}}$. Clearly, given f and $\overline{x}$, there is an y such that $f(\overline{x}, y) = 0$ iff there is some y such that $f_{\overline{x}}(y) = 0$.

(2)$\leq_m$(3): Given the function f, consider the function $g(x_1, \ldots, x_n, y) = f(y)$ and let e be its index. The transformation is

$$f(y) \quad \rightarrow \quad T(e, \langle 0, \ldots, 0\rangle, y)$$

(the 0's can be replaced by any other constant). The TM with index e starts by writing $\langle x_1, \ldots, x_n\rangle$ on the tape and then computes $g(x_1, \ldots, x_n, y) = f(y)$.

(3)$\leq_m$(1): it is enough to use the $\mathrm{S_{mn}}$ Theorem to "include" e and $\overline{x}$ in the function T, that is, define $f(y) = T_{e,\overline{x}}(y)$.

11 From [Boolos et al]: mainly logic and arithmetic

Languages, [1], pages 100–104.

- Language $L^\star$ of arithmetic: first-order theory (quantifiers, logical connectives...) with non-logical symbols: constant 0, binary predicate $<$, unary function successor, binary functions $+$ and $\times$.
- $\uparrow$-arithmetics (or exp-arithmetics): arithmetics with the extra binary functions $\uparrow$. Easier to prove Gödel's incompleteness Theorem with this enriched language, see for instance [18].
- Rudimentary formulas: formula built up from atomic formulas using only negation, conjunction, disjunction, and bounded quantifications $\forall x < t$ and $\exists x < t$, where t may be any term of the language (not involving x).
- $\exists$-rudimentary formula: formula of form $\exists x : F$ where F is rudimentary. Similarly for $\forall$-rudimentary formula
- Standard interpretation $\mathbb{N}^\star$: $|\mathbb{N}^\star|$ is $\mathbb{N}$, the set of natural (non-negative) integers, $0^{\mathbb{N}^\star}$ is 0 (zero)... (that is, $<^{\mathbb{N}^\star}$, $'^{\mathbb{N}^\star}$, $+^{\mathbb{N}^\star}$, $\cdot^{\mathbb{N}^\star}$.
- Correct: true in the standard interpretation.

Theories, [1], pages 214....

- ([1], pages 207…) The theory Q, *minimal arithmetic*:

$$0 \neq x' \tag{18}$$
$$x' = y' \implies x = y \tag{19}$$
$$x + 0 = x \tag{20}$$
$$x + y' = (x + y)' \tag{21}$$
$$x \cdot 0 = 0 \tag{22}$$
$$x \cdot y' = (x \cdot y) + x \tag{23}$$
$$\neg(x < 0) \tag{24}$$
$$x < y' \Leftrightarrow (x < y \vee x = y) \tag{25}$$
$$0 < y \Leftrightarrow y \neq 0 \tag{26}$$
$$x' < y \Leftrightarrow (x < y \wedge y \neq x') \tag{27}$$

  There is a natural nonstandard model for Q, called the *system of ordinal numbers*.
  Q is not strong enough to prove major theorems of number theory, but the following result is important:

  > An $\exists$-rudimentary sentence is correct if and only if it is a theorem of Q.

- ([1], pages 216…) The theory R, *Robinson arithmetic.* Add to R the axiom schema

$$(x = 0) \vee (\exists y : x = y') \tag{28}$$

  and replace (24)–(27) by

$$x < y \Leftrightarrow \exists z : z' + x = y \tag{29}$$

  There is also a natural nonstandard model for R, called the *system of cardinal numbers*, in which (27) fails.

- ([1], pages 214…) The theory P of Peano arithmetic is the set of all sentences of the language of arithmetic that are provable from (or equivalently, are consequences of) the axioms of Q plus the the schema of induction.

---

[1], Pages 199–204.

- Arithmetic formula $F(x)$ defines the set $S \subseteq \mathbb{N}$: $\forall a \in \mathbb{N} : F(\overline{a}) \Leftrightarrow a \in S$.
  Depending on the arithmetic we also say: "S is arithmetical" or "S is $\uparrow$-arithmetical (or "exp-arithmetical.").

- The arithmetic formula $F(x, y)$ defines the relation $R \subseteq \mathbb{N} \times \mathbb{N}$: $\forall a, b \in \mathbb{N} : F(\overline{a}, \overline{b}) \Leftrightarrow \langle a, b\rangle \in R$.
  We say that the relation R is arithmetic.

- The arithmetic formula $F(x, y)$ defines the function $f(x) : \mathbb{N} \to \mathbb{N}$ if it defines the relation $\{\langle x, f(x)\rangle\}$

– Examples if arithmetical functions; $+(x, y)$, $\dot{-}\ (x, y)$, quo(x, y), rem(x, y), IF-EQ-ZERO(x, y, z)...

– Composition, primitive recursion and minimization applied to exp-arithmetical functions result in exp-arithmetical functions.

– Lemma 16.4: every partial recursive function is exp-arithmetical.

– Every partial recursive function is arithmetical (Lemma 16.6).

– Every semi-decidable set is arithmetical (Lemma 16.6).

– Every partial recursive function is arithmetically definable by a generalized rudimentary arithmetic formula (Lemma 16.6).

– Every generalized $\exists$-rudimentary arithmetic formula is arithmetically equivalent to an $\exists$-rudimentary formula (Lemma 16.6).

– Every partial recursive function is arithmetically definable by an $\exists$-rudimentary formula (Lemma 16.6).

– Every partial recursive function is obtainable by composition of from rudimentary functions (Lemma 16.6).

11.1 Relation to Turing machines

The Turing computable sets of natural numbers are exactly the sets at level Δ^0_1 of the arithmetical hierarchy.

The recursively enumerable sets are exactly the sets at level Σ^0_1.

No oracle machine is capable of solving its own halting problem (a variation of Turing's proof applies). The halting problem for a $\Delta^{0,Y}_n$ oracle in fact sits in $\Sigma^{0,Y}_{n+1}$.

Post's theorem establishes a close connection between the arithmetical hierarchy of sets of natural numbers and the Turing degrees. In particular, it establishes the following facts for all $n \geq 1$:

- The set $\emptyset^{(n)}$ (the nth Turing jump of the empty set) is many-one complete in Σ^0_n.
- The set $\mathbb{N} \setminus \emptyset^{(n)}$ is many-one complete in Π^0_n.
- The set $\emptyset^{(n-1)}$ is Turing complete in Δ^0_n.

The polynomial hierarchy is a "feasible resource-bounded" version of the arithmetical hierarchy in which polynomial length bounds are placed on the numbers involved (or, equivalently, polynomial time bounds are placed on the Turing machines involved). It gives a finer classification of some sets of natural numbers that are at level Δ^0_1 of the arithmetical.

11.2 Logic and decidability, Chapter 11

([1], page 132. . .)

If the decision problem for logical implication were solvable, the halting problem would be solvable, which (assuming Turing's thesis) we know it is not. Hence we have established the following result, assuming Turing's thesis.

Theorem 26 (Church's theorem) *The decision problem for logical implication is unsolvable.*

Thus we have reduced the problem of determining whether for some n we have $f(m, n) = 0$ to the problem of determining whether Γ implies $D(\overline{m})$. That is, we have established that if the decision problem for logical implication were solvable, the nullity problem for f would be solvable, which it is known, as we have said, that it is not, assuming Church's thesis. Hence we have established the following result, assuming Church's thesis.

Theorem 27 (Church's theorem) . *The decision problem for logical implication is unsolvable.*

12 The n-Category Café – Weak Systems of Arithmetic

From [golem.ph.utexas.edu/category/2011/10/weak_systems_of_arithmetic.html]. Authors: John Baez, Jeffrey Ketland, et al.

