Computability Theory and Big Numbers

Alexander Davydov

August 30, 2018

Classic Large Numbers

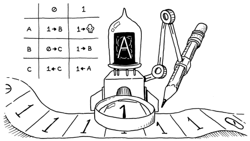
- Factorial
    - $100! \approx 9.33 \times 10^{157}$
    - $(100!)! >> 100!$
- Power towers
    - $3 \uparrow\uparrow 4 = 3^{3^{3^{3}}} = 3^{7625597500000}$

Graham's Number

- g_{64}
  - $g_n = \begin{cases} 3 \uparrow\uparrow\uparrow\uparrow 3 & \text{if } n = 1 \\ 3 \uparrow^{g_{n-1}} 3 & \text{if } n \geq 2 \text{ and } n \in \mathbb{N} \end{cases}$
- So large that the observable universe is too small to contain the digital representation of Grahams Number, even if each digit were one Planck volume
  - 1 Planck volume $= 4.2217 \times 10^{-105} m^3$

Turing Machines

- A Turing machine serves as a mathematical model for computation
- Informally:
  - One-dimensional tape of cells that extends infinitely in either direction
  - Each cell contains a symbol from the "alphabet" of the machine
    - Typically 0, 1, and possibly a blank symbol
  - Machine contains a "head" that reads the symbol underneath it
  - Machine is in one of finitely many "states" that determine what the machine does
  - At each step, based on the symbol that the head reads, the head will overwrite the symbol that it just read and then move either to the left or right and then enter a new state
  - The machine has a q_{HALT} state. When the machine reaches the q_{HALT} state, the machine halts and whatever is written on the tape is outputted



	∅	1
A	$1 \rightarrow B$	$1 \rightarrow$ 💀
B	$\emptyset \rightarrow C$	$1 \rightarrow B$
C	$1 \leftarrow C$	$1 \leftarrow A$

Formal Definition of a Turing Machine

- A Turing Machine M is described by a tuple (Γ, Q, δ) containing:
  - A set Γ of the symbols that M's tape can contain. Γ is typically called the alphabet of M
  - A set Q of all possible states that M can be in. Q contains a designated start state and q_{HALT}
  - A transition function $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$ which determines based on the machines current state and the symbol it reads what symbol the machine will write, what new state it will go to, and whether the machine moves left or right

Example of a Turing Machine that adds 1

1. When starting at the left end of the number, walk to the right end of the number
2. Walking right to left, change all 1s to 0s
3. At first 0, change to 1 and halt

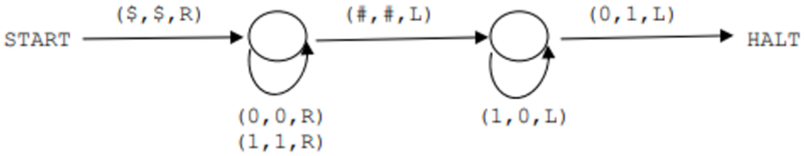
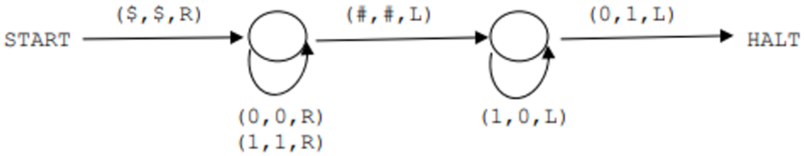


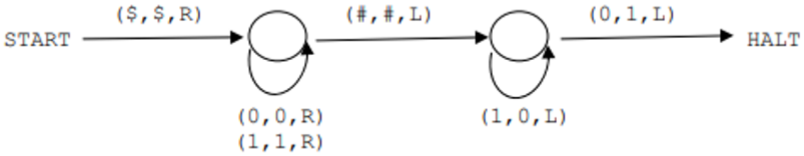
Figure: State Diagram for this Algorithm

