## thy.1 The Normal Form Theorem

cmp:thy:nfm:sec

cmp:thy:nfm:thm:normal-form **Theorem thy.1** (Kleene's Normal Form Theorem)**.** *There are a primitive recursive relation $T(k, x, s)$ and a primitive recursive function $U(s)$, with the following property: if $f$ is any partial computable function, then for some $k$,*

$$f(x) \simeq U(\mu s\, T(k, x, s))$$

*for every $x$.*

*Proof Sketch.* For any model of computation one can rigorously define a description of the computable function $f$ and code such description using a natural number $k$. One can also rigorously define a notion of "computation sequence" which records the process of computing the function with index $k$ for input $x$. These computation sequences can likewise be coded as numbers $s$. This can be done in such a way that (a) it is decidable whether a number $s$ codes the computation sequence of the function with index $k$ on input $x$ and (b) what the end result of the computation sequence coded by $s$ is. In fact, the relation in (a) and the function in (b) are primitive recursive. □

In order to give a rigorous proof of the Normal Form Theorem, we would have to fix a model of computation and carry out the coding of descriptions of computable functions and of computation sequences in detail, and verify that the relation $T$ and function $U$ are primitive recursive. For most applications, it suffices that $T$ and $U$ are computable and that $U$ is total. explanation

It is probably best to remember the proof of the normal form theorem in slogan form: $\mu s\, T(k, x, s)$ searches for a computation sequence of the function with index $k$ on input $x$, and $U$ returns the output of the computation sequence if one can be found.

$T$ and $U$ can be used to define the enumeration $\varphi_0, \varphi_1, \varphi_2, \dots$. From now on, we will assume that we have fixed a suitable choice of $T$ and $U$, and take the equation

$$\varphi_e(x) \simeq U(\mu s\, T(e, x, s))$$

to be the *definition* of $\varphi_e$.

Here is another useful fact:

**Theorem thy.2.** *Every partial computable function has infinitely many indices.*

Again, this is intuitively clear. Given any (description of) a computable function, one can come up with a different description which computes the same function (input-output pair) but does so, e.g., by first doing something that has no effect on the computation (say, test if $0 = 0$, or count to 5, etc.). The index of the altered description will always be different from the original index. Both are indices of the same function, just computed slightly differently.

## Photo Credits

## Bibliography

 