Problems, problems, problems...

Damir D. Dzhafarov
University of Connecticut

April 10, 2017

Functions.

Recall from your math classes that...

A function is an **assignment** to each **input** of some **output**.

Functions.

Recall from your math classes that...

A function is an **assignment** to each **input** of some **output**.

Examples.

- The function assigning each person in this room their height.

Functions.

Recall from your math classes that...

A function is an **assignment** to each **input** of some **output**.

Examples.

- The function assigning each person in this room their height.
- The function assigning each of my students their midterm score.

Functions.

Recall from your math classes that...

A function is an **assignment** to each **input** of some **output**.

Examples.

- The function assigning each person in this room their height.
- The function assigning each of my students their midterm score.
- The function assigning each 9-digit US phone number its owner.

Functions.

Recall from your math classes that...

A function is an **assignment** to each **input** of some **output**.

Examples.

- The function assigning each person in this room their height.
- The function assigning each of my students their midterm score.
- The function assigning each 9-digit US phone number its owner.
- The function assigning each person their biological mother.

Functions.

Recall from your math classes that...

A function is an **assignment** to each **input** of some **output**.

Examples.

- The function assigning each person in this room their height.
- The function assigning each of my students their midterm score.
- The function assigning each 9-digit US phone number its owner.
- The function assigning each person their biological mother.
- The function assigning each whole number its square ($F(n) = n^2$).

Ranges.

The range of a function is the **set of all its outputs**.

Ranges.

The range of a function is the **set of all its outputs**.

Examples.

- The range of the function that assigns each person in this room their height might include numbers like 5’6”, 4’2”, 6’1”, etc., but likely won’t include 0’2”, and almost certainly not 12’3”.

Ranges.

The range of a function is the **set of all its outputs**.

Examples.

- The range of the function that assigns each person in this room their height might include numbers like 5'6", 4'2", 6'1", etc., but likely won't include 0'2", and almost certainly not 12'3".

- The range of the function assigning each 9-digit phone number its owner will include me, but not my friend Denis who doesn't have a phone.

Ranges.

The range of a function is the **set of all its outputs**.

Examples.

- The range of the function that assigns each person in this room their height might include numbers like 5'6", 4'2", 6'1", etc., but likely won't include 0'2", and almost certainly not 12'3".
- The range of the function assigning each 9-digit phone number its owner will include me, but not my friend Denis who doesn't have a phone.
- The range of the function assigning each whole number its square will include 1 (because $1 = 1^2$), and it will include 9 (because $9 = 3^2$), but it won't include 5 (because there is no whole number n such that $n^2 = 5$).

Ranges (continued).

Ranges are super **useful!**

...but only if we know the range, i.e., if we know which things are in the range and which things aren't.

Ranges (continued).

Ranges are super **useful!**

...but only if we know the range, i.e., if we know which things are in the range and which things aren't.

Example.

- If I know the range of the function that assigns each of my students their midterm score, I can compute the mean and the median of the scores, and curve my students' grades.

Ranges (continued).

Ranges are super **useful!**

...but only if we know the range, i.e., if we know which things are in the range and which things aren't.

Example.

- If I know the range of the function that assigns each of my students their midterm score, I can compute the mean and the median of the scores, and curve my students' grades.

In all our examples, figuring out the range seems **doable**. But for some functions, it is clearly easier to do than for others.

How hard is it to know the range of a function, in general?

The Range Problem.

Problem.

Suppose we are given:

- A function F assigning a whole number to each whole number.
- A whole number k.

Figure out whether or not k is in the range of the function F.

The Range Problem.

Problem.

Suppose we are given:

- A function F assigning a whole number to each whole number.
- A whole number k.

Figure out whether or not k is in the range of the function F.

Another problem.

Another problem.

Spinning beachballs are super **annoying!**

Another problem.

Spinning beachballs are super **annoying!**

In practice, we can interrupt a spinning beachball (reboot our computer), but anything not stored in permanent memory will be lost.

On the other hand, losing data is better than waiting forever...

Another problem.

Spinning beachballs are super **annoying!**

In practice, we can interrupt a spinning beachball (reboot our computer), but anything not stored in permanent memory will be lost.

On the other hand, losing data is better than waiting forever...

Million $$$ iPhone App Idea.

- Design an app that, when you enter your computer's current configuration after you get a spinning beachball, tells you whether to reboot or wait it out.

How hard would it be to write an app like this?

The Spinning Beachball Problem.

Problem.

Suppose we are given:

- A computer, running a program.
- A spinning beachball on our screen.

Figure out whether the beachball will stop spinning or not.

Algorithms.

What is a computer program, anyway?

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.
- Here is a program (algorithm) to find all prime numbers:

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.
- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.
- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
					⋮				

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~ 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~
11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~
~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~
31 ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ 37 ~~38~~ ~~39~~ ~~40~~
⋮

Algorithms.