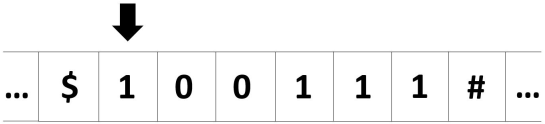
Example of a Turing Machine that adds 1



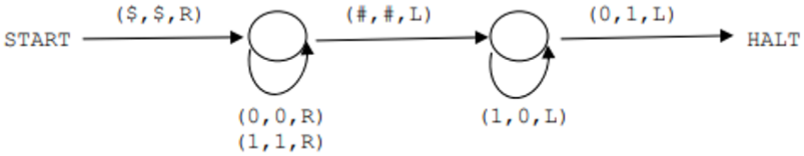
...	$	1	0	0	1	1	1	#	...

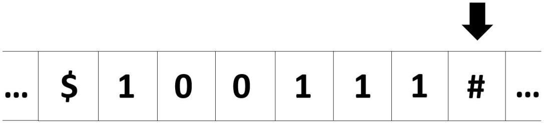
Example of a Turing Machine that adds 1



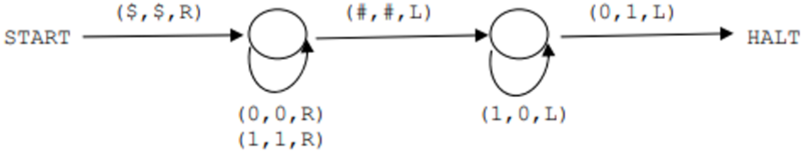


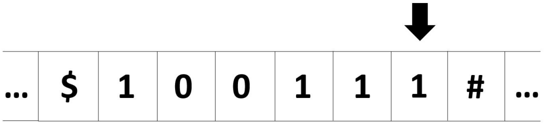
Example of a Turing Machine that adds 1



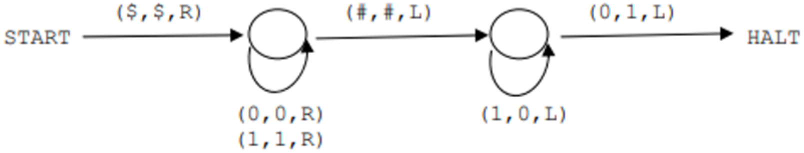


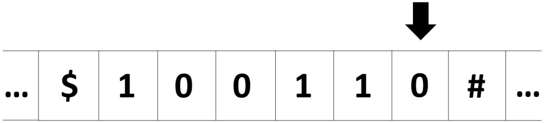
Example of a Turing Machine that adds 1



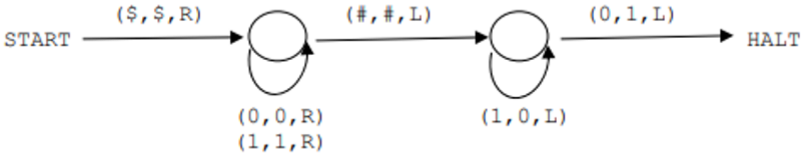


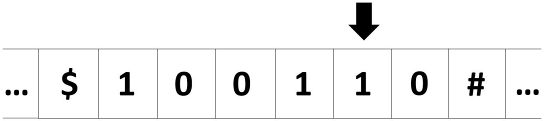
Example of a Turing Machine that adds 1



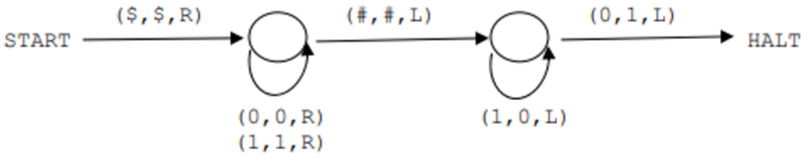


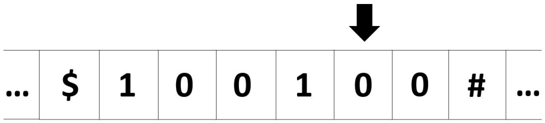
Example of a Turing Machine that adds 1



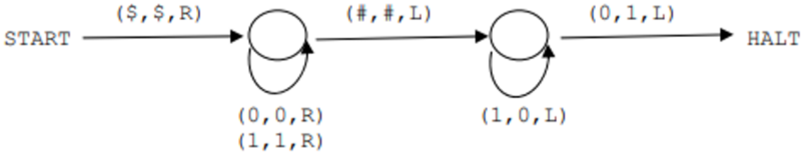


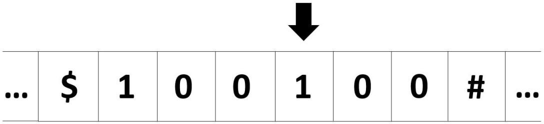
Example of a Turing Machine that adds 1



