The Busy Beaver Problem

Esther Fu and Sarah Pan
(Mentor: Alexandra Hoey)

PRIMES Circle

MIT

May 21, 2022

Computers are great!

Why do we love computers?

Computers are great!

Why do we love computers?

Maybe they can help resolve long-standing mathematical questions!

A Few Problems

The Riemann Hypothesis (1859)

The Riemann zeta function has zeros only at negative even integers and complex numbers with real part $\frac{1}{2}$.

A Few Problems

The Riemann Hypothesis (1859)

The Riemann zeta function has zeros only at negative even integers and complex numbers with real part $\frac{1}{2}$.

Goldbach's conjecture (1742)

Every even integer greater than two can be expressed as the sum of two primes.

A Problem in Particular

Goldbach's conjecture (1742)

Every even integer greater than two can be expressed as the sum of two primes.

A Problem in Particular

Goldbach's conjecture (1742)

Every even integer greater than two can be expressed as the sum of two primes.

Example

$$10 = 3 + 7$$

A Problem in Particular

Goldbach's conjecture (1742)

Every even integer greater than two can be expressed as the sum of two primes.

Example

$$10 = 3 + 7$$
$$148 = 101 + 47$$

A Problem in Particular

Goldbach's conjecture (1742)

Every even integer greater than two can be expressed as the sum of two primes.

Example

$$10 = 3 + 7$$
$$148 = 101 + 47$$
$$4390 = 1091 + 3299$$

A possible solution to Goldbach's conjecture?

Tadaa!!

```
000(023Rb|001Rb)
001(017La|002Rb)
002(021La|003Rb)
003(021La|004La)
004(009Rb|005Lb)
005(004Ra|005La)
006(008La|007La)
007(009Rb|007La)
008(009Ra|008La)
009(010Ra|026Ra)
010(010Rb|011Ra)
011(012Rb|011Rb)
012(014La|013La)
013(006Lb|013Lb)
014(015La|014Lb)
015(016Rb|019Lb)
016(017Lb|ERR--)
017(018Lb|017Lb)
018(009Ra|025La)
019(020Rb|019Lb)
020(002Rb|020Rb)
021(022La|021Lb)
022(000Ra|024Lb)
023(024Lb|023Rb)
024(000Ra|024Lb)
025(HALT-|024Rb)
026(018Lb|026Rb)
```

A possible solution to Goldbach's conjecture?

Tadaa!!

```
000(023Rb|001Rb)
001(017La|002Rb)
002(021La|003Rb)
003(021La|004La)
004(009Rb|005Lb)
005(004Ra|005La)
006(008La|007La)
007(009Rb|007La)
008(009Ra|008La)
009(010Ra|026Ra)
010(010Rb|011Ra)
011(012Rb|011Rb)
012(014La|013La)
013(006Lb|013Lb)
014(015La|014Lb)
015(016Rb|019Lb)
016(017Lb|ERR--)
017(018Lb|017Lb)
018(009Ra|025La)
019(020Rb|019Lb)
020(002Rb|020Rb)
021(022La|021Lb)
022(000Ra|024Lb)
023(024Lb|023Rb)
024(000Ra|024Lb)
025(HALT-|024Rb)
026(018Lb|026Rb)
```

Except it runs for a very long time...

A possible solution to Goldbach's conjecture?

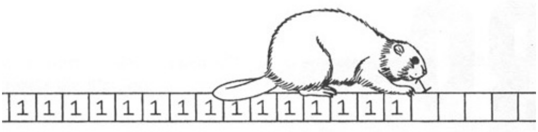
Tadaa!!

```
000(023Rb|001Rb)
001(017La|002Rb)
002(021La|003Rb)
003(021La|004La)
004(009Rb|005Lb)
005(004Ra|005La)
006(008La|007La)
007(009Rb|007La)
008(009Ra|008La)
009(010Ra|026Ra)
010(010Rb|011Ra)
011(012Rb|011Rb)
012(014La|013La)
013(006Lb|013Lb)
014(015La|014Lb)
015(016Rb|019Lb)
016(017Lb|ERR--)
017(018Lb|017Lb)
018(009Ra|025La)
019(020Rb|019Lb)
020(002Rb|020Rb)
021(022La|021Lb)
022(000Ra|024Lb)
023(024Lb|023Rb)
024(000Ra|024Lb)
025(HALT-|024Rb)
026(018Lb|026Rb)
```

Except it runs for a very long time...