What is a computer program, anyway?

Example.

- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

~~1~~ 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~
11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~
~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~
31 ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ 37 ~~38~~ ~~39~~ ~~40~~
⋮

Algorithms.

What is a computer program, anyway?

Example.

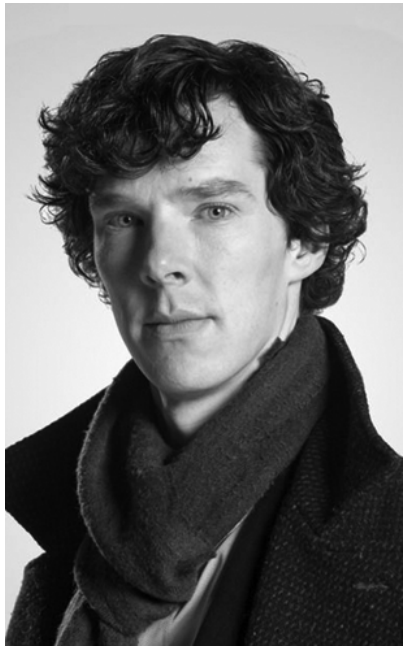
- Recall that a prime number is a whole number $p > 1$ whose only 'factors are 1 and p itself.

- Here is a program (algorithm) to find all prime numbers:

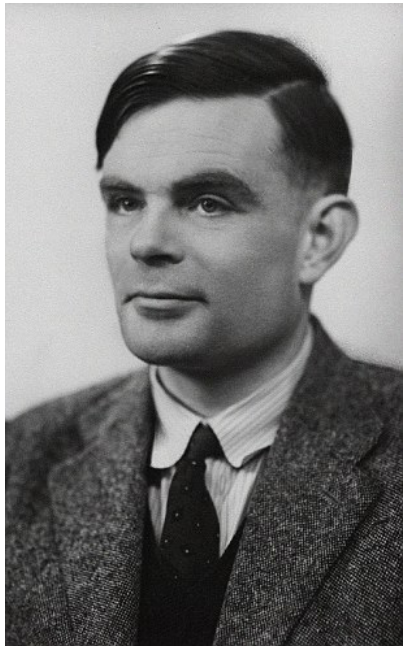
~~1~~ 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~
11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~
~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~
31 ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ 37 ~~38~~ ~~39~~ ~~40~~
⋮

- You can easily write this in your favorite language (C, Python, etc.).

Alan Turing.



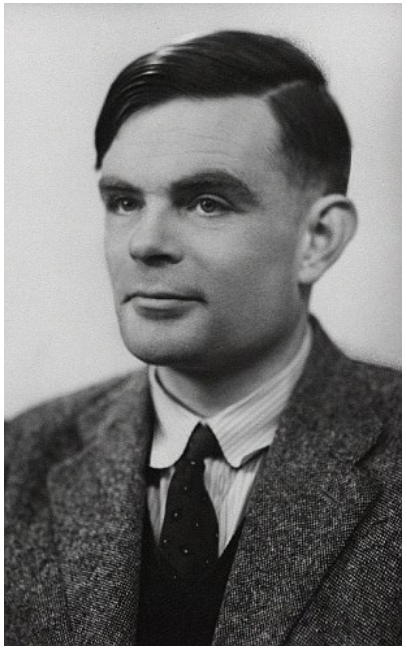
Alan Turing.



Alan Turing.

Born June 23, 1912.

British mathematician, computer scientist, and biologist.

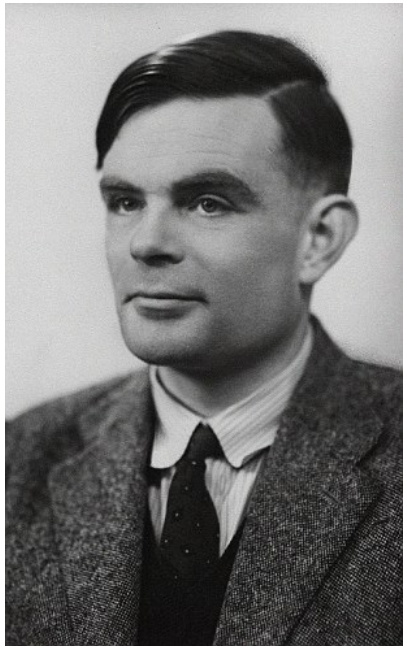


Alan Turing.