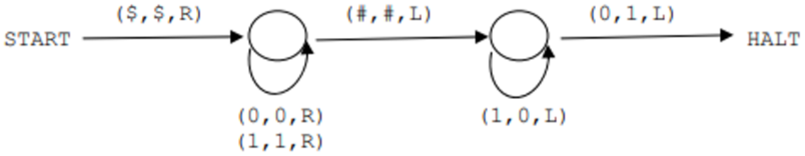


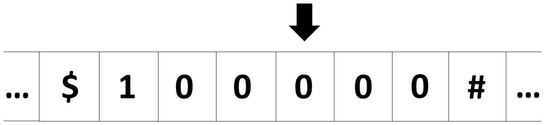
Example of a Turing Machine that adds 1



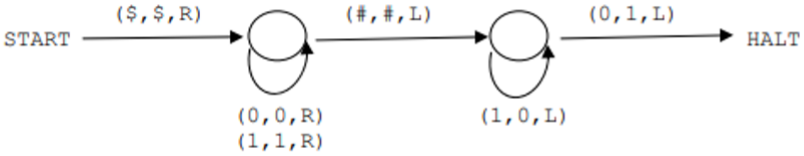


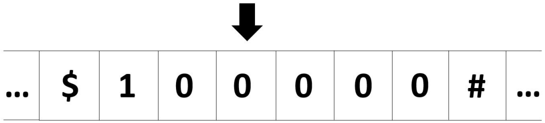
Example of a Turing Machine that adds 1



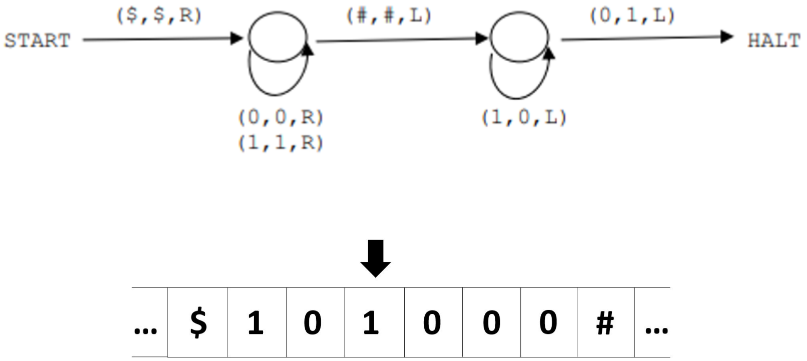


Example of a Turing Machine that adds 1





Example of a Turing Machine that adds 1



Computable and Uncomputable Functions

Definition

A function $f : \mathbb{N} \mapsto \mathbb{N}$ is said to be computable if there exists a TM M such that if you give n as input to M, M will output $f(n)$

- Probably almost every function you can think of is computable

The Halting Problem

- Define a function *HALT* which takes as input a Turing machine, M, and some input, x, and outputs 1 if the given Turing machine halts on x, 0 otherwise

