LECTURE 5: ARITHMETIZATION OF SYNTAX

In the previous lectures we have introduced:

- A notion of computability for number-theoretic functions, namely the *recursive* (and *primitive recursive*) functions.
- The *syntactical*[1] notions of a first-order language: symbols, expressions, terms, formulas.
- The *semantic*[2] notions associated with *interpreting* the language: models (structures), evaluation of terms when variables are assigened values, and the notion of *satisfaction* of a formula, and truth of a sentence, and logical implication.

In this development it has been clear how the semantic notions rest upon the syntactical notions, but so far, it is not clear how the computability-theoretic notions connect to the rest. The goal of this lecture is to now connect the syntactical notions to computability.

Warning: In this lecture, we will be using the normal round parentheses, (and), to enclose tuples, and not the angled parenthesis. The angled parentheses, $\langle$ and $\rangle$, will now be be used for something else, namely it will be used for a *number* which codes a tuple of elements of ω. Though this may seem confusing at a first glance, you will get used to this fairly quickly; and only in part (*ii*) of the warning on p. 4 does a conflict of notation arise, albeit briefly.

Acknowledgment: I gratefully acknowledge that these notes owe their existence to the efforts of Jonas Jensen, a student who took the course in 2013. They are based on a set of notes TEX'ed up by Jonas Jensen, who selflessly took the time to convert my original handwritten notes for the benefit of all future students of the course.

1. Coding and decoding sequences of numbers using recursive functions

Throughout, we shall take for granted the fact that there is a recursive (in fact primitive recursive) function, $\omega \to \omega : i \mapsto p_i$, where p_i is the $(i+1)$st prime. (So $p_0 = 2$, $p_1 = 3$, $p_3 = 5$ etc.) Eventually you will be asked to prove this fact in the exercises. We also remind the reader of the relation Div, defined by

$$\mathrm{Div} = \{(m, n) \in \omega^2 : m \text{ divides } n\},$$

and that this is a primitive recursive relation.

Definition 1.1. For $(a_0, \ldots, a_{m-1}) \in \omega^m$, let

$$\langle a_0, \ldots, a_{m-1}\rangle = p_0^{a_0+1} \cdots p_{m-1}^{a_{m-1}+1} = \prod_{i<m} p_i^{a_i+1} \in \omega.$$

So $\langle a_0, \ldots, a_{m-1}\rangle$ is a *single* number[3] in ω. By the uniqueness of prime factorization, we can recover ("decode") the tuple $(a_0, \ldots, a_{m-1})$ from $\langle a_0, \ldots, a_{m-1}\rangle$. In other words,

$$\omega^m \to \omega : (a_0, \ldots, a_{m-1}) \mapsto \langle a_0, \ldots, a_{m-1}\rangle$$

[1]Syntactical: Pertaining to the rules of arranging symbols correctly, according to grammatical laws.

[2]Semantic: Pertaining to the meaning of symbols.

[3]The number $\langle a_0, \ldots, a_{m-1}\rangle$ is sometimes called the *code* for the sequence $(a_0, \ldots, a_{m-1})$

is an injection.

The next lemma shows that this coding is "nice" in the sense that recursive (and primitive recursive) functions can decode and perform elementary sequence operations on the codes for sequences.

Lemma 1.2. (1) *For each $m \in \omega$, the map*

$$\omega^m \to \omega : (a_0, \dots, a_{m-1}) \mapsto \langle a_0, \dots, a_{m-1} \rangle$$

is primitive recursive.

(2) *There is a recursive function $\omega^2 \to \omega : (a, b) \mapsto (a)_b$, such that if $a = \langle a_0, \dots, a_{m-1} \rangle$ for some $m \in \omega$ and $b < m$, then $(a)_b = a_b$.*

(3) *The set of "sequence numbers",*

$$\mathrm{Seq} = \{n \in \omega : (\exists m)(\exists a_0, \dots, a_{m-1} \in \omega) n = \langle a_0, \dots, a_{m-1} \rangle\}$$

is primitive recursive

(4) *There is a recursive function $\ell\mathrm{h} : \omega \to \omega$ such that $\ell\mathrm{h}(\langle a_0, \dots, a_{m-1} \rangle) = m$.*

(5) *There is a recursive function $\omega^2 \to \omega : (a, b) \mapsto a \upharpoonright b$, such that if $a = \langle a_0, \dots, a_{m-1} \rangle$ for some $m \in \omega$ and some $a_0, \dots, a_{m-1} \in \omega$, then $a \upharpoonright b = \langle a_0, \dots, a_{b-1} \rangle$ for any $b \leqslant m$.*

Remark 1.3. In (2), (4) and (5) above, we are describing recursive functions that have required properties on a subset of their domain. For instance, in (2), you may rightfully wonder what happens when $b \geqslant m$ and we try to compute $(a)_b$. The truth is that we don't care, as long as the recursive function does the right thing on the part of the domain we care about (which in (2) means for a of the form $a = \langle a_0, \dots, a_{m-1} \rangle$ and $b \leqslant m$). In computer programming this principle is often called "garbage in, garbage out": If your input is not of the form required by the prorgam, the programmer takes no responsibility for what nonsense output the computer may produce.

