CHAPTER 6

Okay it's undecidable, but it can't be incomplete too (Cont'd)

3. What do you mean the dirty work starts now?

Let's try to compose ourselves, we have a goal:

> There is an algorithm, which eventually prints out every universally valid $\mathcal{L}_{Peano}$-formula.

In posh words:

> The set of all universally valid $\mathcal{L}_{Peano}$-formulas is recursively enumerable.

We'll get there in several steps:

- **Step 1.** Code any $\mathcal{L}_{Peano}$-formula into a natural number, in an effective way.
- **Step 2.** Prove that the set of axioms of our proof system is a primitive recursive set.
- **Step 3.** Code any $\vdash_{\mathcal{L}_{Peano}}$ proof into a natural number, in an efficient way.
- **Step 4.** Prove that the set of derivations of formulas from T_{PA} (or any recursive theory) is primitive recursive.
- **Step 5.** Use the basic facts about recursively enumerable sets we proved in the previous chapter to conclude.

To get there, we will use our sweet primitive recursive coding functions tuple and untuple and many more tools we developed earlier on.

Before we get into it though, you'll have to excuse the following lemma, which looks very technical, because it is:

Lemma 3.0.1 (Multiple recursion)**.** *Let $p, n \in \mathbb{N}$ and suppose that we are given primitive recursive functions $k_1, \ldots, k_n \in \mathcal{F}_1$, $g \in \mathcal{F}_p$ and $h \in \mathcal{F}_{p+n+h}$. If, for all $y > 0$ we have that $k_i(y) < y$, then the function f defined by:*

- $f(0, x_1, \ldots, x_p) = g(x_1, \ldots, x_p)$.

- $f(y, x_1, \ldots, x_p) = h\left(y, f(k_1(y), x_1, \ldots, x_p), \ldots, f(k_n(y), x_1, \ldots, x_p), x_1, \ldots, x_p\right)$

is primitive recursive.

What's the point here? Well, in our original definition of recursive definitions, we required that the argument in which we are defining the function by recursion decreased on every step. This doesn't quite fit our definition of recursion!

PROOF (SKETCH). A formal proof wouldn't be so hard. We've developed enough coding to using the tools of coding of sequences we've defined. Alternatively, invoking Church's thesis, it's enough to write pseudocode for a computer program that computes f! Indeed, this is an easy exercise. □

Great, now we can start.

3.1. Coding of formulas. Remember how, a while back we spent loads of time coding register machines into integers? Now we'll pull a similar trick, but we'll code $\mathcal{L}_{Peano}$-formulas into integers.

Our new brilliant idea is to code a term t as a triple of natural number (a, b, c) where:

- The third coordinate tells us if this term is an elementary term or a composite term.
- If the term is elementary, then the first coordinate tells us if it's $\underline{0}$ or a variable.
- If the term is composite, then the third coordinate tells us if it is a term involving $\underline{S}$, $\underline{+}$ or $\underline{\times}$ and the remaining coordinates store the (already coded) terms that appear.

Since we are arithmeticising syntax, we need to be somewhat more precise about syntax(!) from now on. In particular, we will assume that: $\mathsf{Var} = \{x_1, x_2, \ldots\}$ Word salad again... Here's the formal definition:

Definition 3.1.1. We define, for an $\mathcal{L}_{Peano}$ terms t the *Gödel number* of t, denoted $\#t$, by induction as follows:

(1) If t is $\underline{0}$ then $\#t = \mathsf{tuple}^3(0, 0, 0)$.

(2) If t is the variable x_n, then $\#t = \mathsf{tuple}^3(n, 0, 0)$.

