22 The semicomputable functions ϕ_m and the Fixed Point Theorem

The completeness theorem says that (in a consistent theory) every formula A is either provable or admits a counterexample. The question is: how to find a proof of A. We are now looking at computability questions in first-order logic.

22.1 Halting computations

- We only consider the Turing machines which take bitstrings as input and produce bitstrings as output.
- Given a Turing machine T, it is possible, by identifying the halting configurations and adding more quintuples, to produce a Turing machine T' such that, whenever it halts, it halts with a single bitstring w on the tape, possibly empty, and with the r/w head positioned at the first bit in w (if $w \neq \lambda$).
- On input z, the machine T' either loops, or halts with a well-defined string w on its tape. In terms of the length-lex encoding, the machine computes a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$, where for any $n \in \mathbb{N}$, $f(n)$ is either
  - undefined, if T' loops on input z, where z is the reverse encoding of n (i.e., n is the length-lex encoding of z), or
  - r, if T' halts on input z (the same z) with bitstring w on the tape, where r is the length-lex encoding of w.
- Notation:

  $$f(n) \uparrow$$

  $f(n)$ is undefined when the machine loops, and

  $$f(n) \downarrow r$$

  if $f(n)$ is defined and the output is r.
- If y is a bitstring encoding of T, then T' has a bitstring encoding y' and the map $y \mapsto y'$ is well-defined on TM, i.e., whenever y is a valid bitstring encoding a Turing machine.

  The map can be extended to $\{0,1\}^*$, all bitstrings. Choose some $y_0 \in \text{TM}$ such that T_{y_0} always loops. Map $y \mapsto y_0$ if $y \notin \text{TM}$.

  The extended map $y \mapsto y'$ is recursive: it is computable by some TM (Turing machine) which halts on all inputs y, although it would be an extremely complicated TM.

(22.1) Definition *For $m = 0, 1, \ldots$ the partial function*

$$\phi_m : \mathbb{N} \rightharpoonup \mathbb{N}$$

is defined as follows. Let $y \in \{0,1\}^*$ be the length-lex encoding of m.

- *Let y' be as described above.*
- *Then ϕ_m is the function computed by $T_{y'}$ under the length-lex encoding.*
- *So, if $y \notin \text{TM}$ then $\phi_m \uparrow$, i.e., it is nowhere defined, its domain is $\varnothing$.*
- *If $x \in \text{TM}$ then for any $n \in \mathbb{N}$, let y be its length-lex encoding; then*

$$\phi_m(n) = \begin{cases} \uparrow & \text{if } T_x(y) \uparrow \\ k & \text{if } T_x(y) \downarrow z \text{ and} \\ & k \text{ is the length-lex decoding of } z \end{cases}$$

(22.2) Definition *By 'semicomputable function' we mean a partial function $f : \mathbb{N} \nrightarrow \mathbb{N}$ which can be computed by a Turing machine, i.e., f is one of the partial functions ϕ_m. By 'computable' we mean a function $f : \mathbb{N} \to \mathbb{N}$ which is semicomputable.*

In other words, f can be computed by a Turing machine which halts on all inputs.

Other words synonymous with 'computable': recursive, fully computable, total recursive.

Next we have a very interesting result. It is due to Kleene, I think.

(22.3) Theorem (The Fixed Point Theorem or Recursion Theorem). *Let $f : \mathbb{N} \to \mathbb{N}$ be a recursive function. Then there exists an index n such that*

$$\phi_n = \phi_{f(n)}$$

The theorem will be based on the following

(22.4) Lemma *There exists a recursive function g such that for any $m \in \mathbb{N}$,*

$$\phi_{g(m)} = \begin{cases} \uparrow & \text{if } \phi_m(m) \uparrow \\ \phi_{\phi_m(m)} & \text{if } \phi_m(m) \downarrow \end{cases}$$

Proof. Given m, let y be its length-lex encoding. We construct a Turing machine M based on information which is easy to extract from y.

If $y \notin \text{TM}$ then $\phi_m(m) \uparrow$ and M should loop on every input.

If $y \in \text{TM}$ then, on input z, M should first ignore its own input z and imitate the Universal Turing machine on input yy. This amounts to computing $\phi_m(m)$.

If $T_y(y) \uparrow$, i.e., $\phi_m(m) \uparrow$, then M will loop.

Otherwise, T_y halts on input y with output w, say. Then M should imitate U on input wz. If it halts then its output should be that of $T_w(z)$.

That is, given input z, with length-lex value n, the Turing machine M either loops or halts with output $T_{T_y(y)}(z)$, which is the length-lex encoding of

$$\phi_{\phi_m(m)}(n).$$

While the *behaviour* of M is hard to predict, its *construction* is a straightforward procedure starting with the length-lex encoding y of m. That is, a bitstring v encoding M can be given as a recursive function of y. The function $g(m)$ is the length-lex value of v. ∎

Proof of Fixed Point Theorem. Let g be as above. Given a recursive function f, we shall choose $n = g(m)$ where m is another index. We would then want to show

$$\phi_{f \circ g(m)} = \phi_{g(m)}$$

that is

$$\phi_{f \circ g(m)} = \phi_{\phi_m(m)}.$$

This can be achieved if m is an index of the function $f \circ g$, which is recursive. So

- Choose m so that ϕ_m is the recursive function $f \circ g$.
- Let $n = g(m)$.

Then

$$\phi_n = \phi_{g(m)} = \phi_{\phi_m(m)} = \phi_{f \circ g(m)} = \phi_{f(n)} \quad \blacksquare$$