Posted by John Baez
[http://golem.ph.utexas.edu/ distler/blog/mathml.html]

The recent discussion about the consistency of arithmetic made me want to brush up on my logic. I'd like to learn a bit about axioms for arithmetic that are weaker than Peano arithmetic [http://en.wikipedia.org/wiki/Peano_axioms]. The most famous is Robinson arithmetic:

- Robinson arithmetic [http://en.wikipedia.org/wiki/Robinson_arithmetic], Wikipedia.

Robinson arithmetic is also known as Q, after a Star Trek character who could instantly judge whether any statement was provable in this system, or not:

Instead of Peano arithmetic's axiom schema for mathematical induction, Q only has inductive definitions of addition and multiplication, together with an axiom saying that every number other than zero is a successor. It's so weak that it has computable nonstandard models! But, as the above article notes:

Q fascinates because it is a finitely axiomatized first-order theory that is considerably weaker than Peano arithmetic (PA), and whose axioms contain only one existential quantifier, yet like PA is incomplete and incompletable in the sense of Gödel's Incompleteness Theorems, and essentially undecidable.

But there are many interesting systems of arithmetic between PA and Q in strength. I'm hoping that if I tell you a bit about these, experts will step in and tell us more interesting things—hopefully things we can understand!

---

addrhttp://golem.ph.utexas.edu/~distler/blog/mathml.html

For example, there's primitive recursive arithmetic, or PRA:

- Primitive recursive arithmetic [http://en.wikipedia.org/wiki/Primitive_recursive_arithmetic], Wikipedia.

This system lacks quantifiers, and has a separate predicate for each primitive recursive function, together with an axiom recursively defining it.

What's an interesting result about PRA? Here's the only one I've seen: its proof-theoretic ordinal [http://en.wikipedia.org/wiki/Proof-theoretic_ordinal] is ω^ω. This is much smaller than the proof-theoretic ordinal for Peano arithmetic, namely ε_0.

What's ε_0? It's a big but still countable ordinal which I explained back in week236 [http://math.ucr.edu/home/baez/week236.html]. And what's the proof-theoretic ordinal of a theory?

> Ordinal analysis concerns true, effective (recursive) theories that can interpret a sufficient portion of arithmetic to make statements about ordinal notations. The proof theoretic ordinal of such a theory T is the smallest recursive ordinal that the theory cannot prove is well founded – the supremum of all ordinals α for which there exists a notation o in Kleene's sense such that **T** proves that σ is an ordinal notation.

For more details, try this wonderfully well-written article:

- Michael Rathjen, The art of ordinal analysis [http://www.icm2006.org/proceedings/Vol_II/contents/ICM_Vol_2_03.pdf], International Congress of Mathematicians, II, Eur. Math. Soc., Zurich, pp. 45–69.

Climbing down the ladder we eventually meet elementary function arithmetic, or EFA:

- Elementary function arithmetic [http://en.wikipedia.org/wiki/Elementary_function_arithmetic], Wikipedia.

Its proof-theoretic ordinal is just ω^3. It's famous because Harvey Friedman made a grand conjecture about it:

> Every theorem published in the Annals of Mathematics whose statement involves only finitary mathematical objects (i.e., what logicians call an

arithmetical statement) can be proved in EFA. EFA is the weak fragment of Peano Arithmetic based on the usual quantifier-free axioms for 0, 1, $+$, $\times$, exponential, together with the scheme of induction for all formulas in the language all of whose quantifiers are bounded.

Does anyone know yet if Fermat's Last Theorem can be proved in EFA? I seem to remember early discussions where people were wondering if Wiles' proof could be formalized in Peano arithmetic.

But let's climb further down the ladder. How low can we go? I guess ω is too low to be the proof-theoretic ordinal of any theory "that can interpret a sufficient portion of arithmetic to make statements about ordinal notations." Is that right? How about $\omega+1$, 2ω, and so on?

There are some theories of arithmetic whose proof-theoretic ordinal is just ω^2. One of them is called $I\Delta_0$. This is Peano arithmetic with induction restricted to predicates where all the for-alls and there-exists quantify over variables whose range is explicitly bounded, like this:

$$\forall i \leq n \forall j \leq n \forall k \leq n : i^3 + j^3 \neq k^3$$

Every predicate of this sort can be checked in an explicitly bounded amount of time, so these are the most innocuous ones.

Such predicates lie at the very bottom of the arithmetical hierarchy [http://planetmath.org/encyclopedia/ArithmeticalHierarchy.html], which is a way of classifying predicates by the complexity of their quantifiers. We can also limit induction to predicates at higher levels of the arithmetic hierarchy, and get flavors of arithmetic with higher proof-theoretic ordinals.

But you can always make infinities bigger – to me, that gets a bit dull after a while. I'm more interested in life near the bottom. After all, that's where I live: I can barely multiply 5-digit numbers without making a mistake.

There are even systems of arithmetic too weak to make statements about ordinal notations. I guess Q is one of these. As far as I can tell, it doesn't even make sense to assign proof-theoretic ordinals to these wimpy systems. Is there some other well-known way to rank them?

Much weaker than Q, for example, is Presburger arithmetic:

- Presburger arithmetic [http://en.wikipedia.org/wiki/Presburger_arithmetic], Wikipedia.

This is roughly Peano arithmetic without multiplication! It's so simple you can read all the axioms without falling asleep:

$$\neg(0 = x + 1) x + 1 = y + 1 \Rightarrow x = y x + 0 = x (x + y) + 1 = x + (y + 1)$$

and an axiom schema for induction saying:

$$(P(0) \wedge (P(x) \Rightarrow P(x+1))) \;\Rightarrow\; P(y)$$

or all predicates P that you can write in the language of Presburger arithmetic.

Presburger arithmetic is so simple, Gödel's first incompleteness theorem doesn't apply to it. It's consistent. It's complete: for every statement in Presburger arithmetic, either it or its negation is provable. But it's also decidable: there's an algorithm that decides which of these two alternatives holds!

However, Fischer and Rabin [www.lcs.mit.edu/publications/pubs/ps/MIT-LCS-TM-043.ps] showed that no algorithm can do this for all statements of length n in less than $2^{2^{cn}}$ steps. So, Presburger arithmetic is still fairly complicated from a practical perspective. (In 1978, Derek Oppen showed an algorithm with triply exponential run time can do the job.)

Presburger arithmetic can't prove itself consistent: it's not smart enough to even say that it's consistent! However, there are [http://mathoverflow.net/questions/9864/presburger-arithmetic/10027#10027] weak systems of arithmetic that can prove themselves consistent. I'd like to learn more about those. How interesting can they get before the hand of Gödel comes down and smashes them out of existence?

Re: Weak Systems of Arithmetic

Does anyone know yet if Fermat's Last Theorem can be proved in EFA? I seem to remember early discussions where people were wondering if Wiles' proof could be formalized in Peano arithmetic.

I've heard Angus MacIntyre talking about this. He is working on a paper arguing that Wiles' proof translates into PA. I say "arguing" rather than "proving" because all he plans to do is show that the central objects and steps can be formalised in PA, rather than translate the entirety of Wiles' proof, which would be a a ridiculously Herculean task. I don't know if his paper is available yet, but there's some discussion of it here [http://rjlipton.wordpress.com/2011/02/03/infinite-objects-and-deep-proofs/].

Posted by: Richard Elwes on October 11, 2011 8:52 AM

[http://golem.ph.utexas.edu/~distler/blog/mathml.html] Richard wrote:

> I've heard Angus MacIntyre talking about this. He is working on a paper arguing that Wiles' proof translates into PA.

Hmm, that's interesting! Sounds like a lot of work—but work that's interesting if you really know and like number theory and logic. Of course one would really want to do this for Modularity Theorem [http://en.wikipedia.org/wiki/Modularity_theorem], not just that piddling spinoff called Fermat's Last Theorem.