(3) If $\#t_1$ and $\#t_2$ have already been defined, then

   (a) $\#(\underline{S}t_1) = \mathsf{tuple}^3(\#t_1, 0, 1)$

(b) $\#(t_1 \underline{+} t_2) := \mathsf{tuple}^3(\#t_1, \#t_2, 2)$

(c) $\#(t_1 \underline{\times} t_2) := \mathsf{tuple}^3(\#t_1, \#t_2, 3)$

Example 3.1.2. Way we're given the $\mathcal{L}_{Peano}$-term $\underline{0} \underline{+} \underline{S}(\underline{0})$. We go one step at a time. By definition, $\#\underline{0} = \mathsf{tuple}^3(0) = 0$. Now,

$$\#(\underline{S}(\underline{0}) = \mathsf{tuple}^3(0,0,1) = \mathsf{pair}((0,0),1) = \mathsf{pair}(0,1) = 2$$

so all in all:

$$\#(\underline{0} \underline{+} \underline{S}(\underline{0})) = \mathsf{pair}(\mathsf{pair}(0,2),2) = \mathsf{pair}(5,2) = 30.$$

Lemma 3.1.3. *The set* Term *of Gödel codes for $\mathcal{L}_{Peano}$-terms is primitive recursive.*

Proof. Since this is the first time we're coming across something like this though, maybe it's a good shout to actually give a (partial but formal) proof. The proof below is essentially our proof-template for all similar statements we come across in the future.

Let's think a little bit about what tuple^3 does to a triple (x, y, z). By definition, we have that:

$$\mathsf{tuple}^3(0,0,0) = 0, \text{ and } \mathsf{tuple}^3(1,0,0) = 1,$$

and it is easy to check that for all other (x, y, z) we have that $\mathsf{tuple}^3(x, y, z) > 1$. Moreover, for any $x > 1$ we have that $\mathsf{untuple}^3_i(x) < x$.

We can now define the characteristic function of Term as follows:

- $\chi(0) = 0$ [The number 0 represents the constant symbol $\underline{0}$].
- $\chi(1) = 1$ [The number 1 represents the variable x_1]
- If $x \geq 1$ then we use $\mathsf{untuple}^3_i$ as follows:
  - If $\mathsf{untuple}^3_3(x) = 0$ (i.e. we are in the variable case), then $\chi(x) = 1$ if and only if $\mathsf{untuple}^3_2(x) = 0$ [The only allowed codes with 0 in their last coordinate must have 0 in their second coordinate.]
  - If $\mathsf{untuple}^3_3(x) = 1$ then $\chi(x) = 1$ if and only if $\mathsf{untuple}^3_2(x) = 0$ and $\chi(\mathsf{untuple}^3_1(x)) = 1$ [Here we use recursion, we are saying that you're a valid code term of the form $\underline{S}t$ if your second coordinate is 0 and your first coordinate is a valid code for a term.]

  ...

In the definition of χ we didn't quite use primitive recursion, but we used the version of recursion we saw in Lemma 3.0.1, since at each point $\mathsf{untuple}^3_i(x) < x$. □

Exercise 3.1.4. Prove that the Gödel number function is injective on terms, i.e. if $\#t_1 = \#t_2$ then $t_1 = t_2$.

Now that we have coded terms into numbers (similar to when we coded instructions of register machines into numbers) we'll move on up to coding formulas into numbers (similar to coding configurations). We'll keep the same notation for the Gödel number of a formula, and the idea will be analogous:

Definition 3.1.5. Let ϕ be a propositional formula. We define the *Gödel number* of ϕ, $\#\phi$, is defined by induction as follows:

(1) If ϕ is atomic, then we define:

- $\#(t_1 \doteq t_2)$ to be $\mathsf{tuple}^3(\#t_1, \#t_2, 0)$

(2) Once $\#\phi_1$ and $\#\phi_2$ have been defined, we set:

- $\#(\phi_1 \wedge \phi_2) = \mathsf{tuple}^3(\#\phi_1, \#\phi_2, 1)$,
- $\#(\phi_1 \vee \phi_2) = \mathsf{tuple}^3(\#\phi_1, \#\phi_2, 2)$.
- $\#(\phi_1 \to \phi_2) = \mathsf{tuple}^3(\#\phi_1, \#\phi_2, 3)$
- $\#(\neg\phi_1) = \mathsf{tuple}^3(\#\phi_1, 0, 4)$,
- $\#((\forall x_n)\phi_1) = \mathsf{tuple}^3(\#\phi_1, n, 5)$,
- $\#((\exists x_n)\phi_1) = \mathsf{tuple}^3(\#\phi_1, n, 6)$,

Lemma 3.1.6. *The set* Fml *of Gödel codes for $\mathcal{L}_{Peano}$-formulas is primitive recursive.*

Proof. The proof is identical to the proof for terms, we only need to observe that 0 is a valid code for a formula (since $\#(\underline{0} \doteq \underline{0}) = 0$) and then recurse (using Lemma 3.0.1). In the case where $\mathsf{untuple}^3_3(x) = 0$ we use that Term is primitive recursive, of course. □

Again, as in terms, the coding we defined above is injective (i.e. ϕ and ψ are SYNTACTICALLY distinct $\mathcal{L}_{Peano}$-formulas, then $\#\phi \neq \#\psi$). Coding doesn't know anything about semantic equivalence!