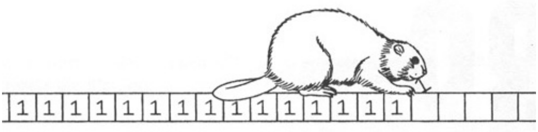
How do we know if the program will stop running?

The Busy Beaver Problem



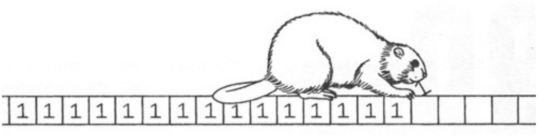
- The busy beaver problem was introduced by mathematician Tibor Radó in 1962.

The Busy Beaver Problem



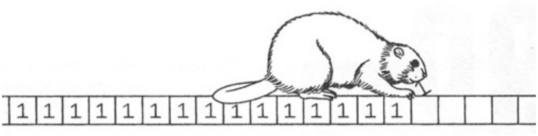
- The busy beaver problem was introduced by mathematician Tibor Radó in 1962.
- Objective: find the "busiest" algorithm of a given size.

The Busy Beaver Problem



- The busy beaver problem was introduced by mathematician Tibor Radó in 1962.
- Objective: find the "busiest" algorithm of a given size.
- Using this bound, we can figure out whether programs loop indefinitely!

The Busy Beaver Problem



- The busy beaver problem was introduced by mathematician Tibor Radó in 1962.
- Objective: find the “busiest” algorithm of a given size.
- Using this bound, we can figure out whether programs loop indefinitely!
- Except we can’t compute these upper bounds :(

Table of Contents

Goldbach Algorithm

Recall:

```
000(023Rb|001Rb)
001(017La|002Rb)
002(021La|003Rb)
003(021La|004La)
004(009Rb|005Lb)
005(004Ra|005La)
006(008La|007La)
007(009Rb|007La)
008(009Ra|008La)
009(010Ra|026Ra)
010(010Rb|011Ra)
011(012Rb|011Rb)
012(014La|013La)
013(006Lb|013Lb)
014(015La|014Lb)
015(016Rb|019Lb)
016(017Lb|ERR--)
017(018Lb|017Lb)
018(009Ra|025La)
019(020Rb|019Lb)
020(002Rb|020Rb)
021(022La|021Lb)
022(000Ra|024Lb)
023(024Lb|023Rb)
024(000Ra|024Lb)
025(HALT-|024Rb)
026(018Lb|026Rb)
```

Goldbach Algorithm

Recall:

```
000(023Rb|001Rb)
001(017La|002Rb)
002(021La|003Rb)
003(021La|004La)
004(009Rb|005Lb)
005(004Ra|005La)
006(008La|007La)
007(009Rb|007La)
008(009Ra|008La)
009(010Ra|026Ra)
010(010Rb|011Ra)
011(012Rb|011Rb)
012(014La|013La)
013(006Lb|013Lb)
014(015La|014Lb)
015(016Rb|019Lb)
016(017Lb|ERR--)
017(018Lb|017Lb)
018(009Ra|025La)
019(020Rb|019Lb)
020(002Rb|020Rb)
021(022La|021Lb)
022(000Ra|024Lb)
023(024Lb|023Rb)
024(000Ra|024Lb)
025(HALT-|024Rb)
026(018Lb|026Rb)
```

What is this?

Goldbach Algorithm

Answer

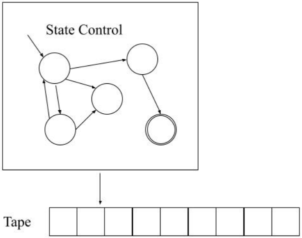
Goldbach Algorithm

Answer

It's the description of a Turing machine!

Turing Machines

Turing machines are a theoretical model of computation that behave similarly to modern computers.



Turing machine schematic

Languages

- Turing machines recognize whether its input belongs in a language.

Languages

- Turing machines recognize whether its input belongs in a language.

Definition

A language is a set of strings that usually follow a certain rule.