Proof. (1) this follows from that multiplication and exponentiation are primitive recursive.

(2) Let

$$R = \{(a, b, n) : a \neq 0 \wedge p_b^{n+1} \mid a\} = \{(a, b, n) \in \omega^3 : a \neq 0 \wedge (p_b^{n+1}, a) \in \mathrm{Div}\},$$

where Div is as defined above (and in the recursion theory notes). It is not hard to see that R is primitive recursive (do it!). (2) now follows since

$$(a)_b = \mu n[(a, b, n) \notin R] \dot{-} 1.$$

(3) Let now $R = \{(a, i) \in \omega^2 : a > 0 \wedge (\mathrm{Div}(p_{i+1}, a) \to \mathrm{Div}(p_i, a))\}$. Then this is primitive recursive (check this!), and consider

$$R^{\forall^<} = \{(a, n) \in \omega^2 : (\forall i < n) R(a, i)\}.$$

This is primitive recursive, and it is easy to see that

$$\mathrm{Seq} = \{a \in \omega : (a, a+1) \in R^{\forall^<}\}.$$

(4) $\ell\mathrm{h}(a) = \mu n[(p_n, a) \notin \mathrm{Div}]$.

(5) We have

$$a \upharpoonright b = \mu n[a = 0 \vee (n \neq 0 \wedge (\forall j < b)(\forall k < a) \mathrm{Div}(p_j^k, a) \to \mathrm{Div}(p_j^k, n))],$$

from which it is easy to check that $(a, b) \mapsto a \upharpoonright b$ is recursive. ☐

Exercise 1. Give the details of the proof of (1) in the previous lemma.

Exercise 2. Fill in the details in the proofs of (4) and (5) in the previous lemma. What this amounts to two things: First proving *why* the definitions given above actually work to produce the desired functions, and the proving that the definition, after suitable unpacking, define recursive functions.

Remark 1.4. In the lemma above, with some additional work we could show that all these functions are in fact primitive recursive, but we don't need this strengthening.

Our last sequence operation, which we need to check can be done "recursively in the codes", is concatenation, that is, the act of putting two sequences together by sticking the second one on the end of the first. That is taken care of by (1) in the next Lemma, while (2) allows concatenation of possibly *more* than two sequences. In fact, (2) expresses the fact that concatenation of $a \in \omega$ sequences can be done in a recursive way *uniformly* in a.

Lemma 1.5 (Concatenation operations)**.**
(1) *There is a recursive function* $\omega^2 \to \omega : (a, b) \mapsto a*b$, *such that* $\langle a_0, \dots, a_{m-1}\rangle * \langle b_0, \dots, b_{k-1}\rangle = \langle a_0, \dots, a_{m-1}, b_0, \dots, b_{k-1}\rangle$ *for any* $m, k \in \omega$ *and* $(a_0, \dots, a_{m-1}) \in \omega^m$ *and* $(b_0, \dots, b_{k-1}) \in \omega^k$.
(2) *More generally, if* $f : \omega^{n+1} \to \omega$ *is a recursive function, then there is a recursive function* $F : \omega^{n+1} \to \omega$, *such that whenever* $a, b_1, \dots, b_n \in \omega$ *and for all* $i < a$ *we have* $f(i, b_1, \dots, b_n) \in Seq$, *then*

$$F(a, b_1, \dots, b_n) = \circledast_{i<a} f(i, b_1, \dots, b_n) \overset{\text{def!}}{=} f(0, b_1, \dots, b_n) * \cdots * f(a-1, b_1, \dots, b_n)$$

Proof. We will return to this in the future, or you can find a proof on p. 223 in our textbook. □

2. Gödel numbering

We will now see that our syntactical notions can be "coded" by natural numbers, and "decoded" and described by (primitive) recursive functions and relations.

Definition 2.1. (1) We assign (once and for all) to each logical symbol (and =) a number as follows:

Symbol	$\forall$	(	)	$\neg$	$\to$	$=$	v_i
Number	0	1	3	5	7	9	$9+2i$

(2) Let $\mathcal{L}$ be a countable language, and let $h : \mathcal{L} \to \omega$ be an injection from the non-logical symbols of $\mathcal{L}$ to the positive even numbers. We say that h is a *recursive numbering*[4] of $\mathcal{L}$ if the sets

$$F_{\mathcal{L}} = \{\langle k, m\rangle \in \omega : k \text{ is the value of } h \text{ at some } m\text{-place function symbol of } \mathcal{L}\}$$

and

$$R_{\mathcal{L}} = \{\langle k, m\rangle \in \omega : k \text{ is the value of } h \text{ at some } m\text{-place relation symbol of } \mathcal{L}\},$$

are recursive.