Remark 3.1.7. A number can of course represent many objects! For example, the number 0 represents both the term $\underline{0}$ and the formula $\underline{0} = \underline{0}$. That's okay, all we ask is that in each of our (e.g. Term or Fml) numbers represent a unique object!

Moreover, most reasonable properties of formulas are also primitive recursive (as sets of codes of formulas). Here is a long list of useful fellas:

Lemma 3.1.8. *The following sets are all primitive recursive:*

(1) The set of pairs $(\#t, n)$ where t is a term and $v_n \notin \mathsf{Var}(t)$.

(2) The set of pairs $(\#t, n)$ where t is a term and $v_n \in \mathsf{Var}(t)$.

(3) The set of pairs $(\#\phi, n)$ where ϕ is a formula and $v_n \notin \mathsf{Var}(\phi)$.

(4) The set of pairs $(\#\phi, n)$ where ϕ is a formula and $v_n \in \mathsf{Var}(\phi)$.

(5) The set of pairs $(\#\phi, n)$ where ϕ is a formula and $v_n \notin \mathsf{Bound}(\phi)$.

(6) The set of pairs $(\#\phi, n)$ where ϕ is a formula and $v_n \in \mathsf{Bound}(\phi)$.

(7) The set of pairs $(\#\phi, n)$ where ϕ is a formula and $v_n \notin \mathsf{Free}(\phi)$.

(8) The set of pairs $(\#\phi, n)$ where ϕ is a formula and $v_n \in \mathsf{Free}(\phi)$.

Proof (Sketch). The argument is again very similar to the previous arguments. In each case we have to write out a long list of primitive recursive cases that define our characteristic functions. If you're feeling passionate about case-by-case analysis then try to do (1). □

In particular, the following set is primitive recursive:

$$\mathsf{Sen} := \{\#\phi : \phi \text{ is an } \mathcal{L}_{Peano}\text{-sentence}\}.$$

One last messy construction, that we really have to get out of the way (but hopefully at this point you just trust that it works):

Lemma 3.1.9. *For each $n \in \mathbb{N}$ there exist two primitive recursive functions $\mathsf{subst}_{trm}, \mathsf{subst}_{fla} : \mathbb{N}^{2n+1} \to \mathbb{N}$ such that if t and $s_1, \ldots, s_n$ are $\mathcal{L}_{Peano}$-terms, ϕ an $\mathcal{L}_{Peano}$ formula, and $i_1, \ldots, i_n \in \mathbb{N}$ we have that:*

- $\mathsf{subst}_{trm}(i_1, \ldots, i_n, \#s_1, \ldots, \#s_n, \#t) := \#(t[s_1, \ldots, s_n/x_{i_1}, \ldots, x_{i_n}])$.
- $\mathsf{subst}_{fla}(i_1, \ldots, i_n, \#s_1, \ldots, \#s_n, \#\phi) := \#(\phi[s_1, \ldots, s_n/x_{i_1}, \ldots, x_{i_n}])$.

Proof (Very sketch). All the hard work we did in the first-order logic chapter was to describe algorithmic ways of computing substitutions. It should be clear, hopefully, that in writing such an algorithm we only need to use primitive recursion (and no minimalisation) and by Church's thesis, this should be enough to finish the proof. □

3.2. Our axioms are primitive recursive. Now that we have an effective way of coding formulas, we will show that the set:

$$\{\#\phi : \phi \text{ is an instance of an axiom of } \vdash_{\mathcal{L}_{Peano}}\}$$

is primitive recursive.

I'll start with a little (unnecessary) diversion, which is to formally prove Corollary 1.5.9, our result about "decidability" of propositional logic, now that we have the words for it.