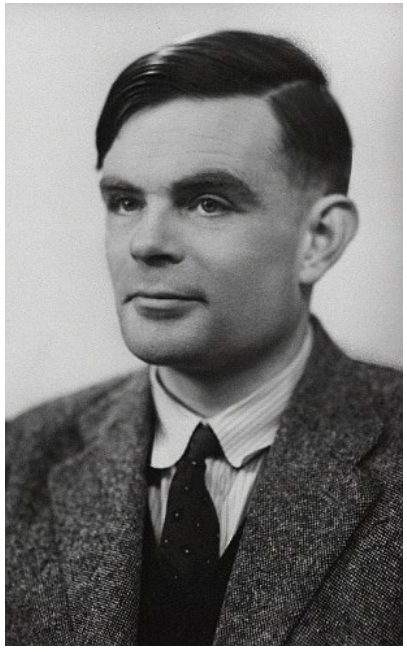
Born June 23, 1912.

British mathematician, computer scientist, and biologist.

Helped devise methods of breaking German encryption in WWII.



Alan Turing.



Born June 23, 1912.

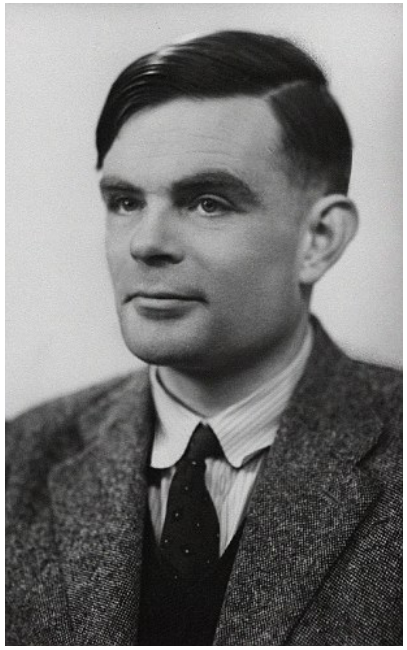
British mathematician, computer scientist, and biologist.

Helped devise methods of breaking German encryption in WWII.

Developed a **formal definition** of the concept of an algorithm via (what we today call) Turing machines.

- Resulted in development of computability theory and computer science. Led to the invention of the modern computer.

Alan Turing.



Born June 23, 1912.

British mathematician, computer scientist, and biologist.

Helped devise methods of breaking German encryption in WWII.

Developed a **formal definition** of the concept of an algorithm via (what we today call) Turing machines.

- Resulted in development of computability theory and computer science. Led to the invention of the modern computer.

Arrested for being a homosexual, and forced to accept agonizing hormone “therapy”, which led to his suicide in 1954.

Turing machines.

Turing's model of computation is incredibly elegant.

It was also absolutely revolutionary for its time.

Turing machines.

Turing's model of computation is incredibly elegant.

It was also absolutely revolutionary for its time.

A Turing machine consists of:

- A read/write head.

Turing machines.

Turing's model of computation is incredibly elegant.

It was also absolutely revolutionary for its time.

A Turing machine consists of:

- A read/write head.
- A tape on which to write/over-write 0s and 1s.

Turing machines.

Turing's model of computation is incredibly elegant.

It was also absolutely revolutionary for its time.

A Turing machine consists of:

- A read/write head.
- A tape on which to write/over-write 0s and 1s.
- The ability to move left and right along this tape.

Turing machines.

Turing's model of computation is incredibly elegant.

It was also absolutely revolutionary for its time.

A Turing machine consists of:

- A read/write head.
- A tape on which to write/over-write 0s and 1s.
- The ability to move left and right along this tape.
- A system of internal states.

Turing machines.

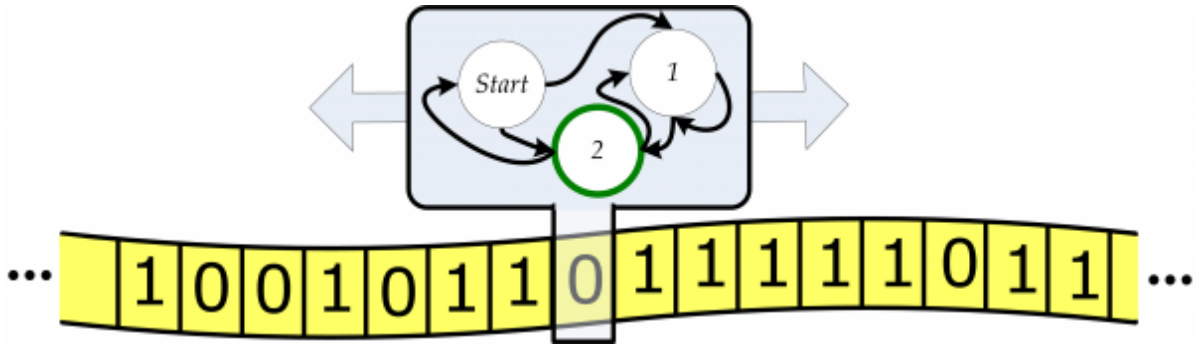
Turing's model of computation is incredibly elegant.

