Mathematical Logic

Lecture 8

Harbin, 2023

Let's take a closer look at rule (R3) from the previous lecture. One of the conditions was: for all $a \in \mathbb{N}^n$ there exists $y \in \mathbb{N}$ such that $G(a, y) = 0$. This condition is not constructive: it could be satisfied for a certain G without us ever knowing it.

We shall now argue informally that it is impossible to generate in a fully constructive way exactly the computable functions. Such a constructive generation process would presumably enable us to enumerate effectively a sequence of algorithms $\alpha_0, \alpha_1, \alpha_2, \ldots$ such that each α_n computes a (computable) function $f_n : \mathbb{N} \to \mathbb{N}$, and such that every computable function $f : \mathbb{N} \to \mathbb{N}$ occurs in the sequence $f_0, f_1, f_2, \ldots$, possibly more than once. Now consider the function $f_{diag} : \mathbb{N} \to \mathbb{N}$ defined by

$$f_{diag}(n) = f_n(n) + 1.$$

Then f_{diag} is clearly computable in the intuitive sense, but $f_{diag} \neq f_n$ for all n, in violation of the Church-Turing Thesis.

This way of producing a new function f_{diag} from a sequence (f_n) is called diagonalization. Here is a class of computable functions that can be generated constructively:
The primitive recursive functions are the functions $f : \mathbb{N}^n \to \mathbb{N}$ obtained inductively as follows:

(PR1) The nullary function $\mathbb{N}^0 \to \mathbb{N}$ with value 0, the unary successor function S, and all coordinate functions I_i^n are primitive recursive.

(PR2) If $G : \mathbb{N}^m \to \mathbb{N}$ is primitive recursive and $H_1, \dots, H_m : \mathbb{N}^n \to \mathbb{N}$ are primitive recursive, then $G(H_1, \dots, H_m)$ is primitive recursive.

(PR3) If $F : \mathbb{N}^{n+1} \to \mathbb{N}$ is obtained by primitive recursion from primitive recursive functions $G : \mathbb{N}^n \to \mathbb{N}$ and $H : \mathbb{N}^{n+2} \to \mathbb{N}$, then F is primitive recursive :

$$F(x_1, \dots, x_{n-1}, 0) = G(x_1, \dots, x_{n-1})$$

$$F(x_1, \dots, x_{n-1}, m+1) = H(x_1, \dots, x_{n-1}, m, F(x_1, \dots, x_{n-1}, m))$$

Example. $f(x,y) = x + y$. Very formally we would say

1. $g(x) = x$ is primitive recursive by the projection rule.
2. $I_3(x,y,z) = z$ is primitive recursive by the projection rule.
3. $S(x) = x + 1$ is primitive recursive by the successor rule.
4. $h(x,y,z) = S(I_3(x,y,z)) = z + 1$ is primitive recursive by the composition rule and composing S and I_3.
5. Define f using the recursion rule and g, h:

$$f(x,0) = g(x) = x$$

$$f(x,n+1) = h(x,n,f(x,n)) = S(I_3(x,n,f(x,n))) = f(x,n) + 1$$

<u>Example.</u> $f(x, y) = xy$ is primitive recursive via

$$f(x, 0) = 0$$

$$f(x, y + 1) = f(x, y) + x$$

Now that we have multiplication is primitive recursive, we use it to define powers.

<u>Example.</u> $f(x, y) = x^y$ is primitive recursive via

$$f(x, 0) = 1$$

$$f(x, y + 1) = xf(x, y)$$

<u>Example.</u> In all the above examples we took a well known function and showed it was primitive recursive. Here we define a function directly. What this function does is, if $x > 0$ then it subtracts 1, otherwise it just returns 0.

$$f(0) = 0$$

$$f(x+1) = x$$

To do this formally, recall that in the recursion rule the function h can depend on x and $f(x)$. Henceforth, call this function $M(x)$. We now use M to define a version of subtraction.