Languages

- Turing machines recognize whether its input belongs in a language.

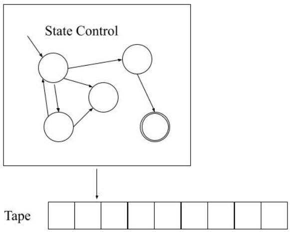
Definition

A language is a set of strings that usually follow a certain rule.

Example

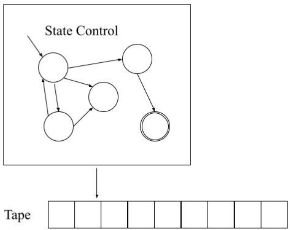
$L = \{w | w \in \{0, 1\}\}$

Turing Machines Cont.



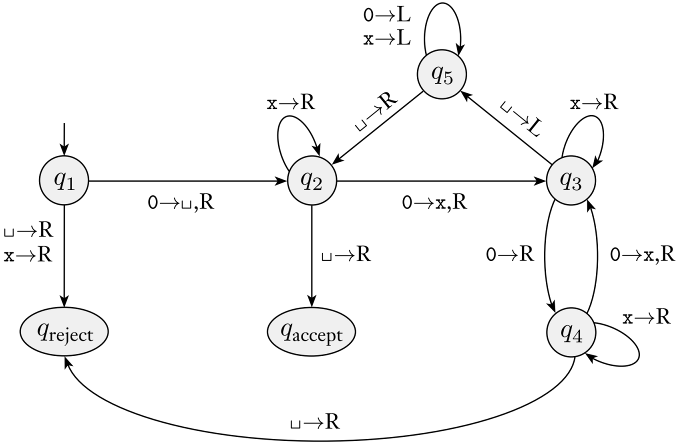
- Turing machines are finite state machines with infinite memory (the tape).

Turing Machines Cont.



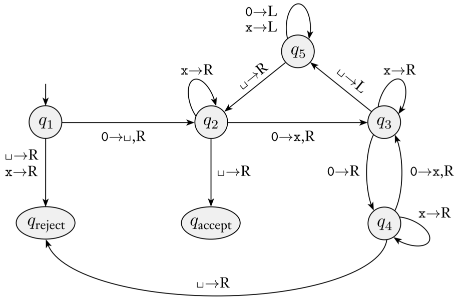
- Turing machines are finite state machines with infinite memory (the tape).
- Most importantly, the Church-Turing Thesis states that all algorithms map to a corresponding Turing machine

Turing Machine Example



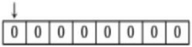
State diagram of Turing machine

Turing Machine Example

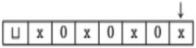


State diagram for a TM that recognizes $A = \{0^{2^n} | n \geq 0\}$

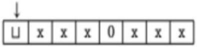
1



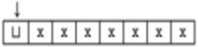
2



3



4



Decidability

Definition

Language A is decidable if there exists some Turing machine M such that

$$M \text{ accepts all } s \in A \text{ and rejects all } s \notin A.$$

Decidability

Definition

Language A is decidable if there exists some Turing machine M such that

$$M \text{ accepts all } s \in A \text{ and rejects all } s \notin A.$$

Corollary

If a language is undecidable, there is no algorithm that decides whether $s \in A$.

Two Undecidable Languages

Two Undecidable Languages

The first one:

$$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$$

Two Undecidable Languages

The first one:

$$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$$

The second:

$$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$$

Proving Undecidability Through Diagonalization

Recall: $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$

Theorem

The language A_{TM} is undecidable.

Proof.

Assume to the contary that TM H decides A_{TM}.

Proving Undecidability Through Diagonalization

Recall: $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$

Theorem

The language A_{TM} is undecidable.

Proof.

Assume to the contary that TM H decides A_{TM}.

- Create TM D as follows:

Proving Undecidability Through Diagonalization

Recall: $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$

Theorem

The language A_{TM} is undecidable.

Proof.

Assume to the contary that TM H decides A_{TM}.

- Create TM D as follows:
  - It runs H on $\langle M, \langle M\rangle\rangle$.

Proving Undecidability Through Diagonalization

Recall: $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$

Theorem

The language A_{TM} is undecidable.

Proof.