Theorem

HALT is not computable by any Turing Machine

The Busy Beaver Function

- First introduced by Tibor Radó in his 1962 paper "On Non-Computable Functions"
- The Busy Beaver function $\Sigma(n)$ is defined by the maximum number of 1s written on a blank tape (all 0s) by a two symbol $(0, 1)$, n-state (not counting q_{HALT}) Turing machine that halts

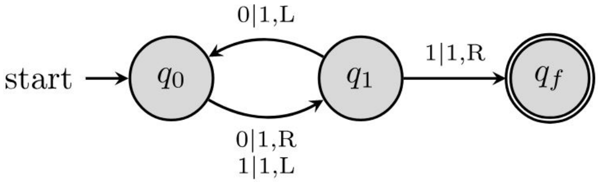
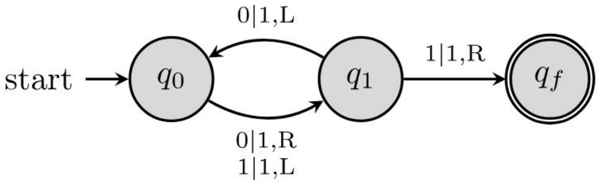
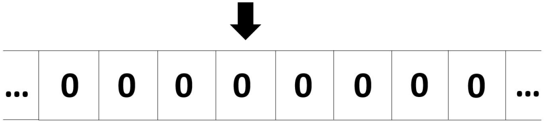

$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[r, bend right, "0|1,R \\ 1|1,L"'] & q_1 \arrow[l, bend right, "0|1,L"'] \arrow[r, "1|1,R"] & q_f
\end{tikzcd}$$

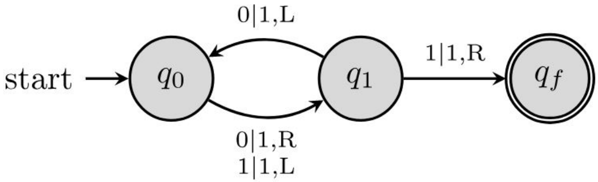
Figure: 2-state TM that writes the most 1's