We'll first have to redo a little bit of work, but it's pretty much obvious leg work. A computer could do it, really.

3.2.1. *Decidability of propositional logic.* In our formalisation of propositional variable we kept around a countable set of propositional variables. Now, let us essentially redefine how to code propositional formulas:[1]

Definition 3.2.1. Let ϕ be a propositional formula. We define the *Gödel number* of ϕ, $\#\phi$, is defined by induction as follows:

(1) If ϕ is A_n, then $\#\phi$ is $\mathsf{tuple}^3(n, 0, 0)$.

(2) If ϕ is $\bot$ then $\#\phi$ is $\mathsf{tuple}^3(0, 0, 1)$.

(3) If ϕ is $\top$ then $\#\phi$ is $\mathsf{tuple}^3(0, 0, 2)$.

(4) If ϕ is $\psi \wedge \chi$ then $\#\phi$ is $\mathsf{tuple}^3(\#\phi, \#\psi, 3)$.

(5) If ϕ is $\psi \vee \chi$ then $\#\phi$ is $\mathsf{tuple}^3(\#\phi, \#\psi, 4)$

(6) If ϕ is $\psi \to \chi$ then $\#\phi$ is $\mathsf{tuple}^3(\#\phi, \#\psi, 4)$

Given Church's thesis and the effectiveness of rewriting formulas in disjunctive normal form and then replacing instances of $\vee$ and $\wedge$ with $\to$ and $\bot$, the following remark is immediate:

Remark 3.2.2.

(1) There is a primitive recursive function $f : \mathbb{N} \to \mathbb{N}$ which given the Gödel number of an arbitrary propositional formula ϕ returns the Gödel number of a formula ψ which is logically equivalent to ϕ and is only written using connectives from $\to$ and $\bot$.

(2) Moreover, the set Prop of all Gödel numbers of propositional formulas is primitive recursive.

[1]This is essentially the same as definition Definition 3.1.5, so you can just skim through it.

I'll reserve the definition of decidability proper for first-order theories, but the following result essentially tells us what we intuitively understood from the very beginning. There is an algorithm (i.e. a register machine program) which given any finite set of propositional formulas Γ and any propositional formula ϕ can compute if $\Gamma \vdash \phi$. The proof below implicitly uses the soundness and completeness theorem.

THEOREM 3.2.3. *Let Γ be a finite set of propositional formulas, then the set:*

$$\{\#\phi : \Gamma \vdash \phi\}$$

is primitive recursive.

PROOF. Let me give a somewhat more formal proof than the usual appeal to Church's thesis (which was essentially the "fake" proof I gave at the end of Chapter 2, but also at this point, such a proof should be enough). The trick is that we can code assignments of truth values into nice primitive recursive functions. Here's a funky way:

For all $k \in \mathbb{N}$, we define an assignment $\mathcal{A}_k : \mathsf{Var} \to \{T, F\}$ as follows:

$$\lambda_k(A_n) := \begin{cases} T & \text{if } \mathsf{pr}(n) | k \\ F & \text{otherwise.} \end{cases}$$

Let $\mathcal{A} : \mathsf{Var} \to \{T, F\}$ be any assignment. For all $N \in \mathbb{N}$ there is some $k \in \mathbb{N}$ such that for all $n \leq N$ we have that $\mathcal{A}_k(A_n) = \mathcal{A}(A_n)$. Indeed, we may well just take

$$k = \prod_{0 \leq n \leq N} \mathsf{pr}(n)^{\mathcal{A}(A_n)}.$$[2]

Crucially, k can be chosen to be at most $\mathsf{pr}(N)$!

For a propositional formula ϕ we obviously have that if $A_n \in \mathsf{Var}(\phi)$, then $n \leq \#\phi$. Thus:

$$\Gamma \vdash \phi \text{ is a tautology if and only if for all } k \leq \left(\mathsf{pr}(\#\phi) \textstyle\prod_{\psi \in \Gamma} \mathsf{pr}(\#\psi)\right)! \text{ we have that}$$
$$\text{if } \mathcal{A}_k(\Gamma) = T, \text{ then } \mathcal{A}(\phi) = T.$$

Now we define a primitive recursive function:

$$\mathsf{E}(k, x) := \begin{cases} 0 & \text{if } x \notin \#\mathsf{Prop} \\ 1 & \text{if } x = \#A \text{ and } \mathcal{A}_k(A) = T. \\ 0 & \text{otherwise.} \end{cases}$$

[2]This is shorthand for $\mathsf{pr}(n)^1$ if $\mathcal{A}(A_n) = T$ and $\mathsf{pr}(n)^0$ otherwise.