Assume to the contary that TM H decides A_{TM}.

- Create TM D as follows:
  - It runs H on $\langle M, \langle M\rangle\rangle$.
  - If H accepts, *reject*.

Proving Undecidability Through Diagonalization

Recall: $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$

Theorem

The language A_{TM} is undecidable.

Proof.

Assume to the contary that TM H decides A_{TM}.

- Create TM D as follows:
  - It runs H on $\langle M, \langle M\rangle\rangle$.
  - If H accepts, *reject*.
  - If H rejects, *accept*.

□

Proving Undecidability Through Diagonalization

To visualize H (a decider for A_{TM})...

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...	$\langle D \rangle$
M1	accept	reject	accept	...	accept
M2	reject	reject	reject	...	reject
M3	accept	accept	accept	...	accept
...	...	...	...	...	...
D	reject	accept	reject	...	accept

Proving Undecidability Through Diagonalization

To visualize H (a decider for A_{TM})...

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...	$\langle D \rangle$
M1	**accept**	reject	accept	...	accept
M2	reject	**reject**	reject	...	reject
M3	accept	accept	**accept**	...	accept
...	...	...	...	...	...
D	reject	accept	reject	...	**accept**

Proving Undecidability Through Diagonalization

To visualize H (a decider for A_{TM})...

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...	$\langle D \rangle$
M1	**accept**	reject	accept	...	accept
M2	reject	**reject**	reject	...	reject
M3	accept	accept	**accept**	...	accept
...	...	...	...	...	...
D	reject	accept	reject	...	**accept** ?

Proving Undecidability Through Diagonalization

To visualize H (a decider for A_{TM})...

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...	$\langle D \rangle$
M1	**accept**	reject	accept	...	accept
M2	reject	**reject**	reject	...	reject
M3	accept	accept	**accept**	...	accept
...	...	...	...	...	...
D	reject	accept	reject	...	**accept** ?

Turing machine D accepts $\langle D \rangle$ if and only if D rejects $\langle D \rangle$.

Proving Undecidability Through Diagonalization

To visualize H (a decider for A_{TM})...

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...	$\langle D \rangle$
M1	**accept**	reject	accept	...	accept
M2	reject	**reject**	reject	...	reject
M3	accept	accept	**accept**	...	accept
...	...	...	...	...	...
D	reject	accept	reject	...	**accept** ?

Turing machine D accepts $\langle D \rangle$ if and only if D rejects $\langle D \rangle$.
There is no such TM that decides A_{TM}.

$HALT_{TM}$

Recall:

$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}.$

$HALT_{TM}$

Recall:

$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$.

Let's take advantage of its similarity with A_{TM}.

Reducibility

Example

Is it possible for me to go to the beach tomorrow?

Reducibility

Example

Is it possible for me to go to the beach tomorrow?
Reduces to: Do I have a ride?

Reducing from A_{TM} to $HALT_{TM}$

Recall:
$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$,
$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$.

Reducing from A_{TM} to $HALT_{TM}$

Recall:
$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$,
$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$.

Theorem

The language $HALT_{TM}$ is undecidable.

Reducing from A_{TM} to $HALT_{TM}$

Recall:
$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$,
$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$.

Theorem

The language $HALT_{TM}$ is undecidable.

Proof. Assume to the contrary that TM S decides $HALT_{TM}$.

Reducing from A_{TM} to $HALT_{TM}$

Recall:
$HALT_{TM} = \{\langle M, w\rangle | M$ is a TM and M halts on input $w\}$,
$A_{TM} = \{\langle M, w\rangle | M$ is a TM and M accepts $w\}$.

Theorem

The language $HALT_{TM}$ is undecidable.

Proof. Assume to the contrary that TM S decides $HALT_{TM}$.

- Create TM H that takes in $\langle M, w\rangle$ as follows:

Reducing from A_{TM} to $HALT_{TM}$

Recall:
$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$,
$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$.

Theorem

The language $HALT_{TM}$ is undecidable.

Proof. Assume to the contrary that TM S decides $HALT_{TM}$.

- Create TM H that takes in $\langle M, w\rangle$ as follows:
  - It runs S on $\langle M, w\rangle$.

Reducing from A_{TM} to $HALT_{TM}$

Recall:
$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$,
$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$.

