Non-Computable Functions (part 2)

1 The Busy Beaver Function

- $\textbf{Productivity}(\boldsymbol{M}) = \begin{cases} k, & \text{if } M \text{ yields output } k \text{ on an empty input} \\ 0, & \text{otherwise} \end{cases}$
- $\boldsymbol{BB(n)} =$ the productivity of the most productive (one-symbol) Turing Machine with n states or fewer.

2 $BB(n)$ is not Turing-computable

- Assume for *reductio*: Turing Machine M^{BB} computes $BB(n)$.
- Construct Turing Machine M^I, which behaves as follows on an empty input:

  Step 1: Print a sequence of k ones, for a certain k (specified below).
  Result: k.

  Step 2: Duplicate your string of ones.
  Result: $2k$.

  Step 3 Apply BB to your string of ones (using M^{BB}).
  Result: $BB(2k)$.

  Step 4 Add one to your string of ones.
  Result: $BB(2k) + 1$.

- Let $k = b + c + d$

  $b =$ the number of states used in Step 2 (to duplicate)

  $c =$ the number of states used in Step 3 (to apply BB)

  $d =$ the number of states used in Step 4 (to add one)

  Note: since a Turing Machine can output k using k states,

$$\overline{M^I} = k + b + c + d = 2k$$

- M^{BB} is impossible:
  - At Stage 3, it produces as long a sequence of ones as a machine with $2k$ states could possibly produce.
  - But (as noted above) $\overline{M^I} = 2k$.
  - So at Stage 3, it produces as long a sequence of ones as it itself could possibly produce.
  - So at Stage 4, it produces a *longer* string of ones than it itself could possibly produce.
- So M^H isn't computable after all.

3 The Universal Turing Machine

There is a **Universal Turing Machine**, M^U, which does the following:

- if the mth Turing Machine halts given input n, leaving the tape in configuration p, then M^U halts given input $\langle m, n \rangle$ leaving the tape in configuration p.
- if the mth Turing Machine never halts given input n, then M^U never halts given input $\langle m, n \rangle$.

4 The Fundamental Theorem

The reason Turing Machines are so valuable is that it is possible to prove the following theorem:

Fundamental Theorem of Turing Machines A function from natural numbers to natural numbers is Turing-computable if and only if it can be computed by an ordinary computer, assuming unlimited memory and running time.

- One shows that every Turing-computable function is computable by an ordinary computer (given unlimited memory and running time) by showing that one can program an ordinary computer to simulate any given Turing Machine.

- One shows that every function computable by an ordinary computer (given unlimited memory and running time) is Turing-computable by showing that one can find a Turing Machine that simulates any given ordinary computer.

5 Church-Turing

Computer scientists tend to think that something stronger than the Fundamental Theorem is true:

Church-Turing Thesis A function is Turing-computable if and only if it can be computed algorithmically.

For a problem to be solvable **algorithmically** is for it to be possible to specify a finite list of instructions for solving the problem such that:

1. Following the instructions is guaranteed to yield a solution to the problem, in a finite amount of time.
2. The instructions are specified in such a way that carrying them out requires no ingenuity of any kind: they can be followed mechanistically.
3. No special resources are required to carry out the instructions: they could in principle be carried out by a machine built from transistors.
4. No special physical conditions are required for the computation to succeed (no need for faster-than-light travel, special solutions to Einstein's equations, etc).