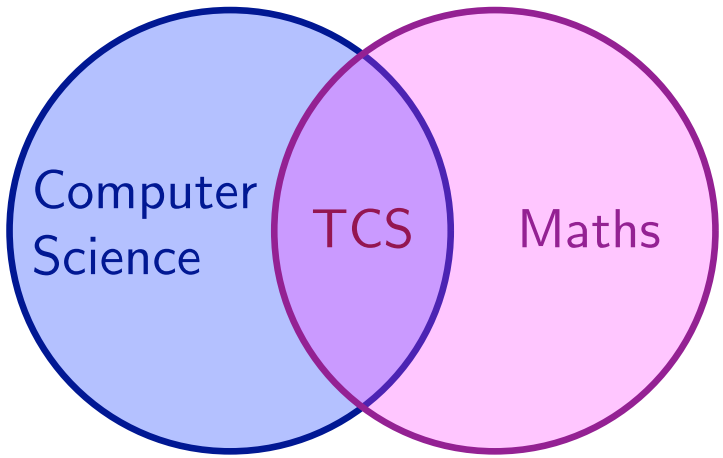
Programming counter machines

Dillon Mayhew

Outreach Coordinator and Professor of Theoretical Computer Science at the School of Computer Science, University of Leeds.

Email: d.mayhew@leeds.ac.uk



UNIVERSITY OF LEEDS

Model computers

In theoretical computer science, we often study model computers:

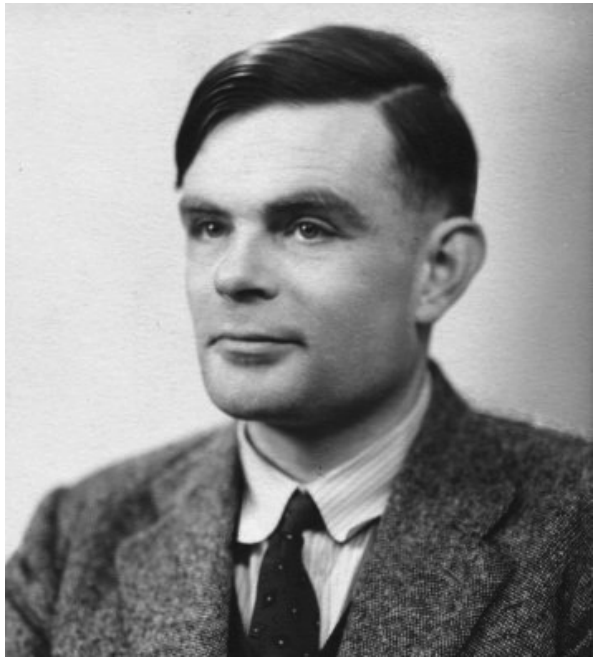
- mathematically defined computing machines,
- complicated enough to carry out any computation we can imagine,
- simple enough to analyse using mathematical ideas.

Church-Turing Thesis

The most famous example of a model computer is a Turing machine.

All useful model computers are Turing complete. This means that any model computer can simulate any other model computer.

We believe any imaginable program can be simulated by any Turing-complete machine. This is known as the Church-Turing Thesis.



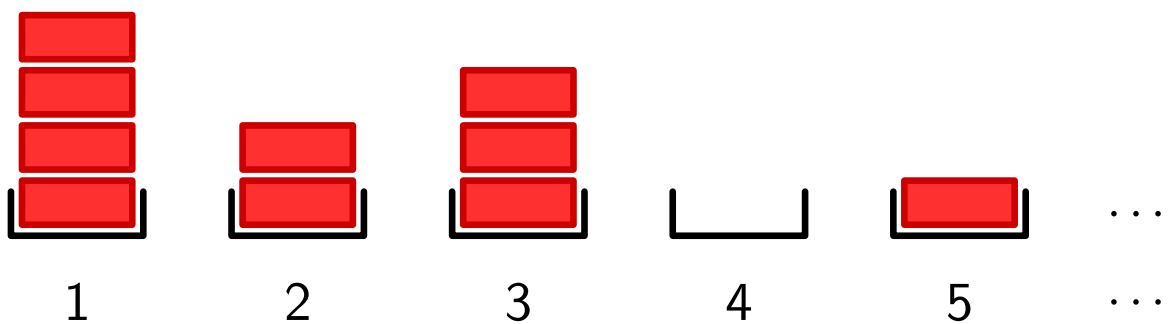
Alan Turing
(1912 – 1954)