To show that such an E exists we need Lemma 3.0.1. Using this lemma and the fact that primitive recursion is closed under definition by cases, we can define $\mathsf{E}(k,x)$ as follows:

- If $x \notin \mathsf{Prop}$ then $\mathsf{E}(k,x) = 0$.
- If $x \in \mathsf{Prop}$ then:
  - If $\mathsf{untuple}^3(x) = 1$, then $\mathsf{E}(x,k) = 0$.
  - If $\mathsf{untuple}^3(x) = 4$, then $\mathsf{E}(k,x) = \begin{cases} 0 & \text{if } \mathsf{E}(k,\mathsf{untuple}_1(x)) = 0 \text{ and } \mathsf{E}(k,\mathsf{untuple}_2(x)) = \\ 1 & \text{otherwise .} \end{cases}$

This makes sense, us $\mathsf{untuple}_i(x) < x$ always. Finally, we have that:

$$x \in \{\#\phi : \Gamma \vdash \phi\} \text{ iff } (\forall k) \leq M \mathsf{E}(k,x) \prod_{\psi \in \Gamma} \mathsf{E}(k,\psi) = 1,$$

where $M = \left(\mathsf{pr}(\#\phi) \prod_{\psi\in\Gamma} \mathsf{pr}(\#\psi)\right)!$, □

Of course, we can just forget Γ in the theorem above to obtain:

Corollary 3.2.4. *The set $\{\#\phi :\vdash \phi\}$ of all (Gödel codes of) propositional tautologies is primitive recursive.*

Now, I'm not sure that this was really informative, but it sure was formal.

End of digression

BACK TO AXIOMS NOW.

Proposition 3.2.5. *The set*

$$\{\#\phi : \phi \textit{ is an } \mathcal{L}\textit{-formula which is an instance } (\text{A1}),(\text{A2}),(\text{A3})\}$$

is primitive recursive.

PROOF. Let's do this for (A1):

$$(\phi \to (\psi \to \phi))$$

It is clear that for any $\mathcal{L}_{Peano}$-formulas ϕ we have that

$$\#((\phi \to (\psi \to \phi))) = \mathsf{tuple}^3(\#\phi, \mathsf{tuple}^3(\#\psi, \#\phi, 3), 3).$$

Hence, x is an instance of (A1) if and only if there are $y, z \leq x$ such that $y, z \in \mathsf{Form}$ and

$$x = \mathsf{tuple}^3(y, \mathsf{tuple}^3(z, y, 3), 3).$$

This is of course primitive recursive. The argument for instances of the other two axioms is pretty much the same. □

Similarly, the quantifier axioms are primitive recursive:

Proposition 3.2.6. *The set*

$$\{\#\phi : \phi \text{ is an } \mathcal{L}\text{-formula which is an instance (Q1),(Q2),(Q3),(Q4)}\}$$

is primitive recursive.

PROOF. The proof is essentially identical. The only technicality that I need to point out is that this proof hinges on the fact that subst_{fla} is primitive recursive (for axioms (Q2) and (Q3)). □

Proposition 3.2.7. *The set:*

$$\{\#\phi : \phi \text{ is an } \mathcal{L}\text{-formula which is an instance (E1),(E2),(E3),(E4), (E5)}\}$$

is primitive recursive.

PROOF. Similar to previous proofs. □

Exercise 3.2.8. Write out the details of the proofs above.

To fix notation, we will write Ax for the set of all Gödel numbers of instances of axioms. This is the union of the three sets we just showed are primitive recursive and is therefore primitive recursive. Let's take a very brief break to fix some terminology.

3.3. Recursive and decidable theories. We have only defined coding for $\mathcal{L}_{Peano}$-formulas here, but a similar definition could let us code formulas into numbers in any (finite) language $\mathcal{L}$. I won't do this here explicitly, but the idea is similar. We just assign numbers to the symbols and take it from there!