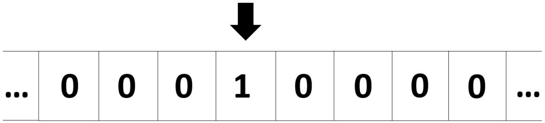
Example of a 2-state Busy Beaver



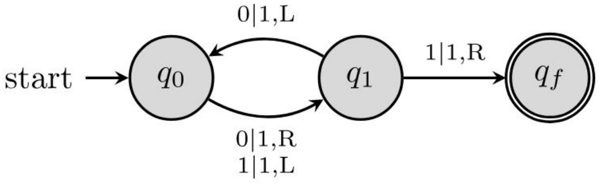


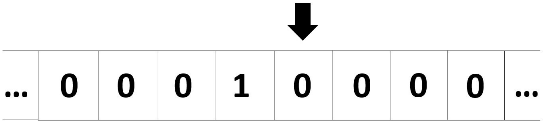
Example of a 2-state Busy Beaver



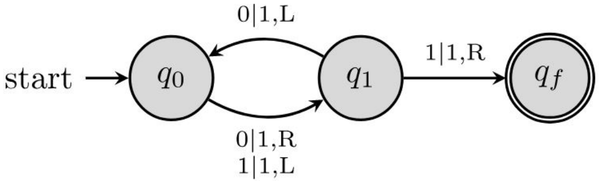


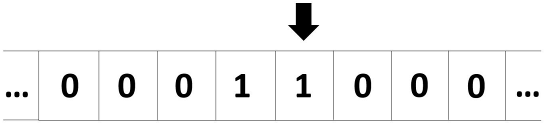
Example of a 2-state Busy Beaver



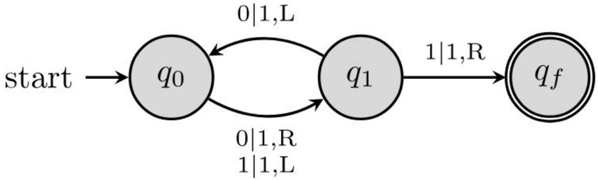


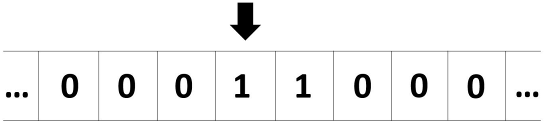
Example of a 2-state Busy Beaver



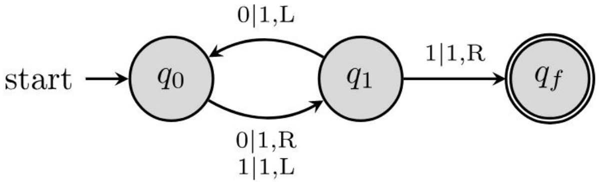


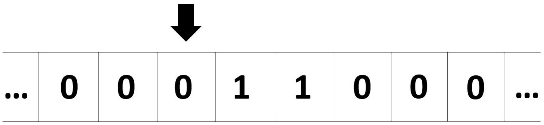
Example of a 2-state Busy Beaver



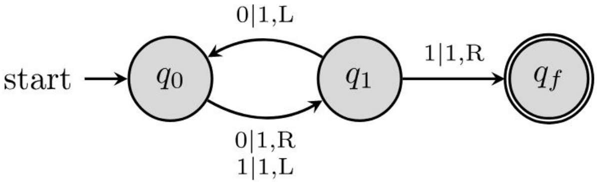


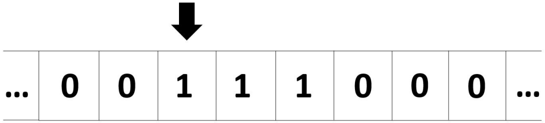
Example of a 2-state Busy Beaver



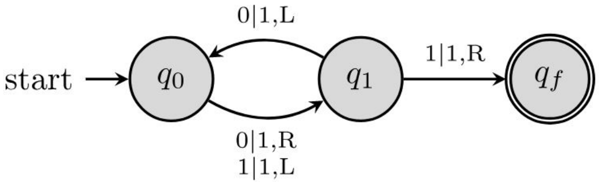


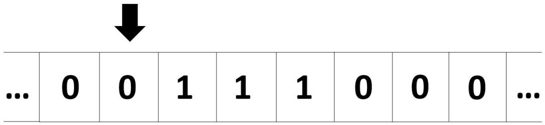
Example of a 2-state Busy Beaver



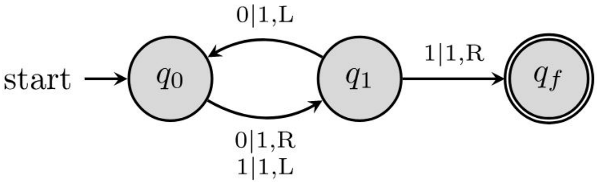


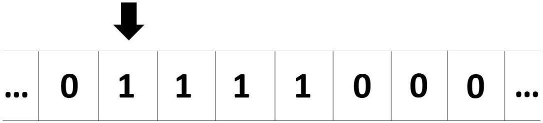
Example of a 2-state Busy Beaver



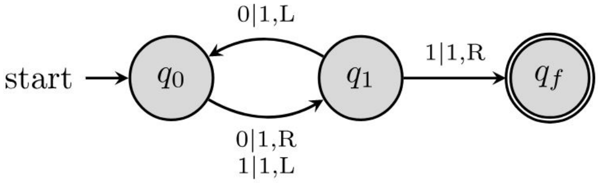


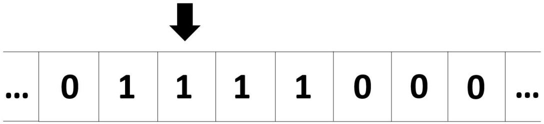
Example of a 2-state Busy Beaver





Example of a 2-state Busy Beaver





Maximum Shifts Function

- Similar to the Busy Beaver Function
- The Maximum shifts function $S(n)$ is defined by the maximum number of shifts on a blank tape (left or right) taken by a two symbol $(0, 1)$, n-state (not counting q_{HALT}) Turing machine that halts