It was also absolutely revolutionary for its time.

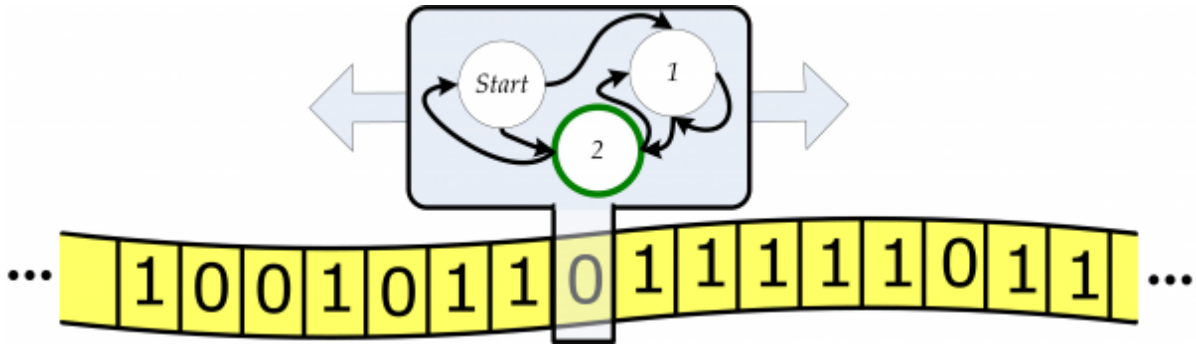
A Turing machine consists of:

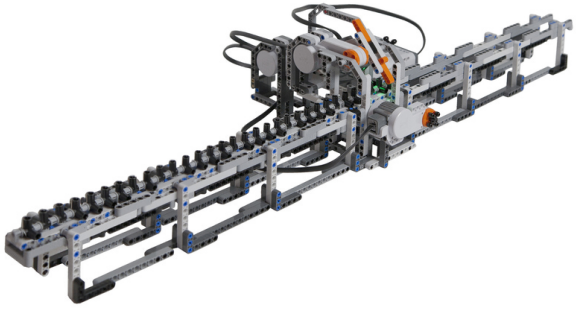
- A read/write head.
- A tape on which to write/over-write 0s and 1s.
- The ability to move left and right along this tape.
- A system of internal states.
- Instructions on what to do when in a given state, and reading a given symbol.

Turing machines (continued).

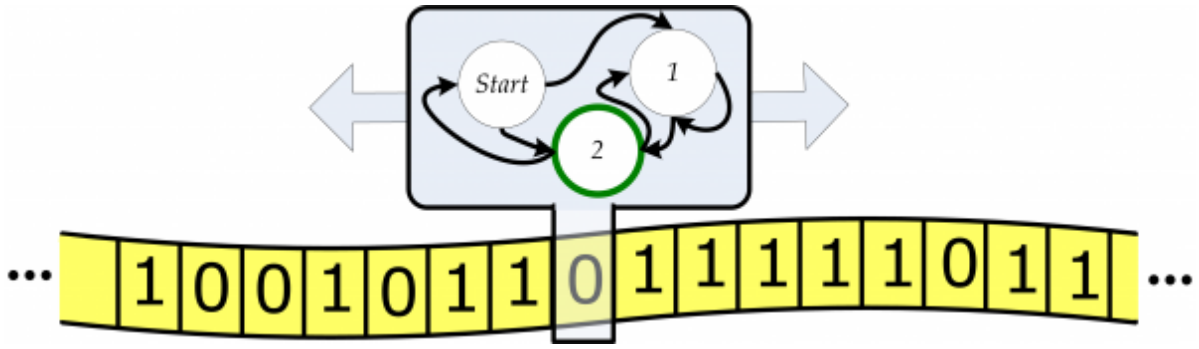


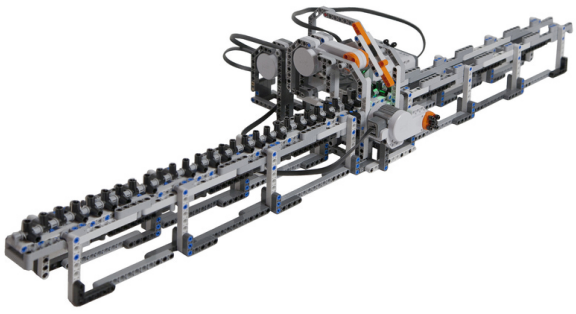
Turing machines (continued).



