-NoValue-

thy.1 The Normal Form Theorem

cmp:thy:nfm:sec

Suppose we can describe definitions of computable functions, and test if some putative description of the complete record of the computation of that function on some input is correct. Then it stands to reason that independently of the model of computation, we can determine the value of any computable function f on any input x as follows:

1. Search through all possible descriptions of records of computation.
2. Test if a given record is the record of a computation of $f(x)$.
3. Extract the value of $f(x)$ from the correct record if we have found it.

That this is in fact true is the content of Kleene's normal form theorem.

cmp:thy:nfm:thm:normal-form

Teorema thy.1 (Kleene's Normal Form Theorem). *There is a primitive recursive relation $T(e, x, s)$ and a primitive recursive function $U(s)$, with the following property: if f is any partial computable function, then for some e,*

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

for every x.

Esboço de prova. For any model of computation one can rigorously define a description of the computable function f and code such description using a natural number e. One can also rigorously define a notion of "computation sequence" which records the process of computing the function with index e for input x. Such a computation sequence can likewise be coded as a number s. This can be done in such a way that

1. the relation $T(e, x, s)$, which holds iff a number s codes the computation sequence of the function with index e on input x, and
2. the function $U(s)$ which maps a computation sequence coded by s to the end result of that computation

are both computable. In fact, the relation T and the function U are primitive recursive. □

explanation

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.

It is probably best to remember the proof of the normal form theorem in slogan form: $\mu s\, T(e, x, s)$ searches for a computation sequence of the function

with index e on input x, and U returns the output of the computation sequence if one can be found.

If the model of computation is the partial recursive functions (which is what Kleene originally used), it shows that only a single use of unbounded search, O.s., a single $\mu y\ f(\vec{x}, y)$ operator is necessary for the definition of any function. In this sense it shows that any partial recursive function has a normal form.

T and U can be used to define the enumeration φ_0, φ_1, φ_2, ... 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 φ_e.

Here is another useful fact:

Teorema 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, p.ex., 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.

Créditos das Imagens

Referências Bibliográficas