Values of $\Sigma(n)$ and $S(n)$

- $\Sigma(1) = 1, S(1) = 1$

Values of $\Sigma(n)$ and $S(n)$

- $\Sigma(1) = 1, S(1) = 1$
- $\Sigma(2) = 4, S(2) = 6$

Values of $\Sigma(n)$ and $S(n)$

- $\Sigma(1) = 1, S(1) = 1$
- $\Sigma(2) = 4, S(2) = 6$
- $\Sigma(3) = 6, S(3) = 21$

Values of $\Sigma(n)$ and $S(n)$

- $\Sigma(1) = 1, S(1) = 1$
- $\Sigma(2) = 4, S(2) = 6$
- $\Sigma(3) = 6, S(3) = 21$
- $\Sigma(4) = 13, S(4) = 107$

Values of $\Sigma(n)$ and $S(n)$

- $\Sigma(1) = 1, S(1) = 1$
- $\Sigma(2) = 4, S(2) = 6$
- $\Sigma(3) = 6, S(3) = 21$
- $\Sigma(4) = 13, S(4) = 107$
- $\Sigma(5) \geq 4098, S(5) \geq 47176870$

Values of $\Sigma(n)$ and $S(n)$

- $\Sigma(1) = 1, S(1) = 1$
- $\Sigma(2) = 4, S(2) = 6$
- $\Sigma(3) = 6, S(3) = 21$
- $\Sigma(4) = 13, S(4) = 107$
- $\Sigma(5) \geq 4098, S(5) \geq 47176870$
- $\Sigma(6) > 3.5 \times 10^{18267}, S(6) > 7.4 \times 10^{36534}$

How Fast do Σ and S Grow?

- Σ and S are both uncomputable
- The functions Σ and S grow faster than any computable function
  - If we had a computable function $f(n)$ that grew faster than Σ or S, we could use f to compute $HALT$
- $\Sigma(64)$ has been proven to be larger than Graham's number
  - A 64 state TM was created that follows a fast-growing hierarchy that dwarfs Graham's Number
  - Some sources say even $\Sigma(18)$ is larger than Graham's number
- $\Sigma(1919)$ is unknowable given the usual axioms of set theory (ZFC set theory with the axiom of choice)
  - A 1919 state TM that cannot be proven to run forever

Application of $S(n)$

- Some unsolved conjectures in mathematics (Twin primes, Goldbach, Collatz, etc.) could be checked using the values of $S(n)$
- Suppose we wrote a Turing machine with 100 states that checked the Collatz conjecture and that we have an upper bound for the value of $S(100)$
- We could run the TM for $S(100)$ steps and if it does not halt, we know that it will never halt so the Collatz conjecture would be true (this TM only halts if a counterexample is found)!

Good News?

- This application is not practical for several reasons
  - It is very difficult to find values (or even upper bounds) for $S(n)$
  - We do not have enough computational capacity in the known part of the universe to even perform S(6) operations
    - Not many useful programs only have 6 states
- Finding values for Σ and S are in some regard, much “harder” than these unproved conjectures because they encapsulate these conjectures!

Beyond Busy Beavers

- We can give Turing machines access to an Oracle which can give the TM more information
  - Oracle machines can query the oracle and determine whether what is written on the tape is part of a predetermined set
  - The HALT-oracle machine will check whether or not the TM and input encoded on the tape will halt
- TMs with oracles have larger values of Σ and S

References I

Sanjeev Arora and Boaz Barak
Complexity Theory: A Modern Approach.
Cambridge University Press 2009.

Adam Yedidia and Scott Aaronson
A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory

Scott Aaronson
Who can name the bigger number?

- Thank you to David Bekkerman for spending the Summer with me talking about Computability and Complexity Theory!