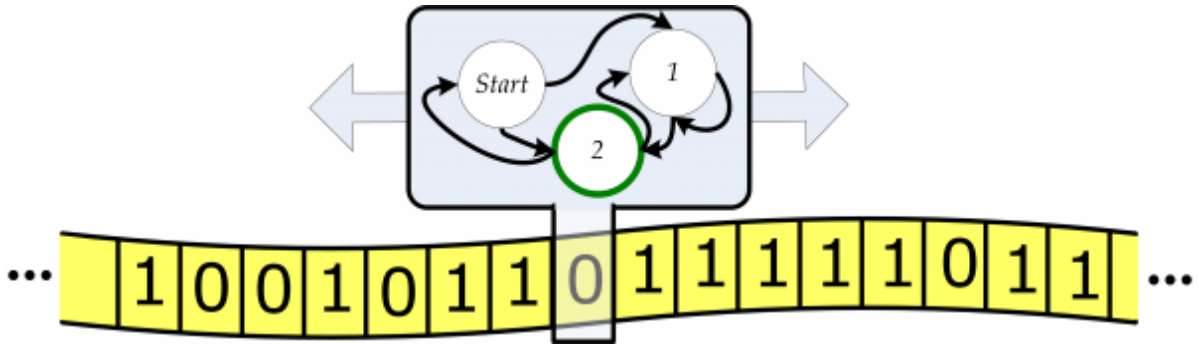


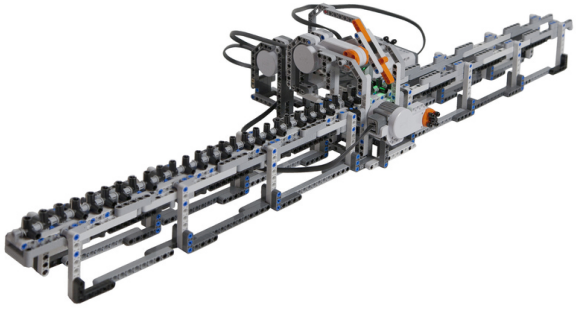
Turing machines (continued).





Turing machines (continued).





Universal TMs.

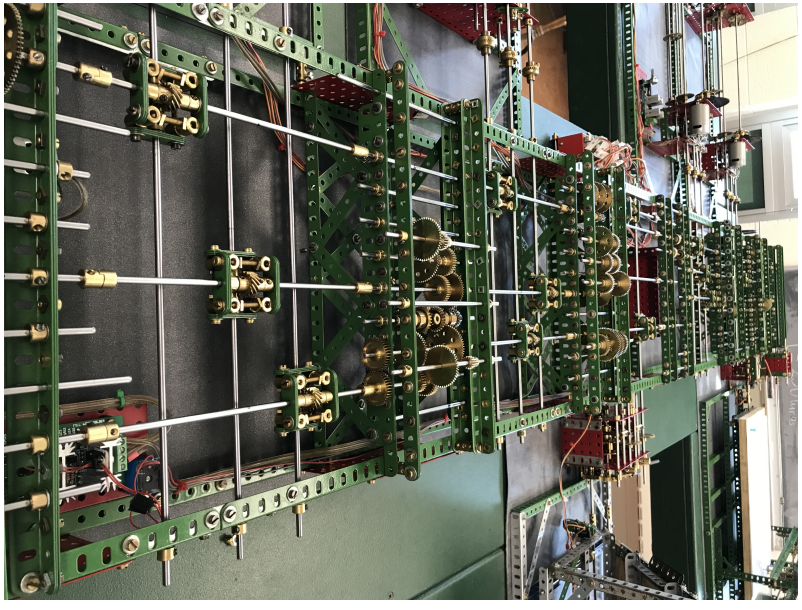
Key fact: there exist **universal Turing machines**. These can solve any problem any other Turing machine can solve (more on this later).

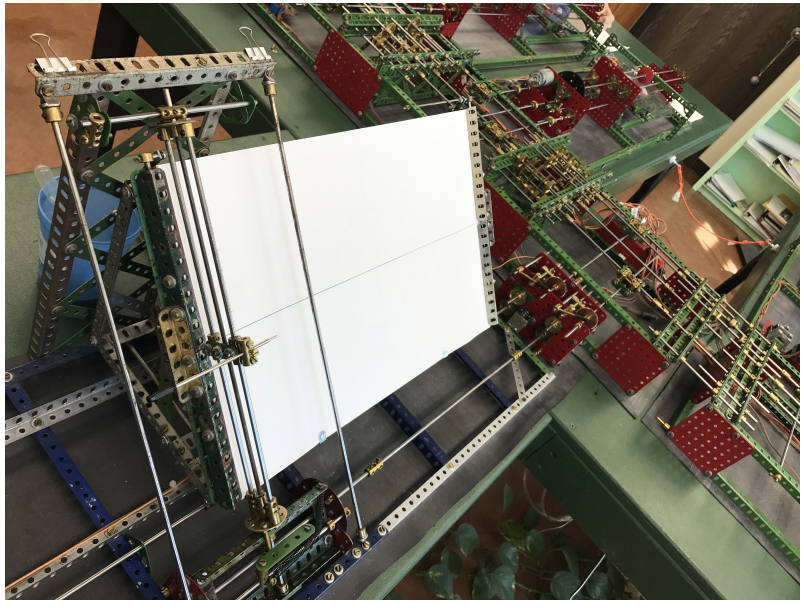
Not all Turing machines are universal Turing machines.