> I say "arguing" rather than "proving" because all he plans to do is show that the central objects and steps can be formalised in PA, rather than translate the entirety of Wiles' proof, which would be a a ridiculously Herculean task.

Right. But by the way, I think most logicians would be perfectly happy to say 'proving' here.

I think most logicians would be perfectly happy to say "proving" here.

Well, when I heard Angus talk he was keen to emphasise that it would not be a complete proof, but would only focus on the major bits of machinery needed. So it seems polite to echo the official line!

Of course one would really want to do this for Modularity Theorem, not just that piddling spinoff called Fermat's Last Theorem.

Yes - my notes from the talk are elsewhere, but I think his main focus is indeed on the central modularity result (I don't know whether he addresses the full theorem, or just the case needed for FLT).

In any case, he claims that it is effectively Π^0_1, and provable in PA.

Posted by: Richard Elwes

Regarding FLT, nLab has a short section [http://ncatlab.org/nlab/show/effects+of+foundations+on+%22real%22+mathematics#fermats_last_theorem_3] on this. So any findings to be added there. It mentions Colin McLarty's research.

I have also heard Angus MacIntyre on a sketch of a proof that PA suffices. He seems to have given a number of talks on this, e.g., here [http://www.cs.ox.ac.uk/seminars/128.html] and here [http://www.cs.ox.ac.uk/seminars/355.html], the later mentioning a discussion on FOM.

There's a paper by Jeremy Avigad – Number theory and elementary arithmetic [http://www.andrew.cmu.edu/user/avigad/Papers/elementary.pdf] – which should interest you.

Posted by: David Corfield

McLarty has recently shown [http://arxiv.org/abs/1102.1773] (I believe) that finite-order arithmetic is sufficient to define pretty much all of Grothendieck-style algebraic geometry necessary for arithmetic questions. nth-order arithmetic admits quantification over Pn(N), the n-times iterated power set for some given n. The n needed depends on the problem in question, and the hope is that $n < 2$ (PA or weaker) is sufficient for FLT, or even the modularity theorem (since there is a proof of the Modularity Theorem which is simpler than Wiles' original proof of the semistable case).

The trick is defining derived functor cohomology for sheaf coefficients. All the algebra content is apparently very basic from a logic point of view.

Posted by: David Roberts

So, I just spent a bit playing around with $I\Delta_0$ to get a sense for it. I wanted to build a Gödel coding, and I found I needed the following lemma (quantifiers range over positive integers):

$$\forall n \exists N \forall k \leq n \exists d : kd = N$$

Easy enough in PA; it's a simple induction on n. But in IΔ_0 I can't make that induction because there is no bound on N. (There's also no bound on d, but *I* can fix that by changing the statement to $\exists d \leq N$; this is also true and trivially implies the above.) I can't fix it by adding in $N \leq n^{100}$ because that's not true; the least such N is of size $\approx e^n$. I can't write $N \leq 4^n$ because I don't have a symbol for exponentiation. Anyone want to give me a tip as to how to prove this in IΔ_0?

Posted by: David Speyer

That's a great puzzle, David! I'm not very good at these things, so I hope someone materializes who can help you out. In the meantime here are some references that might (or might not provide useful clues. At least I found they're somewhat interesting.

First:

- Chris Pollet, Translating IΔ_0 + exp proofs into weaker systems. [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.7317]

I'm guessing you could do what you want in IΔ_0 + exp, but you're struggling to do it in IΔ_0. A few intriguing quotes:

> Of the commonly studied bounded arithmetic theories, IΔ_0 + exp, the theory with induction for bounded formulas in the language of 0, S, $+$, $\times$ together with the axiom saying the exponential function is total, is one of the more interesting...
>
> Wilkie–Paris have shown several interesting connections between IΔ_0 + exp and weaker theories. They have shown IΔ_0 + exp cannot prove Con(Q)...
>
> Despite the fact that IΔ_0 + exp is not interpretable in IΔ_0, it is known if IΔ_0 + exp proves $\forall x : A(x)$ where A is a bounded formula then IΔ_0 proves
>
> $$\forall x(\exists y : y = 2^x_k) \implies A(x))$$
>
> Here 2^x_k is a stack of 2's k high with an x at top.

Here k depends on x in some way. I guess he's saying that while IΔ_0 can be used to describe a relation deserving of the name $y = 2^x_k$, it can't prove that exponentiation is total, so it can't prove there exists a y such that $y = 2^x_k$. So, we need to supplement its wisdom for it to prove something similar to $\forall x A(x)$. Or in his words:

> Intuitively, this results says: given x, if IΔ_0 knows a big enough y exists then it can show $A(x)$ holds.

Of course you don't want to resort to a trick like this!

Posted by: John Baez

That's a great puzzle, David! I'm not very good at these things, so I hope someone materializes who can help you out. In the meantime here are some references that might (or might not provide useful clues. At least I found they're somewhat interesting.

First:

- Chris Pollet, Translating $I\Delta_0$ + exp proofs into weaker systems. [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.7317]

I'm guessing you could do what you want in $I\Delta_0$ + exp, but you're struggling to do it in $I\Delta_0$. A few intriguing quotes:

> Of the commonly studied bounded arithmetic theories, $I\Delta_0$ + exp, the theory with induction for bounded formulas in the language of 0, S, $+$, $\times$ together with the axiom saying the exponential function is total, is one of the more interesting...
>
> Wilkie–Paris have shown several interesting connections between $I\Delta_0$ + exp and weaker theories. They have shown $I\Delta_0$ + exp cannot prove Con(Q)...
>
> Despite the fact that $I\Delta_0$ + exp is not interpretable in $I\Delta_0$, it is known if $I\Delta_0$ + exp proves $\forall x : A(x)$ where A is a bounded formula then $I\Delta_0$ proves
> $$\forall x(\exists y : y = 2x^k) \Leftrightarrow A(x))$$
> Here $2x^k$ is a stackof 2's k high with an x at top.

Here k depends on x in some way. I guess he's saying that while $I\Delta_0$ can be used to describe a relation deserving of the name $y = 2x^k$, it can't prove that exponentiation is total, so it can't prove there exists a y such that $y = 2x^k$. So, we need to supplement its wisdom for it to prove something similar to $\forall x : A(x)$. Or in his words:

Intuitively, this results says: given x, if $I\Delta_0$ knows a big enough y exists then it can show $A(x)$ holds.

Of course you don't want to resort to a trick like this!

Posted by: John Baez

I think you can bet your sweet bippy that he himself knows how to do it. :-) Hey, maybe someone should ask on Math Overflow!

Posted by: Todd Trimble

It looks like the conversation has moved on. In case anyone is still puzzled, let me spell out what Kaveh is saying:

My statement cannot be proved in $I\Delta_0$ because $I\Delta_0$ is polynomially bounded.

I think I understand what this means now. Suppose that $\mathrm{I}\Delta_0$ proves

$$\forall n \exists m : \ldots\ldots$$

where the ellipsis are any grammatical statement about n and m. Then there is some polynomial $p(n)$ there exists an m with $m \leq p(n)$.

This is not true for my statement! The smallest valid N is LCM(1,2,...,n),which is $\approx e^n$. (The more obvious choice of N is $n!$, which is even bigger.) So this is a great example of a sentence which is true (as a statement about ordinary integers) and grammatical in $\mathrm{I}\Delta_0$, but not provable in $\mathrm{I}\Delta_0$, on account of the fact that it involves a fast growing function.

This example really helps me understand the more complicated examples of statements which are known to be undecidable in PA because of fast growing functions, like the Paris-Huntington theorem. I always run into a psychological roadblock with examples like Paris-Huntington, because the encoding of those statements into formal language is so complex. This example is straightforwardly a number theoretic statement, so I think I'll use it as my standard example of a statement which is undecidable for growth rate reasons in the future.