Definition 3.3.1. We say that a theory T is *recursive* if the set $\{\#\phi : \phi \in T\}$ is recursive.

Most natural theories are recursive, Indeed, the empty theory is recursive, and so is any finite theory. In particular T_{PA_0} is recursive.

Exercise 3.3.2. Prove that T_{PA} is recursive.

Definition 3.3.3. A theory T is *decidable* if the set:

$$\{\#\phi : \phi \in \mathsf{Sen}(\mathcal{L}) \text{ and } T \vdash \phi\}$$

is recursive. Otherwise, we say that T is *undecidable*.

Okay, now back to coding.

3.4. Coding proofs. This will be easy enough. We have just one new definition. Let $\Delta = (\delta_1, \ldots, \delta_n)$ be a sequence of formulas. Then:

$$\#\#\Delta := \langle \#\delta_1, \ldots, \#\delta_n \rangle.$$

As usual, we'll call $\#\#\Delta$ the **Gödel number** of Δ.

THEOREM 3.4.1. *Let T be a recursive theory. Then, the set:*

$$\mathsf{Drv}(T) := \{(\#\phi, \#\#\Delta) : \phi \textit{ is a formula, and } \Delta \textit{ a derivation of } \phi \textit{ in } T\}$$

is primitive recursive.

PROOF. It really suffices to go back to our definition of $\vdash$ and see that the procedure for recognising whether a sequence of formulas is a derivation is an effective one! Indeed, $(n, m) \in \mathsf{Drv}(T)$ if and only if the following three conditions are satisfied:

(1) For all $i < \mathsf{lg}(m)$, $(m)_i \in \mathsf{Fml}$

(2) We have that $(m)_{\mathsf{lg}(m)-1} = n$

(3) For all $i \leq \mathsf{lg}(m)$ one of the following holds:

- Either $(m)_i \in \mathsf{Ax} \cup \{\#\phi : \phi \in T\}$, or
- There is some $j < i$ and $p < m$ such that $(m)_i = \mathsf{tuple}^3((m)_j, p, 5)$,
- There are $k, l < i$ such that $(m)_k = \mathsf{tuple}^3((m)_j, (m)_i, 3)$.

The three conditions in (3) express that any formula in the derivation is either an instance of an axiom, or it is obtained by (Gen) [and the variable that appears in the quantifier cannot have index greater than m] or it is obtained by (MP). You should probably spend a little bit of time thinking about this! □

We may pretty much conclude the following:

Corollary 3.4.2. *Let T be a recursive theory. Then*

$$\{\#\phi : \phi \in \mathsf{Sen}(\mathcal{L}) \text{ and } T \vdash \phi\}$$

is recursively enumerable.

PROOF. Observe that $\#\phi \in \{\#\phi : \phi \in \mathsf{Sen}(\mathcal{L}) \text{ and } T \vdash \phi\}$ if and only if $\#\phi \in \mathsf{Sen}(\mathcal{L})$ (i.e. it is a sentence) and there is some $m \in \mathbb{N}$ such that $(\#\phi, m) \in \mathsf{Drv}(T)$. The former set is, as we have essentially seen primitive recursive, and the latter is the projection of a primitive recursive set onto the last coordinate (so by Theorem 3.2.7 it is recursively enumerable). Thus $\#\phi \in \{\#\phi : \phi \in \mathsf{Sen}(\mathcal{L}) \text{ and } T \vdash \phi\}$ if and only if it is in the intersection of two recursively enumerable sets. □

In particular, the sets

$$\{\#\phi : \phi \in \mathsf{Sen}(\mathcal{L}_{Peano}) \text{ and } T_{PA} \vdash \phi\}$$

and

$$\{\#\phi : \phi \in \mathsf{Sen}(\mathcal{L}_{Peano}) \text{ and } T_{PA_0} \vdash \phi\}$$

are both recursively enumerable. The cherry on top is the following theorem:

THEOREM 3.4.3. *If T is a recursive and complete theory, then T is decidable.*

PROOF. Because T is complete, the complement of $\{\#\phi : T \vdash \phi\}$ is precisely $\{\#\phi : T \vdash \neg\phi\}$, which by the above is recursively enumerable. □