Universal TMs.

Key fact: there exist **universal Turing machines**. These can solve any problem any other Turing machine can solve (more on this later).

Not all Turing machines are universal Turing machines.





Mechanical Integrator/Differential Analyzer Lab at Marshall University.

A list of computer programs.

Imagine infinitely many monkeys, each typing randomly on a keyboard...

- Eventually, one of them write all of Shakespeare.

A list of computer programs.

Imagine infinitely many monkeys, each typing randomly on a keyboard...

- Eventually, one of them will write all of Shakespeare.
- And...one of them will also write all the Harry Potter books.

A list of computer programs.

Imagine infinitely many monkeys, each typing randomly on a keyboard...

- Eventually, one of them write all of Shakespeare.
- And...one of them will also write all the Harry Potter books.
- And one will write the source code for Pac-Man, and Angry Birds, and **every computer program** ever written, or that will be written.

A list of computer programs.

Imagine infinitely many monkeys, each typing randomly on a keyboard...

- Eventually, one of them write all of Shakespeare.
- And...one of them will also write all the Harry Potter books.
- And one will write the source code for Pac-Man, and Angry Birds, and **every computer program** ever written, or that will be written.

So there is a list $P_1, P_2, P_3, P_4, \ldots$ of all 'programs' written in this room.

A list of computer programs.

Imagine infinitely many monkeys, each typing randomly on a keyboard...

- Eventually, one of them write all of Shakespeare.
- And...one of them will also write all the Harry Potter books.
- And one will write the source code for Pac-Man, and Angry Birds, and **every computer program** ever written, or that will be written.

So there is a list $P_1, P_2, P_3, P_4, \ldots$ of all 'programs' written in this room.

- Angry Birds might be P_{208}, for example.

A list of computer programs.

Imagine infinitely many monkeys, each typing randomly on a keyboard...

- Eventually, one of them write all of Shakespeare.
- And...one of them will also write all the Harry Potter books.
- And one will write the source code for Pac-Man, and Angry Birds, and **every computer program** ever written, or that will be written.

So there is a list $P_1, P_2, P_3, P_4, \ldots$ of all 'programs' written in this room.

- Angry Birds might be P_{208}, for example.
- P_{637} might be " bv hjjhhjbiuillmoik8k,bj cfy".

The Halting Problem.

If we run P_{637}, the computer will be confused, and show us 

The Halting Problem.

If we run P_{637}, the computer will be confused, and show us 

If we run P_{208}, it will parse and compile the code, and play Angry Birds.

The Halting Problem.

If we run P_{637}, the computer will be confused, and show us 

If we run P_{208}, it will parse and compile the code, and play Angry Birds.

The technical term for this is that P_{208} halts, while P_{637} does not halt.

Problem.

Suppose we are given:

- A positive whole number k.

Figure out whether P_k halts or not.

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

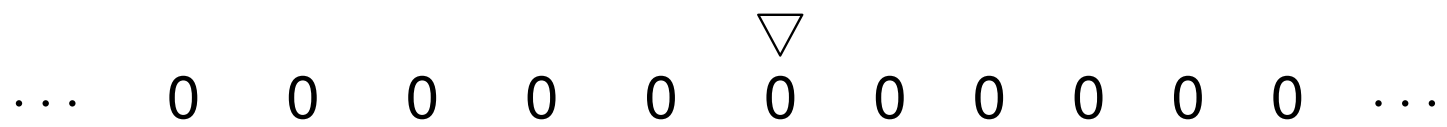
- Solve the **Addition Problem**. (You know, like $3 + 4$.)

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)



The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

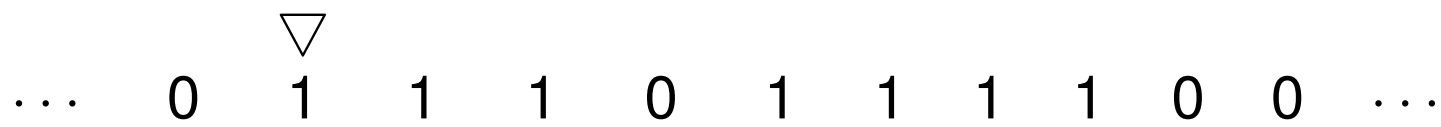
- Solve the **Addition Problem**. (You know, like $3 + 4$.)