I'll point out that there is plenty of stuff which is provable in $\mathrm{I}\Delta_0$. I got through showing "if x divides y then $x \leq y$", "every positive integer is either of the form $2k$ or $2k+1$", "if a divides c and b divides c, then $\mathrm{LCM}(a,b)$ divides c", and several other standard examples of induction in elementary number theory before trying this one.

Posted by: David Speyer

I have two naive questions:

On the wikipedia page "ordinal analysis", RFA (rudimentary function arithmetic) is mentioned as having proof-theoretic ordinal $omega^2$, but nothing is said about it. Has anyone here heard of it? Is it EFA minus exponentiation?

Even if some systems may be too weak to be assigned proof-theoretic ordinals, is it possible to make sense of "if that system had a proof-theoretic ordinal in any reasonable sense, then this ordinal would be..."? In view of the wikipedia page on the Grzegorczyk hierarchy (which gives systems of strength ω^n), it is tempting to say that Presburger arithmetic "should" have strength omega.

Posted by: ben

While naive, these questions are not sufficiently naive for me to answer them. So, I hope someone else can! They're interesting.

Posted by: John Baez

What's an interesting result about PRA?

It is suggested that PRA is an upper bound for what Hilbert considered to be finitistic reasoning.

> Is there some other well-known way to rank them?

There are (e.g. by the class of their provably total functions), but I guess you want something similar to ordinal analysis. In that case check Arnold Beckmann [http://cs.swan.ac.uk/~csarnold/]'s project on dynamic ordinal analysis [http://cs.swan.ac.uk/~csarnold/amllcc/give-page.php?2] .

> How interesting can they get before the hand of Gödel comes down and smashes them out of existence?

For most purposes the bounded arithmetic theory V^0 (which is quite similar to $I\Delta_0$) is the natural starting point. The provably total functions of V^0 are exactly AC^0 functions (the smallest complexity class complexity theorist usually consider). For comparison, provably total functions of $I\Delta_0$ are Linear Time Hierarchy (LTH). V^0 is capable of talking about sequences using a better encoding that Gödel's beta function (write the numbers in the sequence in binary, add 2 between each consecutive pair, read in base 4). It can also check if a given number encodes the computation of a given Turing machine on a given input.

But a more natural theory to work with might be VTC^0 whose provably total functions are complexity class TC^0 which can also parse syntax. See Cook and Nguyen [http://www.cambridge.org/gb/knowledge/isbn/item2708116/?site_locale=en_GB] (draft [http://www.cs.toronto.edu/~sacook/homepage/book/]) for more information.

I think self-verifying theories [http://en.wikipedia.org/wiki/Self-verifying_theories] that can prove their own consistency (in the usual formalization) are artificial. For more information about them see:

Dan Willard, "Self Verifying Axiom Systems, the Incompleteness Theorem and the Tangibility Reflection Principible", Journal of Symbolic Logic 66 (2001) pp. 536-596.

Dan Willard, "An Exploration of the Partial Respects in which an Axiom System Recognizing Solely Addition as a Total Function Can Verify Its Own Consistency", Journal of Symbolic Logic 70 (2005) pp. 1171-1209.

I just spent a bit playing around with $I\Delta_0$ to get a sense for it. I wanted to build a Gödel coding...

You cannot prove that in $I\Delta_0$ because that would give a exponentially growing function while $I\Delta_0$ is a polynomially bounded theory.

On the wikipedia page "ordinal analysis", RFA (rudimentary function arithmetic) is mentioned as having proof-theoretic ordinal ω^2, but nothing is said about it.

Rudimentary sets are defined in Smullyan 1961. They are essentially Δ_0 =LTH. I am not sure about the theory RFA but I would guess it is essentially IΔ_0. EFA is IΔ_0 + EXP (where EXP expresses that the exponential function is total).

Even if some systems may be too weak to be assigned proof-theoretic ordinals… See above (the part about Beckmann's research).

Posted by: Kaveh

Thanks for posting this, John.

(Small quibble though - you have "Q only has inductive definitions of addition, multiplication and exponentiation", but Q lacks the primitive recursive defn for exponentiation. Those axioms would be:

$$\forall x : (x^0 = S(0)) \tag{30}$$
$$\forall x \forall y : (x^{S(y)} = x \times x^y) \tag{31}$$

But in standard logic, simply assuming a function symbol more or less presupposes the totality of corresponding function, i.e., exponentiation in this case. I.e., if we have a primitive binary symbol (i.e., x^y), then $\forall x \forall y \exists z(z = x^y)$ is a theorem of logic! For if f is, say, a 1-place function symbol, then $\vdash \forall x : (f(x) = f(x))$. And this gives us $\vdash \forall x \exists y : (y = f(x))$. Quick indirect proof: suppose $\exists x \forall y(y \neq f(x))$. So, $\forall y : (y \neq f(c))$, by introducing a new constant c; which gives the contradiction $f(c) \neq f(c)$.)

When Joel David Hamkins says that any weak system in the hierarchy IΣ_n simulates computation, I think (I am guessing) he just means that any recursive function is representable in Q and its extensions. E.g., if $f : \mathbb{N}^p \to \mathbb{N}$ is a partial recursive function, then there is an LA-formula $\phi(y, x_1, \ldots, x_p)$ such that, for all k, n_1,…, $n_p \in \mathbb{N}$,

$$\text{if } \ k = f(n_1, \ldots, n_p), \ \text{ then } \ Q \vdash \forall y : (y = \underline{k} \Leftrightarrow \phi(y, n_1, \ldots, n_p))$$

In particular,exponentiation, $f(a, b) = a^b$,is recursive. So, there is an LA-formula Exp(y, x_1, x_2) such that, for all n, m, $k \in \mathbb{N}$,

$$\text{if } \ k = n^m \text{then} Q \vdash \forall y : (y = \underline{k} \Leftrightarrow \text{Exp}(y, \underline{n}, \underline{m}))$$

So Exp(y, x_1, x_2) represents exponentiation. However, Q cannot prove it total. I.e., for any such representing formula Exp

$$Q \nvdash \forall x_1 \forall x_2 \exists y : \text{Exp}(y, x_1, x_2)$$

It's a long time since I worked through some of the details of bounded arithmetic, and my copy of Hayek and Pudlack is in Munich. So I can't see immediately how to give a model for this. Still, Q is very, very weak and here is a simple non-standard model. (From Boolos and Jeffrey's textbook). Let

$A = dom(\square) = \omega \cup \{a, b\}$, where a and b are new objects not in ω. These will behave like "infinite numbers". We need to define functions $S^\square$, $+^\square$ and $\times^\square$ on A interpreting the LA-symbols S, $+$ and $\times$. Let $S^\square$ have its standard values on $n \in \omega$ (i.e., $S^\square(2) = 3$, etc.), but let $S^\square(a) = a$ and $S^\square(b) = b$. Similarly, $+$ and $\times$ are interpreted standardly on ω, but one can define an odd multiplication table for the values of $a +^\square a$, $a +^\square b$, $a \times^\square b$, etc. Then one proves $\square \models Q$, even though $\square \not\approx \mathbb{N}$. This model is such that,

1. $\square \not\models \forall x \forall y : (x + y = y + x)$
2. $\square \not\models \forall x \forall y : (x \times y = y \times x)$

So, this tells us that Q doesn't prove that $+$ and $\times$ are commutative.

I don't think this simple model $\square$ with two infinite elements, a and b, is enough to show that Q doesn't prove that exponentiation is total.

The idea would be to find a model $\mathcal{B} \models Q$ such that $B \not\models \forall x_1 \forall x_2 \exists y : \phi(y, x_1, x_2)$, for any LA-formula $\phi(y, x_1, x_2)$ that represents $f(a, b) = a^b$ in Q. I don't know off-hand what such a model $\mathcal{B}$ looks like though.

