und.1 Universal Turing Machines

tur:und:uni:sec

In **??** we discussed how every Turing machine can be described by a finite sequence of integers. This sequence encodes the states, alphabet, start state, and instructions of the Turing machine. We also pointed out that the set of all of these descriptions is enumerable. Since the set of such descriptions is denumerable, this means that there is a surjective function from $\mathbb{N}$ to these descriptions. Such a surjective function can be obtained, for instance, using Cantor's zig-zag method. It gives us a way of enumerating all (descriptions) of Turing machines. If we fix one such enumeration, it now makes sense to talk of the 1st, 2nd, ..., eth Turing machine. These numbers are called *indices*.

Definition und.1. If M is the eth Turing machine (in our fixed enumeration), we say that e is an *index* of M. We write M_e for the eth Turing machine.

A machine may have more than one index, e.g., two descriptions of M may differ in the order in which we list its instructions, and these different descriptions will have different indices.

Importantly, it is possible to give the enumeration of Turing machine descriptions in such a way that we can effectively compute the description of M from its index, and to effectively compute an index of a machine M from its description. By the Church–Turing thesis, it is then possible to find a Turing machine which recovers the description of the Turing machine with index e and writes the corresponding description on its tape as output. The description would be a sequence of blocks of 1's (representing the positive integers in the sequence describing M_e).

Given this, it now becomes natural to ask: what functions of Turing machine indices are themselves computable by Turing machines? What properties of Turing machine indices can be decided by Turing machines? An example: the function that maps an index e to the number of states the Turing machine with index e has, is computable by a Turing machine. Here's what such a Turing machine would do: started on a tape containing a single block of e 1's, it would first decode e into its description. The description is now represented by a sequence of blocks of 1's on the tape. Since the first element in this sequence is the number of states. So all that has to be done now is to erase everything but the first block of 1's and then halt.

A remarkable result is the following:

tur:und:uni:thm:universal-tm

Theorem und.2. *There is a* universal Turing machine U *which, when started on input* $\langle e, n\rangle$

1. *halts iff* M_e *halts on input* n*, and*
2. *if* M_e *halts with output* m*, so does* U*.*

U thus computes the function $f : \mathbb{N} \times \mathbb{N} \nrightarrow \mathbb{N}$ given by $f(e, n) = m$ if M_e started on input n halts with output m, and undefined otherwise.

Proof. To actually produce U is basically impossible, since it is an extremely complicated machine. But we can describe in outline how it works, and then invoke the Church–Turing thesis. When it starts, U's tape contains a block of e 1's followed by a block of n 1's. It first "decodes" the index e to the right of the input n. This produces a list of numbers (i.e., blocks of 1's separated by 0's) that describes the instructions of machine M_e. U then writes the number of the start state of M_e and the number 1 on the tape to the right of the description of M_e. (Again, these are represented in unary, as blocks of 1's.) Next, it copies the input (block of n 1's) to the right—but it replaces each 1 by a block of three 1's (remember, the number of the 1 symbol is 3, 1 being the number of ▷ and 2 being the number of 0). At the left end of this sequence of blocks (separated by 0 symbols on the tape of U), it writes a single 1, the code for ▷.

U now has on its tape: the index e, the number n, the code number of the start state (the "current state"), the number of the initial head position 1 (the "current head position"), and the initial contents of the "tape" (a sequence of blocks of 1's representing the code numbers of the symbols of M_e—the "symbols"—separated by 0's).

It now simulates what M_e would do if started on input n, by doing the following:

1. Find the number k of the "current head position" (at the beginning, that's 1),
2. Move to the kth block in the "tape" to see what the "symbol" there is,
3. Find the instruction matching the current "state" and "symbol," tur:und:uni:find-inst
4. Move back to the kth block on the "tape" and replace the "symbol" there with the code number of the symbol M_e would write,
5. Move the head to where it records the current "state" and replace the number there with the number of the new state,
6. Move to the place where it records the "tape position" and erase a 1 or add a 1 (if the instruction says to move left or right, respectively).
7. Repeat.[1]

If M_e started on input n never halts, then U also never halts, so its output is undefined.

If in step (3) it turns out that the description of M_e contains no instruction for the current "state"/"symbol" pair, then M_e would halt. If this happens, U erases the part of its tape to the left of the "tape." For each block of three 1's (representing a 1 on M_e's tape), it writes a 1 on the left end of its own tape,

[1] We're glossing over some subtle difficulties here. E.g., U may need some extra space when it increases the counter where it keeps track of the "current head position"—in that case it will have to move the entire "tape" to the right.

and successively erases the "tape." When this is done, U's tape contains a single block of 1's of length m.

If U encounters something other than a block of three 1's on the "tape," it immediately halts. Since U's tape in this case does not contain a single block of 1's, its output is not a natural number, i.e., $f(e, n)$ is undefined in this case.□

Photo Credits

Bibliography