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)



The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

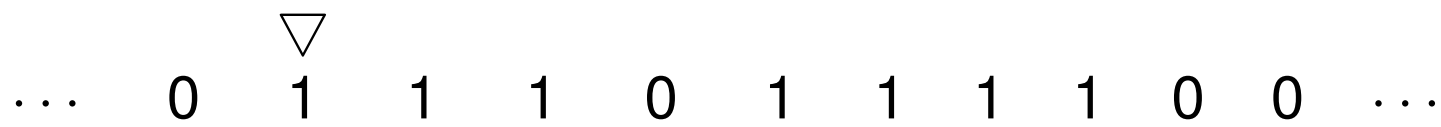
- Solve the **Addition Problem**. (You know, like $3 + 4$.)

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)



The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

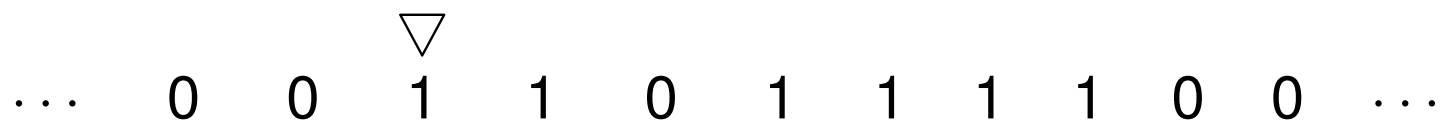
- Solve the **Addition Problem**. (You know, like $3 + 4$.)

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)

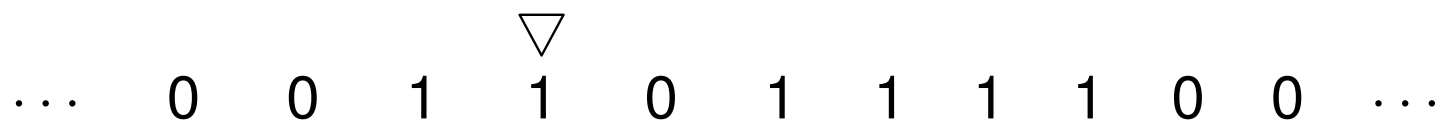


The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)



The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

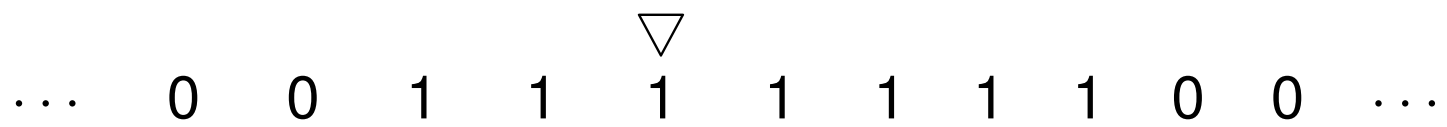
- Solve the **Addition Problem**. (You know, like $3 + 4$.)

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)

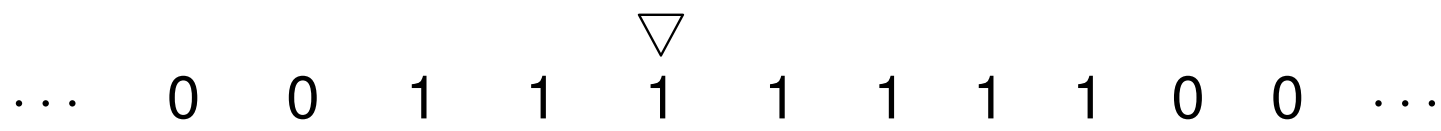


The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)



Theorem (Turing, 1936).

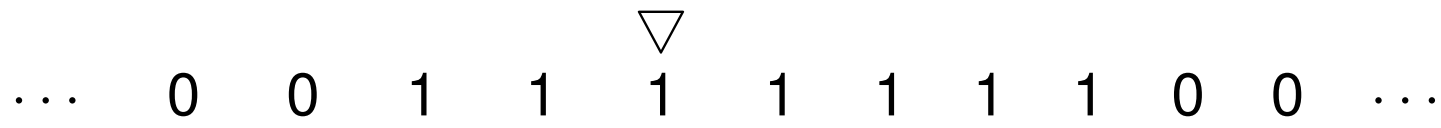
There is no Turing machine that can solve the halting problem.

The Halting Problem (continued).

How does a Turing machine **solve** a problem?

Example.

- Solve the **Addition Problem**. (You know, like $3 + 4$.)



Theorem (Turing, 1936).

There is no Turing machine that can solve the halting problem.

(Sorry, app developers...)

The Range Problem as a computer program.

Recall the range problem: We are given a function F and a number, say 6, and we need to know if 12 is in the range of F.

The Range Problem as a computer program.

Recall the range problem: We are given a function F and a number, say 6, and we need to know if 12 is in the range of F.