Posted by: Jeffrey Ketland

On one of Kaveh's points, another property of PRA is that if ϕ is a $\Pi_1$1-sentence, then:

$$I\Sigma_1 \vdash \phi \text{ iff } PRA \vdash \phi$$

(Parsons 1970). Yes, Tait has argued that PRA represents the upper limit on what a finitist should "accept". However, I think that Kreisel had argued earlier that it should be PA.

Posted by: Jeffrey Ketland

Here's another system (call it FPA) which proves its own consistency. Work in 2-nd order logic, with predicative comprehension. Let 0 be a constant; let N be a (1-ary) predicate, meant to represent being a natural number; and let S be a (2-ary) relationship, meant to represent the successor relationship. Do <u>not</u> assume the totality of S. Instead assume

1. S is functional, i.e. Nx and Sx, y and Sx, z implies $y = z$
2. S is one-to-one, i.e. Nx and Ny and Sx, z and Sy, z implies $x = y$
3. For all n, not $Sn, 0$
4. Induction (full induction, as a schema)

Because the totality of S is not assumed, FPA has the singleton model $\{0\}$. It also has all the initial segments as models, as well as the standard model (well, whichever of those models actually exist). In a nutshell, FPA is "downward"; if you assume that a number n exists, then all numbers less than n exist and behave as you expect. Most of arithmetic is, in fact, "downward", so FPA can prove most familiar propositions, or at least versions of them. It can prove (unlike Q) the commutative laws of addition and multiplication. It can prove Quadratic Reciprocity. It cannot prove that there are an infinite number of primes (it cannot even prove the existence of 2, after all), but it can prove that between $n/2$ and n for any $n > 2$, there always exists a prime. It's not far-fetched to think that FPA can prove Fermat's Last Theorem. So, mathematically anyway, it's pretty strong. (Still it's neither stronger nor weaker than Q. It's incomparable, because it assumes induction, which Q does not, but does not assume the totality of successoring, which Q does.)

In particular FPA can talk about syntax because syntactical elements can be defined in a downward way. Something is a term if it can be decomposed in a particular way. Something is a wff if it can be decomposed in a particular way. Etc.

Now, to prove its own consistency, it suffices for FPA to show that the assumption of a proof in FPA of "not 0=0" leads to a contradiction. But a proof is a number (in Gödel's manner of representing syntactical elements) and, in fact, a very large number. This large number then gives enough room, in FPA, to formalize truth-in-the-singleton-model and to prove that any sentence in the inconsistency proof must be true. But "not 0=0" isn't true. Contradiction! Therefore FPA has proven its own consistency.

Here's a link to a book-long treatise, if it interests anyone:

- Andrew Boucher, Arithmetic Without the Successor Axiom [http://www.andrewboucher.com/papers/arith-succ.pdf].

It's possible to formalize everything in a first-order system, if the second-order is bothersome for some.

Posted by: t

Wow, that's quite interesting! Thanks!

Since this post grew out of our earlier discussions of ultrafinitism [http://golem.ph.utexas.edu/category/2011/09/the_inconsistency_of_arithmeti.html] , I couldn't help noting that this axiom system should be an ultrafinitist's dream, since you can take any model of Peano Arithmetic, throw out all numbers $> n$, and be left with a model of this one!

Indeed I see Andrew Boucher writes:

> Most sub-systems of Peano Arithmetic have focused on weakening induction. Indeed perhaps the most famous sub-system, Robinson's Q, lacks an induction axiom altogether. It is very weak in many respects, unable for instance to prove the Commutative Law of Addition (in any version). Indeed, it is sometimes taken to be the weakest viable system; if a proposition can be proved in Q, then that is supposed to pretty much established that all but Berkleyan skeptics or fools are compelled to accept it.
>
> But weakness of systems is not a linear order, and F is neither stronger nor weaker than Q. F has induction, indeed full induction, which Q

> does not. But F is ontologically much weaker than Q, since Q supposes the Successor Axiom. Q assumes the natural numbers, all of them, ad infinitum. So in terms of strength, F and Q are incomparable. In actual practice, F seems to generate more results of standard arithmetic; and so in that sense only, it is "stronger".
>
> One of the most important practitioners of Q has been Edward Nelson of Princeton, who has developed a considerable body of arithmetic in Q. While Nelson's misgivings with classical mathematics seemed to have their source in doubts about the existence of the natural numbers, the brunt of his skepticism falls on induction, hence his adoption of Q. "The induction principle assumes that the natural number series is given." [p. 1, Predicative Arithmetic] Yet it would seem that induction is neither here nor there when it comes to ontological supposition. Induction states conditions for when something holds of all the natural numbers, and says nothing about how many or what numbers there are. So a skeptic about the natural numbers should put, so to speak, his money where his doubts are, and reject the assumption which is generating all those numbers — namely the Successor Axiom – and leave induction, which those doubts impact at worst secondarily, alone.

He also mentions other systems capable of proving their own consistency:

> A number of arithmetic systems, capable of proving their own consistency, have become known over the years. Jeroslow [Consistency Statements] had an example, which was a certain fixed point extension of $Q \vee \forall x \forall y : (x = y)$. More recently, Yvon Gauthier [Internal Logic and Internal Consistency] used indefinite descent and introduced a special, called "effinite", quantifier. And Dan Willard [Self-Verifying Axiom Systems] has exhibited several cases, based on seven "grounding" functions. These systems lack a certain naturalness and seem to be constructed for the express purpose of proving their own consistency. Finally, Panu Raatikainen constructed what is effectively a first-order, weaker variant of F; this system can prove that it has a model [Truth in a Finite Universe], but its weakness does not allow the author to draw conclusions about intensional correctness and so it seems to fall short of the ability to prove its own self-consistency.

Posted by: John Baez

I remember Andrew Boucher describing his theory F on sci.logic years back; the problem is that it doesn't interpret syntax (e.g., Tarski's TC). (The current state of play is that TC is interpretable in Q.)

The language LF is a second-order language with a binary relation symbol S instead of the usual unary function symbol. Even with this small modification (so as to drop the automatic existence of successors), the syntax of LF is still that of a standard language, with, say, symbols 0, S, $=$ and $\neg$, $\rightarrow$, $\forall$ and variables v, v' , v'', etc. It is straightforward to prove, based merely on the description of LF and the usual assumptions about concatenation, that:

$$|LF| = \aleph_0.$$

So the language LF itself is countably infinite. Denying the existence of numbers while asserting the existence of infinitely many syntactical entities is incoherent, as one of Gödel's basic insights is: syntax = arithmetic.

Suppose we then begin to try and interpret the syntax of LF in F itself. Ignore the second order part, as it introduces needless complexities. In the metatheory, suppose we assign Gödel codes as follows:

$$\begin{array}{llll} \#(0)=1 & \#(S)=2 & \#(=)=3 & \#(\neg)=4 \\ \#(\rightarrow)=5 & \#(\forall)=6 & \#(v)=7 & \#(')=8 \end{array}$$

Incidentally, this already goes beyond F itself, as the metatheory already implicitly assumes the distinctness of these numbers. How would this be done, given that one cannot even prove the existence of 1?

In LA, we encode any string (sequence of primitive symbols) as the sequence of its codes, and we encode a sequence $(n_1, \ldots, n_k)$ of numbers as a sequence number, e.g., as

$$\langle n_1, \ldots, n_k \rangle = (p1)^{n_1+1} \times \ldots \times (pk)^{n_k+1}.$$

For example, the string $\forall\forall S$ is really the sequence $(\forall, \forall, S)$, and is coded as the sequence (6, 6, 2), which becomes the sequence number $27 \times 3^7 \times 5^3$.

But what, in F, is the corresponding numeral for any expression of the language LF? In the usual language LA of arithmetic, an expression ε with code n is assigned the numeral $\underline{n}$, written $[\varepsilon]$, which is $S \ldots S0$. That is, 0 prefixed by n occurrences of S, where S is a function symbol. (Can't get "ulcorner" to work!)