Counter machines

We will study a model computer called a counter machine.

A counter machine consists of a sequence of registers (stacks). Each register contains a number of blocks.

We have a very simple programming language that lets us change the number of the blocks in the registers.



Counter machines

The programming instructions come in seven types:

- `INC` $\#A$ (Increase register $\#A$ by one)
- `DEC` $\#A$ (Decrease register $\#A$ by one — if empty, no change)
- `COPY` $\#A$, $\#B$ (Copy register $\#A$ to $\#B$ — overwrites B, no change to A)
- `CLR` $\#A$ (Clear register $\#A$)
- `GOTO` $\#N$ (Go to the instruction on line $\#N$)
- `IF` $\#A$, $\#N$ (If register $\#A$ is empty, go to line $\#N$)
- `HALT`

Instructions

- Go to `https://github.com/dillon128/CounterMachines`
- Select `CounterMachines.txt`
- Select the code in this file and copy it using `Ctrl-C`.
- Open `https://sagecell.sagemath.org/` in another tab.
- Paste the text into the cell using `Ctrl-V`.

Challenges

Program counter machines to:

(1) calculate $x + y$

(2) calculate $x \times y$,

(3) calculate 2^x,

(4) calculate x^y,

(5) calculate the absolute value of $x - y$,

(6) sort x and y into increasing order,

(7) find the remainder after dividing x by y,

(8) sort x, y, and z into increasing order,

(9) test to see if x is prime.

The Busy Beaver problem

We want to write a program that will run as long as possible, starting with empty stacks, but we insist that the program must eventually halt.

If we are limited to at most 7 lines of instructions, how many steps can we make the program run?

What about 12, 17?

In general, this is known as the Busy Beaver problem.

The Busy Beaver problem

My personal records for the Busy Beaver problem.

Number of lines	Number of steps before halting
7	14
12	46
17	

The Busy Beaver problem

My personal records for the Busy Beaver problem.

Number of lines	Number of steps before halting
7	14
12	46
17	17,433,922,066

The Busy Beaver problem

The Busy Beaver problem is essentially impossible to solve.

If there was a good way to solve the Busy Beaver problem, then we could find solutions to some of the most famous open problems in mathematics.

The Busy Beaver problem

The Goldbach Conjecture

If n is an even positive integer, and $n > 2$, then n can be expressed as the sum of two primes.

Examples

$4 = 2 + 2$, $6 = 3 + 3$, $8 = 5 + 3$, $10 = 7 + 3$, $12 = 7 + 5$,...

We could construct a counter machine that examines each positive even integer in turn and tests to see if it can be written as the sum of two primes.

If it discovers a counterexample to the Goldbach Conjecture, then it will halt, otherwise it will run forever.

To find if the Goldbach Conjecture is true, we need to decide whether the machine will run forever.

The Busy Beaver problem

Imagine that we can write this program with at most 50 instructions, and we know that the solution to the Busy Beaver problem for 50 instructions is a machine that runs for 50^7 steps.

We could run our program for $50^7 + 1$ steps. If it halts before that point, then we know there is a counterexample to the Goldbach Conjecture.

If it is still running at $50^7 + 1$ steps, then we know it will run forever, or else it would be the solution to the busy beaver problem. Therefore there is no counterexample, so the Goldbach Conjecture is true.