Write a program to check if $F(1) = 6$, $F(2) = 6$, $F(3) = 6$, $F(4) = 6$, ..., which keeps going forever unless it finds some n with $F(n) = 6$.

The Range Problem as a computer program.

Recall the range problem: We are given a function F and a number, say 6, and we need to know if 12 is in the range of F.

Write a program to check if $F(1) = 6$, $F(2) = 6$, $F(3) = 6$, $F(4) = 6$, ..., which keeps going forever unless it finds some n with $F(n) = 6$.

```
|   program "check if 6 in range of F"
|   set n to be 1
|   while 6 is not equal to F(n)
|     set n = n + 1
|   end
```

The Range Problem as a computer program.

Recall the range problem: We are given a function F and a number, say 6, and we need to know if 12 is in the range of F.

Write a program to check if $F(1) = 6$, $F(2) = 6$, $F(3) = 6$, $F(4) = 6$, ..., which keeps going forever unless it finds some n with $F(n) = 6$.

```
|   program "check if 6 in range of F"
|   set n to be 1
|   while 6 is not equal to F(n)
|     set n = n + 1
|   end
```

This program is somewhere on our list, say $P_{3817834}$.

Now $P_{3817834}$ halts if and only if 6 is in the range of F.

A little math: coding numbers.

Let's call a number a coding number if it is equal to some number of **powers of** 2 times some number of **powers of** 3.

A little math: coding numbers.

Let's call a number a coding number if it is equal to some number of **powers of** 2 times some number of **powers of** 3.

Examples.

- 6 is a coding number, since it is equal to $2 \cdot 3$.
- So is $12 = 2^2 \cdot 3$, and $18 = 2 \cdot 3^2$, and $2^5 \cdot 3^3$, etc.

A little math: coding numbers.

Let's call a number a coding number if it is equal to some number of **powers of** 2 times some number of **powers of** 3.

Examples.

- 6 is a coding number, since it is equal to $2 \cdot 3$.
- So is $12 = 2^2 \cdot 3$, and $18 = 2 \cdot 3^2$, and $2^5 \cdot 3^3$, etc.

Non-examples.

- 10 is a not coding number, since it is equal to $2 \cdot 5$.
- Neither is $15 = 3 \cdot 5$, or $28 = 2^2 \cdot 7$, etc.

The Halting Problem as the range of a function.

Recall the halting problem: Given k, determine if P_k halts.

The Halting Problem as the range of a function.

Recall the halting problem: Given k, determine if P_k halts.

Define a function F as follows.

- Given a number n, first check if n is a coding number.
- If n is not a coding number, define $F(n) = 0$.
- If n is a coding number, say $n = 2^k 3^t$. Run P_k for t minutes.
- If P_k halts in t minutes, let $F(2^k 3^t) = k$.
- If P_k does not halt in t minutes, let $F(2^k 3^t) = 0$.

The Halting Problem as the range of a function.

Recall the halting problem: Given k, determine if P_k halts.

Define a function F as follows.

- Given a number n, first check if n is a coding number.
- If n is not a coding number, define $F(n) = 0$.
- If n is a coding number, say $n = 2^k 3^t$. Run P_k for t minutes.
- If P_k halts in t minutes, let $F(2^k 3^t) = k$.
- If P_k does not halt in t minutes, let $F(2^k 3^t) = 0$.

Now P_k halts if and only if k is in the range of F.

Application.

A famous mathematical problem called the **Goldbach Conjecture** states that every even number $n > 2$ is the sum of two prime numbers.

Application.

A famous mathematical problem called the **Goldbach Conjecture** states that every even number $n > 2$ is the sum of two prime numbers.

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

Application.

A famous mathematical problem called the **Goldbach Conjecture** states that every even number $n > 2$ is the sum of two prime numbers.

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

We have no idea if the Goldbach Conjecture is true or false!

Application.

A famous mathematical problem called the **Goldbach Conjecture** states that every even number $n > 2$ is the sum of two prime numbers.

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

We have no idea if the Goldbach Conjecture is true or false!

Define a function F as follows: for each number $k > 0$, $F(k) = 0$ if every even $n < k$ is the sum of two primes, and $F(k) = 1$ otherwise.

So 1 is in the range of F if and only if the Goldbach Conjecture is false.

Application.

A famous mathematical problem called the **Goldbach Conjecture** states that every even number $n > 2$ is the sum of two prime numbers.

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

We have no idea if the Goldbach Conjecture is true or false!

Define a function F as follows: for each number $k > 0$, $F(k) = 0$ if every even $n < k$ is the sum of two primes, and $F(k) = 1$ otherwise.

So 1 is in the range of F if and only if the Goldbach Conjecture is false.

This means that if we could solve the range problem (or equivalently, the halting problem), then we could figure out whether or not the Goldbach Conjecture is true—and I'd be out of a job!

Thanks for your attention!