How would this work in F? Consequently, F does not numeralwise represent non-identity of syntactical entities.

For example, in syntax we have

A: "The quantifier $\forall$ is distinct from the conditional $\rightarrow$".

Under the coding above, this becomes

$$A' : \underline{6} \neq \underline{5}$$

which is trivially provable in Q.

Now it's very unclear to me how one even expresses $\underline{6} \neq \underline{5}$ in LF. But however it is done, we get that

$$F \nvdash \underline{6} \neq \underline{5}$$

A requirement on a theory T that interprets syntax is that, for expressions ε_1 , ε_2 , we have unique singular terms $[\varepsilon_1]$, $[\varepsilon_2]$ such that,

$$\text{if } \varepsilon_1 \neq \varepsilon_2 \text{ then } T \vdash [\varepsilon_1] \neq [\varepsilon_2]$$

But F doesn't give this. Instead, we have

$$F \nvdash [\forall] \neq [\rightarrow]$$

So, alleged "agnoticism" about numbers has become "agnoticism" about syntax.

Which contradicts the non-agnoticism of the description of syntactical structure of LF itself.

There is no faithful interpretation of the syntax of LF into F. So, syntactical claims about the properties of F cannot be translated into F. The meta-theory of the syntax of F already assumes an infinity of distinct syntactical entities. In particular, claims about consistency cannot be translated into F.

Posted by: Jeffrey Ketland

Thanks for your comment. Unfortunately, I don't think it's quite right and F does indeed interpret syntax adequately, so that it does express notions of consistency.

First, just as F is agnostic about the infinity of the natural numbers, it is agnostic about the infinity of the syntax. The infinity of syntax comes from assuming that there are an infinite number of variables; F doesn't make this assumption. I guess a stickler might say this is no longer second- (or first-) order logic because these assume that there are an infinite number of variable symbols. But I would hope most would agree this is not an essential feature of the logic.

JK: "Incidentally, this already goes beyond F itself, as the metatheory already implicitly assumes the distinctness of these numbers. How would this be done, given that one cannot even prove the existence of 1?"

While one cannot prove that 1 exists, it is possible to prove that anything which *is* one is unique (and so distinct). That is, it is possible to define a predicate one(x) as (Nx and $S0, x$). It is not possible to prove that there exists x s.t. one(x), but it *is* possible to prove that $(x)(y)$(one(x) and one(y) implies$x = y$) So proof of existence, no; proof of distinctness, yes. One can define two(x) as

$$Nx \text{ and there exists } y \text{ such that one}(y) \text{ and } Sy, x$$

And so forth as far as one wants or has energy to go.

Moreover, one can define the concepts of odd and even. One defines even(x) iff Nx and (there exists $y)(y + y = x)$. Again, no assertion that one can prove that even(x) or odd(x) for any x. But one *can* prove that there is no x such that both even(x) and odd(x). Again, existence no, distinctness yes.

So one can represent the syntax. Define predicates one, two, three, ..., ten. Define Big(x) as Nx and not one(x) and not two(x) and ... and not ten(x). Then x represents a left parentheses if $x = 0$. x represents a right parenthesis if one(x). x represents the implication sign if two(x). x represents the negation sign if three(x). x represent the equal sign if four(x). And so forth. x represents a small-letter variable if Big(x) and even(x). x represents a big-letter variable if Big(x) and odd(x).

One gives the usual recursive definitions to syntactical entities like AtomicWff(x) and Proof(x). Again, one cannot show there exist any x such that AtomicWff(x). But one can show that, *if* AtomicWff(x), then x has all the properties that it should have.

So, given that x cannot prove there exist any syntactical entities, how can it prove its own consistency? Because consistency means there is no proof of "not 0 = 0". So a proof of consistency is not a proof that something exists, but a proof that something does not exist. It *assumes* the existence of a syntactical entity, in this case a proof of "not 0 = 0", and shows that the assumption of the existence of this entity leads to a contradiction. Thus F is able to prove a system's consistency. (What F cannot prove is prove that a system is inconsistent; because then it would have to prove that there exists something, namely a proof of "not 0 = 0", and that it cannot do.)

Anyway, all this is described in gory detail in the link that I gave.

"The infinity of syntax comes from assuming that there are an infinite number of variables; F doesn't make this assumption."

This is not correct. A propositional language L with a single unary connective $\neg$ and a single atom p has infinitely many formulas. So, $\text{Form}(L) = \{p, \neg p, \neg\neg p, \ldots\}$ and

$$|\text{Form}(L)| = \aleph_0$$

The potential infinity here is a consequence of the implicit assumptions governing the concatenation operation *. Formulas are, strictu dictu, finite sequences of elements of the alphabet. It is assumed that sequences are closed under concatenation. If α, β are sequences, then $\alpha * \beta$ is a sequence.

"I guess a stickler might say this is no longer second- (or first-) order logic because these assume that there are an infinite number of variable symbols."

As noted, it has nothing to do with variables. The strings of the propositional language L above form an ω-sequence. In general, if α and β are strings from the language L, then $\alpha * \beta$ is a string. This is simply assumed.

"But I would hope most would agree this is not an essential feature of the logic." That any standard language L for propositional logic (containing at least one atom and one connective) or first-order logic has cardinality $\aleph_0$ is usually a preliminary exercise in logic.

Posted by: Jeffrey Ketland

Well, obviously I wouldn't assume the totality of the concatenation operator.

"As noted, it has nothing to do with variables." This is not correct. Your language is infinitary if the number of variables is infinitary.

"That any standard language L for propositional logic (containing at least one atom and one connective) or first-order logic has cardinality $\aleph_0$ is usually a preliminary exercise in logic."

Of course. But it's not an essential feature of the logic, in the sense one could give an adequate description of the logic without this feature.

Posted by: t

Indeed, the infinitude of variable symbols is entirely a red herring. For those who want a finite alphabet, the standard (AIUI) solution is to have a symbol x and a symbol $'$ such that x is a variable and v' is a variable whenever v is. (Thus the variable symbols are x, x', x'', etc.)

Posted by: Toby Bartels

---

t, "One gives the usual recursive definitions to syntactical entities like AtomicWff(x) and Proof(x)."

What, exactly, are these entities AtomicWff(x) and Proof(x)? How many symbols do they contain? Are they distinct? How does one prove this? Have you ever tried to estimate how many symbols occur in the arithmetic translation of the sentence

"the formula $\forall x : (x = x)$ is the concatenation of $\forall x$ with $(x = x)$"?

You're assuming something that you then claim to "doubt". You do not, in fact, "doubt" it: you assume it.

One never says, in discussing the syntax of a language, "if the symbol $\forall$ is distinct from the symbol v ...". Rather, one says, categorically, "the symbol $\forall$ is distinct from the symbol v". The claim under discussion amounts to the view that one ought to be "agnostic" about the distinctness of, for example, the strings $\forall x : (x = 0)$ and $\forall y : (y \neq 0)$.

One <u>can</u> write down a formal system of arithmetic which has a "top" – called "arithmetic with a top". But it is not as extreme as F. Such theories have been studied in detail by those working in computational complexity and bounded arithmetic (see, e.g., the standard monograph by Hajek and Pudlak, which I don't have with me).

See, e.g., this: [http://www.math.cas.cz/~thapen/nthesis.ps]

Agnosticism about numbers = agnosticism about syntax. You can't have your "strict finitist" cake, while eating your syntactic cake, as they're the same cake!

Jeff

"What, exactly, are these entities AtomicWff(x) and Proof(x)?"

They are syntactical entities. I could write them down for you explicitly here, but as you can probably tell, I'm not gifted writing down logical symbols in these comments. Or you can look at the top of page 110 and on page 111 of the link, where you will find them already written down explicitly.

"Are they distinct? How does one prove this?"

