Non-Computable Functions

1 The Main Result

- We’ll focus on functions $f : \mathbb{N} \to \mathbb{N}$.
- For a computer program to compute $f$ is for it to yield $f(n)$ as output whenever it is given $n$ as input ($n \in \mathbb{N}$).
- **Theorem:** not every function is computable.
  (And I can give you examples!)

2 The Overall Plan

- Turing Machines are computers of an especially simple sort.
- We’ll see that some functions are not Turing-computable.
- But: any function that can be computed using an ordinary computer is also computed by some Turing Machine.

3 Computing functions on a Turing Machine

- Simplifying Assumptions:
  - We’ll focus on one symbol Turing Machines (where the only admissible symbols are ones and blanks).
  - We’ll assume that the tape is only unbounded on the right.
- Turing Computability:
  - $M$ computes a function $f(x)$ if and only if it delivers $f(n)$ as output whenever it is given $n$ as input.
  - $M$ takes $n$ ($n \in \mathbb{N}$) as input if it starts out with a tape that contains only a sequence of $n$ ones (with the reader positioned at the left-most one, if $n > 0$).
4 Coding Turing Machines as Numbers

The Plan
Turing Machine → Sequence of symbols → Sequence of numbers → Unique number

Sequence of symbols → Sequence of numbers

State Symbols:                  Tape Symbols:                  Movement Symbols:

“0” → 0                      “-” → 0                       “r” → 0

“1” → 1                      “1” → 1                       “*” → 1

⋯

Sequence of numbers → Unique number

Codes the sequence \( \langle n_1, n_2, \ldots, n_k \rangle \) as the number:

\[
p_1^{n_1+1} \cdot p_2^{n_2+1} \cdot \ldots \cdot p_k^{n_k+1}
\]

where \( p_i \) is the \( i \)th prime number.

(Treat any number that doesn’t code a valid sequence of command lines as a code for the “empty” Turing Machine.)

4.1 An example

\[
2310 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11
\]

\[
\downarrow
\]

\[
2^{0+1} \cdot 3^{0+1} \cdot 5^{0+1} \cdot 7^{0+1} \cdot 11^{0+1}
\]

\[
\downarrow
\]

\[
0 \ 0 \ 0 \ 0 \ 0
\]

\[
\downarrow
\]

\[
0 \ _ \ _ \ r \ 0
\]
5 The Halting Function

• $H(n, m) = \begin{cases} 
1 & \text{if the } n\text{th Turning Machine halts when given input } m; \\
0 & \text{otherwise.} 
\end{cases}$

For instance: $H(2310, 0) = 0$ and $H(2310, 2310) = 1$.

• $H(n) = H(n, n)$

For instance: $H(2310) = 1$.

6 $H(n)$ is not Turing-computable

• Assume for reductio: Turing Machine $M^H$ computes $H(n)$.

• Construct Turing Machine $M^I$, which behaves as follows on input $k$:

  **Step 1:** Check whether $H(k)$ (using $M^H$).
  
  **Step 2:** \[ \begin{cases} 
  \text{If } H(k) = 1, \text{ go right forever.} \\
  \text{If } H(k) = 0, \text{ halt.} 
  \end{cases} \]

• Informally: What happens when you run $M^I$ on input $M^I$? It figures out whether it itself would halt on input $M^I$. If the answer is yes, it goes off on an infinite task; if the answer is no, it immediately halts.

• Formally: $H(M^I) 1$ or $0$?
  
  – Suppose $H(M^I) = 1$. Then (by Step 2) $M^I$ goes right forever on input $M^I$. So $H(M^I) = 0$.
  
  – Suppose $H(M^I) = 0$. Then (by Step 2) $M^I$ halts on input $M^I$. So $H(M^I) = 1$.

• So $M^I$ is impossible. So $M^H$ isn’t computable after all.
7 The Busy Beaver Function

- \( \text{Productivity}(M) = \begin{cases} 
  k, & \text{if } M \text{ yields output } k \text{ on an empty input} \\
  0, & \text{otherwise} 
\end{cases} \)

- \( BB(n) = \) the productivity of the most productive (one-symbol) Turing Machine with \( n \) states or fewer.

8 \( BB(n) \) is not Turing-computable

- Assume for \textit{reductio}: Turing Machine \( M^{BB} \) computes \( BB(n) \).

- Construct Turing Machine \( M^{I} \), which behaves as follows on an empty input:

  \textbf{Step 1:} Print a sequence of \( k \) ones, for a certain \( k \) (specified below).
  
  \textit{Result:} \( k \).

  \textbf{Step 2:} Duplicate your string of ones.
  
  \textit{Result:} \( 2k \).

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

  \textbf{Step 4} Add one to your string of ones.
  
  \textit{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)

  \textit{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) $M^{I} = 2k$.
– 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.