<u>Example.</u> $x \boxminus y$:

$$f(x,0) = x$$

$$f(x, y+1) = M(f(x,y))$$

A relation $R \subseteq \mathbb{N}^n$ is said to be primitive recursive if its characteristic function χ_R is primitive recursive. As the next two lemmas show, the computable functions that one ordinarily meets with are primitive recursive. Let $x \in \mathbb{N}^m$ (with m depending on the context), and $y \in \mathbb{N}$.

Lemma 1

The following functions and relations are primitive recursive:
(i) each constant function c^n_m
(ii) the binary operations $+, \cdot$, and $(x, y) \mapsto x^y$ on $\mathbb{N}$
(iii) the predecessor function $Pd : \mathbb{N} \to \mathbb{N}$ given by $Pd(x) = x \boxminus 1$, the unary relation $\{x \in \mathbb{N} : x > 0\}$, the function $\boxminus : \mathbb{N}^2 \to \mathbb{N}$
(iv) The binary relations $\geq, \leq$ and $=$ on $\mathbb{N}$.

Proof. The function c^0_m is obtained from c^0_0 by applying (PR2) m times with $G = S$. Next, c^n_m is obtained by applying (PR2) with $G = c^0_m$ (with $k = 0$ and $t = n$). The functions in (ii) are obtained by the usual primitive recursions. It is also easy to write down primitive recursions for the functions in (iii), in the order they are listed. For (iv), note that $\chi_{\geq}(x, y+1) = \chi_{>0}(x) \cdot \chi_{\geq}(Pd(x), y)$ ■

Remark. Almost all Lemmas and Propositions in the previous lecture go through with computable replaced by primitive recursive. In particular, the function β is primitive recursive. Exercise: formulate these statements.

The Ackermann Function

By diagonalization we can produce a computable function that is not primitive recursive, but the so-called Ackermann function does more, and plays a role in several contexts. First we define inductively a sequence $A_0, A_1, A_2, \ldots$ of primitive recursive functions $A_n : \mathbb{N} \to \mathbb{N}$:

$$A_0 = y + 1, \qquad A_{n+1}(0) = A_n(1)$$

$$A_{n+1}(y+1) = A_n(A_{n+1}(y)).$$

Thus $A_0 = S$, and $A_{n+1} \circ A_0 = A_n \circ A_{n+1}$. One verifies easily that $A_1(y) = y + 2$ and $A_2(y) = 2y + 3$ for all y. We define the Ackermann function $A : \mathbb{N}^2 \to \mathbb{N}$ by $A(n, y) := A_n(y)$.

Lemma 2

The function A is computable, and strictly increasing in each variable. Also, for all n and x, y:
(i) $A_n(x+y) \geq A_n(x) + y$
(ii) $n \geq 1 \Rightarrow A_{n+1}(y) > A_n(y) + y$
(iii) $A_{n+1}(y) \geq A_n(y+1)$
(iv) $2A_n(y) < A_{n+2}(y)$
(v) $x < y \;\Rightarrow\; A_n(x+y) \leq A_{n+2}(y)$

Proof. Assume inductively that $A_0, A_1, \ldots, A_n$ are strictly increasing and $A_0(y) < A_1(y) < \ldots < A_n(y)$ for all y. Then

$$A_{n+1}(y+1) = A_n(A_{n+1}(y)) \geq A_0(A_{n+1}(y)) > A_{n+1}(y)$$

so A_{n+1} is strictly increasing. Next we show that $A_{n+1}(y) > A_n(y)$ for any y : $A_{n+1}(0) = A_n(1)$, so $A_{n+1}(0) > A_n(0)$ and $A_{n+1}(0) > 1$, so $A_{n+1}(y) > y+1$ for all y. Hence $A_{n+1}(y+1) = A_n(A_{n+1}(y)) > A_n(y+1)$.