I'm not sure whether you are talking about meta-theory or theory. In the theory F, if you assume there exists something which represents AtomicWff(x) and another thing which represents Proof(x), then you would be able to prove these things distinct, because their ith symbols will be different for some i. But one doesn't need to prove this, certainly not in the proof that the system is consistent. In the meta-theory the two syntactical entities are different, and you see this by writing them down.

“You’re assuming something that you then claim to “doubt”.”

No I’m not. Again you seem to be confusing meta-theory with theory, or assuming that there must be some tight connection between them. You can’t prove that 1 exists in F. You agree, right? So F makes no assumptions that I doubt. Sure I can write down a formula in F which has more than one symbol. So? That has no bearing on what F does or does not assume. In any case my doubts are not that 1 exists, or that 10 exists, but that *every* natural number has a successor. And the fact that I can write down a formula with 1 million symbols (well, if you pay me enough) cannot erase my doubts, nor has any bearing on these doubts.

“One never says, in discussing the syntax of a language, “if the symbol $\forall$ is distinct from the symbol $v \ldots$”.

Your manner of expression is again not clear. “One never says. . . ” Are you talking theory, meta-theory, what? F can prove: “if the symbols $\forall$ and v exist (or to be more precise, if the numbers used to represent them exist), then they are distinct.”

“Rather, one says, categorically, “the symbol $\forall$ is distinct from the symbol v”.” Well, F cannot prove that the numbers representing the symbols exist. But, in order to prove the consistency of itself, F doesn’t need to. Proving the consistency of a system, does not require F to show that anything exists. Rather, it has to show that something does *not* exist.

“The claim under discussion amounts to the view that one ought to be “agnostic” about the distinctness of, for example, the strings $\forall x : (x = 0)$ and $\forall y : (y \neq 0)$.”

No, no, no. For some reason you are hung up on distinctness. F can prove distinctness. Again, it can prove that if these strings (or more precisely, the sequences representing them) exist, then they are distinct. So F is most certainly not agnostic about their distinctness. All that F cannot prove is: the strings exist.

“One can write down a formal system of arithmetic which has a “top” - called “arithmetic with a top”. But it is not as extreme as F. ”

Again, you are making imprecise claims. F allows for the possibility of the standard model. Formal systems with a “top” do not. Everything that F proves will be true in PA. There are things that “top” formal systems prove that are false in PA. So what on earth does “extreme” mean?

Posted by: t

“So what on earth does ‘extreme’ mean?”

A theory of syntax that doesn’t prove that $\forall$ is distinct from $=$?

ROTFL. Ok, you win. I'll grant you that F doesn't "prove that symbols are distinct" in the sense of "prove that they exist." And I'll grant you that this means that its "theory of syntax" is "extreme."

Still, in order to prove that a system is consistent, one can work with an "extreme" "theory of syntax" which doesn't "prove that symbols are distinct" because, to prove a system is consistent, one needs to prove that something *doesn't* exist, not to prove that something *does*. (In your terminology, would this be, "one needs to prove that something isn't distinct, not to prove that something is"??) If you or anyone else thinks that F is inconsistent, then you must come up with a proof of "not $0 = 0$". And, by the mere fact of that proof supposedly existing, F can show that it is able to model truth-in-$\{0\}$ for the statements in the proof and so that "not $0 = 0$" cannot be a statement in the proof. Contradiction. Therefore you, or anyone else, cannot come up with a proof. And since all this reasoning can be done in F, F can prove its own consistency. It's that simple.

Posted by: t

---

t, I see what you wish to do with this theory F. But you lack numerals, since S is not a function symbol. So, instead, for example, one might express $0 \neq 1$ by a formula

$$\forall x \forall y : ((\underline{0}(x) \wedge \underline{1}(y)) \implies x \neq y)$$

where the formulas $\underline{n}(x)$ are given by a recursive definition

$$\begin{aligned} \underline{0}(x) &\Leftrightarrow x = 0 \\ \underline{n+1}(x) &\Leftrightarrow \exists y : (\underline{n}(y) \wedge S(x, y)) \end{aligned}$$

So, to assert the existence of the number 7, for example, you have $\exists x : (\underline{7}(x))$. And, presumably, for all $k \leq n$,

$$F \vdash \exists x : (\underline{n}(x)) \implies \exists x(\underline{k}(x))$$

Then define $\mathrm{NotEq}_{n,m}$ to be the formulas

$$\forall x \forall y : ((\underline{n}(x) \wedge \underline{k}(y)) \implies x \neq y)$$

Then I believe one has: for all n, $k \in \mathbb{N}$,

$$\text{if } n \neq k, \text{ then } F \vdash \mathrm{NotEq}_{n,k}$$

As for syntactic coding, since $\forall$ is coded as 6 and $=$ as 3, then F can define, e.g.,:

$$\begin{aligned} \underline{\forall}(x) &\Leftrightarrow \underline{6}(x) \\ \underline{=}(x) &\Leftrightarrow \underline{3}(x) \end{aligned}$$

Then (I think), F does prove the distinctness of $\forall$ and $=$ in a conditional manner, namely,

$$F \vdash \forall x \forall y : ((\underline{\forall}(x) \wedge \underline{=}(y)) \implies x \neq y)$$

But no, I don't accept that F "proves its own consistency". Just to begin with, one doesn't have a proof predicate which strongly represents the proof relation for F.

And to return to the central issue, you are assuming the existence of a language L_F whose cardinality (the cardinality of its set of formulas) is $\aleph_0$. You're assuming this already in the metatheory. You already have $\aleph_0$ syntactical entities. What is the point of being "agnostic" about, say, the number 1 if you are already assuming, in your informal metatheory, the existence of $\aleph_0$-many syntactical entities? In other words, I am doubting your "agnosticism". You're simply trying to have your syntactic cake while eating (i.e., professing) the "strict finitism" cake. It doesn't work, because they are the same cake.

To repeat: form the point of view of ontology, interpretability, etc., syntax = arithmetic. The same thing. They can be modelled in each other. To "doubt" arithmetic while accepting syntax is incoherent.

To make it work, you need to develop a separate "strictly finite" syntax, for example, a la Quine and Goodman 1947. It would have to drop the totality of concatenation on syntactical entities. It really is not worth bothering with, as it doesn't work, though. At the very best, you simply end up reinventing, in a weird way, all the things that have been discussed countlessly many times in the very rich research literature about nominalism. See, for example,

Burgess, J and Rosen, G. 1997. A Subject with No Object. OUP. Jeff

Posted by: Jeffrey Ketland

"(a lot of things snipped)"

You clearly haven't read the linked paper, or even (I imagine) glanced over it, right? That doesn't seem to faze you in the least, though, in making various definitive pronouncements.

"Just to begin with, one doesn't have a proof predicate which strongly represents the proof relation for F."

Well, you will have to give a reasoned argument why (the technical notion of) representability is essential to (the intuitive notion of) expressability. Consider the simpler case of even(x), which can be defined in F as (there exists y)($y + y = x$). Because of F's ontological limitations, even(x) doesn't represent evenness. Yet even(x) clearly expresses the notion of evenness. I think you can be most succinct in your point by noting that the Hilbert-Bernays conditions of provability do not hold for the provability predicate in F. But as I mention in the linked paper, the Hilbert-Bernays conditions do not adequately capture the (intuitive) notion of provability.

"And to return to the central issue, you are assuming the existence of a language LF whose cardinality (the cardinality of its set of formulas) is $\aleph_0$."

If that's the central issue, then you are wrong, as I am not. Look, you obviously haven't read or thought hard about what I've done or written, so perhaps you should stop saying that I am making assumptions which I do not make. Right? That's only fair, right?

"It would have to drop the totality of concatenation on syntactical entities."

Obviously. I see now you have replied in another place about this, so I will now switch there.