Remark 2.2. In (2) above we identify constant symbols with 0-place function symbols. If $\mathcal{L}$ is finite (i.e., has finitely many non-logical symbols) then $F_{\mathcal{L}}$ and $R_{\mathcal{L}}$ are finite, and so any numbering h is a recursive numbering.

[4] Also called a recursive presentation.

Example 2.3. For $\mathcal{L}_{\text{PA}} = \{\hat{0}, \hat{\mathcal{S}}, \hat{+}, \hat{\cdot}, \hat{<}\}$, we make (once and for all) the following numbering:

Symbol	$\hat{0}$	$\hat{\mathcal{S}}$	$\hat{<}$	$\hat{+}$	$\hat{\cdot}$
Number	2	4	6	8	10

since $\mathcal{L}_{\text{PA}}$ is finite, this numbering is recursive.

Definition 2.4. Let $\mathcal{L}$ be a recursively numbered language, numbered by some $h : \mathcal{L} \to \omega$. Let $\varepsilon = s_0 \cdots s_{n-1}$ be an expression in $\mathcal{L}$ (i.e. a finite string of symbols in $\mathcal{L}$). The *Gödel number* of ε is

$$\#\varepsilon \overset{\text{def}}{=} \langle h(s_0), \dots, h(s_{n-1}) \rangle.$$

Warnings: (i) The Gödel number $\#\varepsilon$ depends on h, unless no non-logical symbol occur in ε. (ii) When we write $\#s$, where $s \in \mathcal{L}$ is some symbol, we (of course) really mean $\#\langle s \rangle$. (Here $\langle s \rangle$ means the one-element sequence consisting of s, that is, the angled parenthesis are being used in their *previous* meaning!) (iii) For any symbol $s \in \mathcal{L}$, $\#s \neq h(s)$ (think about this!).

Example 2.5. Let $\mathcal{L}_{\text{PA}}$ be numbered as before, and consider the expression[5] $(\forall v_6)v_6 = \hat{0}$. Then $\#(\forall v_6)v_6 = \hat{0}$ is

$$\langle 1, 0, 21, 3, 21, 9, 2 \rangle = 2^{1+1}3^{0+1}5^{21+1}7^{3+1}11^{21+1}13^{9+1}17^{2+1} = 2^2 3^1 5^{22} 7^4 11^{22} 13^{10} 17^3$$

The Gödel number $\#(\forall v_6)v_6 = \hat{0}$ is larger than 10^{47}, which is more than the number of water molecules in all the oceans on Earth.

Definition 2.6. Let $\mathcal{L}$ be a recursive presented language, and let Φ be a set of expressions in $\mathcal{L}$. We define

$$\#\Phi = \{\#\varepsilon : \varepsilon \in \Phi\},$$

and say that the set of Gödel numbers for elements of Φ is recursive (respectively primitive recursive) if $\#\Phi$ is recursive (respectively primitive recursive) as a subset of ω.

Lemma 2.7. *The set of Gödel numbers of expression that consist of a* single *variable symbol is primitive recursive. In other words, $\#\{v_i : i \geqslant 1\}$ is primitive recursive.*

Proof. We have

$$\{\#v_i : i \geqslant 1\} = \{a \in \omega : (\exists b \leqslant a) a = 2^{11+2b}\}.$$

To see that this is primitive recursive, note first that

$$R = \{(b, a) \in \omega^2 : a = 2^{11+2b}\}$$

is primitive recursive since it is the graph of the primitive recursive function $b \mapsto 2^{11+2b}$, which is itself a composition of primitive recursive functions.

By bounded quantification

$$R^{\exists <} = \{(c, a) \in \omega^2 : (\exists b < c)(b, a) \in R\}$$

is primitive recursive, and now note that $\{\#v_i : i \geqslant 1\} = \{a : (a+1, a) \in R^{\exists <}\}$. □

[5]I am well aware that $(\forall v_6)v_6 = \hat{0}$ for at least two reasons is not a wff according to our definitions, but that doesn't matter for this example.

What the next theorem means to us is that deciding if a number codes a term or a formula can be done by a recursive function.

Theorem 2.8. *Let $\mathcal{L}$ be a recursively numbered language. Then:*

(1) The set of Gödel numbers for terms in $\mathcal{L}$ is recursive, i.e. $\#\,\mathrm{Term}(\mathcal{L})$ *is recursive.*
(2) The set of Gödel numbers for formulas in $\mathcal{L}$ is recursive, i.e. $\#\,\mathrm{Formula}(\mathcal{L})$ *is recursive.*

We need a little more preparation before we can prove this, so we will get back to this in the future.

Asger Törnquist