Inequality (i) follows easily by induction on n, and a second induction on y. For inequality (ii), we proceed again by induction on (n, y): using $A_1(y) = y + 2$ and $A_2(y) = 2y + 3$, we obtain $A_2(y) > A_1(y) + y$. Let $n > 1$, and assume inductively that $A_n(y) > A_{n-1}(y) + y$. Then $A_{n+1}(0) = A_n(1) > A_n(0) + 0$, and

$$A_{n+1}(y+1) = A_n(A_{n+1}(y)) \geq A_n(y+1+A_n(y)) \geq$$

$$\geq A_n(y+1) + A_n(y) > A_n(y+1) + y + 1.$$

In (iii) we proceed by induction on y. We have equality for $y = 0$. Assuming inductively that (iii) holds for a certain y we obtain

$$A_{n+1}(y+1) = A_n(A_{n+1}(y)) \geq A_n(A_n(y+1)) \geq A_n(y+2)$$

Note that (iv) holds for $n = 0$. For $n > 0$ we have by (i), (ii) and (iii):

$$2A_n(y) \leq A_n(y + A_n(y)) < A_n(A_{n+1}(y)) = A_{n+1}(y+1) \leq A_{n+2}(y).$$

Note that (v) holds for $n = 0$. Assume (v) holds for a certain n. Let $x < y + 1$. We can assume inductively that if $x < y$, then $A_{n+1}(x + y) \leq A_{n+3}(y)$, and we want to show that:

$$A_{n+1}(x + y + 1) \leq A_{n+3}(y + 1)$$

Case 1. $x = y$. Then

$$A_{n+1}(x + y + 1) = A_{n+1}(2x + 1) = A_n(A_{n+1}(2x)) \leq$$

$$\leq A_{n+2}(2x) < A_{n+2}(A_{n+3}(x)) = A_{n+3}(y + 1).$$

Case 2. $x < y$. Then

$$A_{n+1}(x + y + 1) = A_n(A_{n+1}(x + y)) \leq A_{n+2}(A_{n+3}(y)) = A_{n+3}(y + 1).\blacksquare$$

Below we put $|x| := x_1 + \ldots + x_m$ for $x = (x_1, \ldots, x_m) \in \mathbb{N}^m$

Proposition 3

Given any primitive recursive function $F : \mathbb{N}^m \to \mathbb{N}$ there is an $n = n(F)$ such that $F(x) \leq A_n(|x|)$ for all $x \in \mathbb{N}^m$.

Proof. Call an $n = n(F)$ with the property above a bound for F. The nullary constant function with value 0, the successor function S, and each coordinate function I_i^m $(1 \leq i \leq m)$, has bound 0. Next, assume $F = G(H_1, \ldots, H_k)$ where $G : \mathbb{N}^k \to \mathbb{N}$ and $H_1, \ldots, H_k : \mathbb{N}^m \to \mathbb{N}$ are primitive recursive, and assume inductively that $n(G)$ and $n(H_1), \ldots, n(H_k)$ are bounds for G and $H_1, \ldots, H_k$. By part (iv) of the previous lemma we can take $N \in \mathbb{N}$ such that $n(G) \leq N$, and $\sum_i H_i(x) \leq A_{N+1}(|x|)$ for all x. Then

$$F(x) = G(H_1(x), \ldots, H_k(x)) \leq A_N(\sum_i H_i(x)) \leq A_N(A_{N+1}(|x|)) \leq A_{N+2}(|x|$$