Posted by: t

I'm trying to understand this discussion. It seems to me that Jeffrey Ketland is saying, roughly, that because our usual theory of syntax can prove that the system F has infinitely many formulas, while F has finite models (as well as infinite ones), the system F is "incoherent" as a theory of arithmetic. For example, he says:

> To repeat: form the point of view of ontology, interpretability, etc., syntax = arithmetic. The same thing. They can be modelled in each other. To "doubt" arithmetic while accepting syntax is incoherent.
>
> So the language LF itself is countably infinite. Denying the existence of numbers while asserting the existence of infinitely many syntactical entities is incoherent, as one of Gödel's basic insights is: syntax = arithmetic.

But this is puzzling in two ways. First of all, I don't think F "denies the existence of numbers": any model of Peano arithmetic will be a model of F, so you can have all the natural numbers you might want. There's a difference between denying something and not asserting something.

But more importantly, I don't really care whether F is "incoherent" from the point of view of "ontology" due to some claimed mismatch between the syntax of theory F (which has infinitely many formulas, according to standard mathematics) and the models F has (which include finite ones). "Incoherent" and "ontology" are philosophical notions, but I'm a mere mathematician. So I'm much more interested in actual theorems about F.

If these theorems are proved in a metatheory that can prove F has infinitely many formulas, that's fine! — just make sure to tell me what metatheory is being used. And if someone has proved some other theorems, in a metatheory that can't prove F has infinitely formulas — in other words, a metatheory that more closely resembles F itself — that's fine too! All I really want to know is what's been proved, in what framework.

But I guess it all gets a bit tricky around Gödel's 2nd incompleteness theorem. What does it mean for F to "prove its own consistency"? I guess it means something like this. (I haven't thought about this very much, so bear with me.) Using some chosen metatheory, you can prove

$$F \vdash \mathrm{Con}(F)$$

where Con(F) is some statement in F that according to the chosen metatheory states the consistency of F. The Hilbert-Bernays provability conditions [http://en.wikipedia.org/wiki/Hilbert%E2%80%93Bernays_provability_conditions] are supposed to help us know what "states the consistency of F" means, but if you want to use some other conditions, that's okay — as long as you tell me what they are. I can then make up my mind how happy I am.

From the reference above:

Let T be a formal theory of arithmetic with a formalized provability predicate Prov(n), which is expressed as a formula of T with one free number variable. For each formula ϕ in the theory, let $\#(\phi)$ be the Gödel number of ϕ. The Hilbert–Bernays provability conditions are:

1. If T proves a sentence ϕ then T proves Prov($\#(\phi)$).
2. For every sentence ϕ, T proves Prov($\#(\phi)$) $\implies \mathrm{Prov}(\#(\mathrm{Prov}(\#(\phi))))$.
3. T proves that Prov($\#(\phi \implies \Psi)$) and Prov($\#(\phi)$) imply Prov($\#(\Psi)$)

Posted by: John Baez

13 Some open questions and work to do

1. Search the following books to see if the AH can be "started" with the PR-sets and alternating quantifiers [16, 15, 4, 8, 14, 17, 1] – [9, 10, 7, 21]. $\leftarrow$
   - [16], Rogers: Chapters 14, 15. Hypersimple, creative, T-complete. Pages 78 (def.) 316, 330 (AH), 380, 396 (AnH).
   - [15], Odifreddi, pages 306, 341-349 (complete r.e.), 370, 372, 451 (complete in the AH), 362, 375, 381, 392, 393, 511-417, 438.
   - [10]: Kozen, arithmetic hierarchy.
   - [4]: Davis, Kleene hierarchy, Chapter 6.
   - [17]: Sipser, read Chapter 6, self reference. No hierarchies.
2. [2]: read Brandt (arithmetic hierarchy). $\leftarrow$
3. Modify the Kleene normal form to show that the following goal (see previous item) is possible: start from a PR-relation to generate all AH.
4. The set of recursive sets (and functions) is not enumerable; the set of PR-sets is. Characterization by a "model of computation" implies r.e.
5. The class Σ_1^0 (and Σ_1^{PR}) are recursively enumerable. Then, in the AH, only Δ – and possibly Δ_1^0, ... – are not r.e. However, the class PR is r.e.! $\leftarrow$
6. Are there non-complete problems in $\Sigma_1^0 \setminus \Delta$? References? Is there a Ladner-type theorem?
7. What are PR-sets, are there other definitions? Closure properties. Study the characterization of PR-sets by PR functions.
8. Codomain of a PR function: class Σ_1^0. Prove.

1. Suppose that $f'(\overline{x}, y)$ is the relation associated with the function f which is total but not PR. Can $f'(x, y, z)$ be PR? In particular, let $a(x, y)$ be the Ackermann function; is $a'(x, y, z)$ PR?
2. Let $f(x)$ be an unbounded and non decreasing PR function. Is the inverse function $f^{-1}(y) = \min\{x : f(x) \geq y\}$ PR? Particular case: $f(x) = \overset{x}{\uparrow}$.

3. Is the following function PR, where $a(x, y)$ is the Ackermann function?

$$a(m,n,d) = \begin{cases} 0 & \text{if the } d^{\text{th}} \text{ bit of } a(m,n) \text{ is } 0 \\ 1 & \text{if the } d^{\text{th}} \text{ bit of } a(m,n) \text{ is } 1 \\ 2 & \text{if } a(m,n) \text{ has less than } d \text{ bits} \end{cases}$$

References

[1] George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, 2007. Fifth Edition.

[2] Ulrike Brandt. Index sets in the arithmetical hierarchy. *Annals of Pure and Applied Logic*, 37:101–110, 1988.

[3] Computer Science Courses. Course CS 20a, Solution for homework #6, 2002. `http://courses.cms.caltech.edu/cs20/a/hw/hw6/solution/sol6.pdf`.

[4] Martin Davis. *Computability and Unsolvability*. Dover, 1985.

[5] Fran cois G. Dorais. (answer to a question), July 2011.

[6] Rod Downey and Lance Fortnow. Uniformly hard languages, 2001.

[7] Hans Hermes. *Enumerability, Decidability, Computability*. Springer-Verlag, 1969.

[8] Neil Immerman. *Descriptive Complexity*. Springer, 1999.

[9] Stephan Cole Kleene. *Introduction to Metamathematics*. North-Holland, 1952. Reprinted by Ishi press, 2009.

[10] Dexter C. Kozen. *Theory of Computation*. Texts in Computer Science. Springer, 2006.

[11] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, third edition, 2008.

[12] Armando B. Matos and António Porto. Ackermann and the superpowers (revised in 2011). *ACM SIGACT*, 12(Fall 1980), 1980/1991/2011.

[13] A. R. Meyer and D. M. Ritchie. The complexity of loop programs. *Proceedings of 22nd National Conference of the ACM*, pages 465–469, 1967.

[14] Bernard Moret. *The Theory of Computation*. Addison-Wesley, 1998.

[15] Piergiorgio Odifreddi. *Classical Recursion Theory – The Theory of Functions and Sets of Natural Numbers*. Studies in Logic and the Foundations of Mathematics. Elsevier North Holland, first edition, 1989. Second impression.

[16] Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press Cambridge, MA, 1987. Third printing (1992).

[17] Michael Sipser. *Introduction to the Theory of Computation*. PWS, 1997.

[18] Raymond Smullyan. *Godel's Incompleteness Theorems.* Oxford, 1992.

[19] Robert I. Soare. *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets.* Perspectives in Mathematical Logic. Springer, 1987.

[20] Robert I. Soare. *Computability Theory and Applications.* Perspectives in Mathematical Logic. To be published (Springer), 2014? Department of Mathematics, the University of Chicago.

[21] R. Sommerhalder and S.C. van Westrhenen. *The Theory of Computability: Programs, Machines, Effectiveness and Feasibility.* International Computer Science Series. Addison Wesley, 1988.