Finally, assume that $F : \mathbb{N}^{m+1} \to \mathbb{N}$ is obtained by primitive recursion from the primitive recursive functions $G : \mathbb{N}^m \to \mathbb{N}$ and $H : \mathbb{N}^{m+2} \to \mathbb{N}$, and assume inductively that $n(G)$ and $n(H)$ are bounds for G and H. Take $N \in \mathbb{N}$ such that $n(G) \leq N + 3$ and $n(H) \leq N$. We claim that $N + 3$ is a bound for F : $F(x, 0) = G(x) \leq A_{N+3}(|x|)$, and by part (v) of the lemma above,

$$F(x, y+1) = H(x, y, F(x, y)) \leq A_N(|x| + y + A_{N+3}(|x| + y)) \leq$$

$$\leq A_{N+2}(A_{N+3}(|x| + y)) = A_{N+3}(|x| + y + 1) \quad \blacksquare$$

Consider the function $A^* : \mathbb{N} \to \mathbb{N}$ defined by $A^*(n) = A(n, n)$. Then A^* is computable, and for any primitive recursive function $F : \mathbb{N} \to \mathbb{N}$ we have $F(y) < A^*(y)$ for all $y > n(F)$, where $n(F)$ is a bound for F. In particular, A^* is not primitive recursive. Hence A is computable but not primitive recursive.

The recursion in "primitive recursion" involves only one variable; the other variables just act as parameters. The Ackermann function is defined by a recursion involving both variables:

$$A(0, y) = y + 1, \;\; A(x + 1, 0) = A(x, 1) \; A(x + 1, y + 1) = A(x, A(x + 1, y))$$

This kind of double recursion is therefore more powerful in some ways than what can be done in terms of primitive recursion and composition.

Recall that the characteristic function of a predicate $P(x)$ is the function whose value is 1 if $P(x)$ and 0 otherwise.

The representing function of P is similar, but has value 0 if $P(x)$ and 1 otherwise. A predicate is primitive recursive if and only if its representing function is primitive recursive (by definition!).

Remark. As we have seen, the primitive recursive predicates are closed under the logical connectives and under bounded quantification.

Theorem 4.

Every bounded arithmetic formula (i.e., formula with only bounded quantifiers) defines a primitive recursive predicate.

Remark. The converse, due to Gödel, is also true, but is more difficult.

Proof. We already proved above that the primitive recursive predicates are closed under the propositional connectives and bounded quantification. It only remains to check the base case, when the formula is atomic. The only arithmetic predicate is equality, so the atomic formula have the form $t = s$ for terms t and s. These terms are built up from successor, $+$, and $\cdot$. Hence they are equivalent to polynomial equations $p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)$, where p and q are polynomials with coefficients in $\mathbb{N}$.
The representing function of such a relation is given by

$$Rep(x) = sign(p(x) \boxminus q(x))$$

Since every polynomial is primitive recursive, and $\boxminus$ is primitive recursive, f is also primitive recursive. That completes the proof. ■

Interesting examples

Example A. Since the function $f(n) = A(n, n)$ considered above grows very rapidly, its inverse function, f^{-1}, grows very slowly. This inverse Ackermann function f^{-1} is usually denoted by α. In fact, $\alpha(n)$ is less than 5 for any practical input size n, since $A(4, 4)$ is on the order of $2^{2^{10^{19729}}}$.
Example II. One needs to know what the Goldbach Conjecture to appreciate this example: Goldbach's conjecture is still unknown. It is: every even is the sum of two primes.
Let f be the function such that if Goldbach's conjecture is true then f is 74 on all numbers less than 4 and zero elsewhere, and if Goldbach's conjecture is false then f is 17 on all less than 3 and zero elsewhere. Since we don't know whether or not Goldbach's Conjecture is true, WE DO NOT KNOW what f is. But we DO know that EITHER

1.) $f(1) = 74, f(2) = 74, f(3) = 74, f(x) = 0$ elsewhere, OR
2.) $f(1) = 17, f(2) = 17, f(x) = 0$ elsewhere.

So THERE EXISTS a program for f. In fact, we can write down two programs, and know that one of them computes f , but we don't know which one. But to show that f is computable WE DO NOT CARE WHICH ONE! The definition of computability only said THERE EXISTS a program, it didn't say we could find it.