Theorem

The language $HALT_{TM}$ is undecidable.

Proof. Assume to the contrary that TM S decides $HALT_{TM}$.

- Create TM H that takes in $\langle M, w\rangle$ as follows:
  - It runs S on $\langle M, w\rangle$.
  - If S rejects, *reject*.

Reducing from A_{TM} to $HALT_{TM}$

Recall:
$HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$,
$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$.

Theorem

The language $HALT_{TM}$ is undecidable.

Proof. Assume to the contrary that TM S decides $HALT_{TM}$.

- Create TM H that takes in $\langle M, w\rangle$ as follows:
  - It runs S on $\langle M, w\rangle$.
  - If S rejects, *reject*.
  - If S accepts, run M on w. Do what M does.

Reducing from A_{TM} to $HALT_{TM}$

Recall:
$HALT_{TM} = \{\langle M, w\rangle | M$ is a TM and M halts on input $w\}$,
$A_{TM} = \{\langle M, w\rangle | M$ is a TM and M accepts $w\}$.

Theorem

The language $HALT_{TM}$ is undecidable.

Proof. Assume to the contrary that TM S decides $HALT_{TM}$.

- Create TM H that takes in $\langle M, w\rangle$ as follows:
  - It runs S on $\langle M, w\rangle$.
  - If S rejects, *reject*.
  - If S accepts, run M on w. Do what M does.

Because $HALT_{TM}$ reduces to A_{TM}, it is also undecidable.

The Busy Beaver Game

To reiterate: the busy beaver problem

Find the maximum number of computations a halting algorithm with a given size can perform.

The Busy Beaver Game

To reiterate: the busy beaver problem

Find the maximum number of computations a halting algorithm with a given size can perform.

But more specifically: the busy beaver problem

Find the maximum the number of computations a halting ~~algorithm~~ **Turing machine** with a given ~~size~~ **number of states** can perform.

The Busy Beaver Game cont.

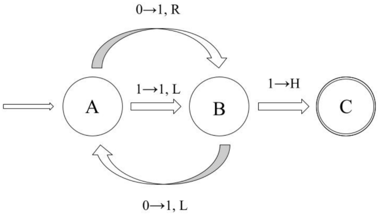
Our metric for *number of computations* is the number of state shifts a Turing machine undergoes.

Definition

$BB(n) :=$ the maximum number of shifts for a Turing machine with n non-halt states.

Example: $BB(2) = 6$

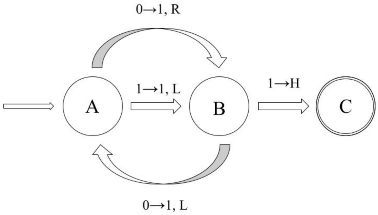
Modification: this Turing machine has an all-encompassing halt state.



0	0	0	0	0	0	0	0	0

Example: $BB(2) = 6$

Modification: this Turing machine has an all-encompassing halt state.

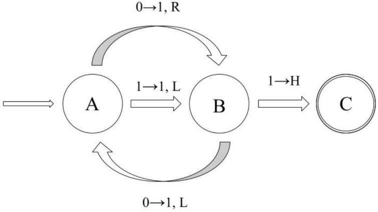


				↓				
0	0	0	0	0	0	0	0	0

					↓			
0	0	0	0	1	0	0	0	0

Example: $BB(2) = 6$

Modification: this Turing machine has an all-encompassing halt state.



0	0	0	0	0	0	0	0	0

0	0	0	0	1	0	0	0	0

0	0	0	0	1	1	0	0	0

0	0	0	0	1	1	0	0	0

0	0	0	1	1	1	0	0	0

0	0	1	1	1	1	0	0	0

0	0	1	1	1	1	0	0	0

Busy Beaver Numbers

A few busy beaver numbers:

- $BB(1) = 1$

Busy Beaver Numbers

A few busy beaver numbers:

- $BB(1) = 1$
- $BB(2) = 6$

Busy Beaver Numbers

A few busy beaver numbers:

- $BB(1) = 1$
- $BB(2) = 6$
- $BB(3) = 21$

Busy Beaver Numbers

A few busy beaver numbers:

- $BB(1) = 1$
- $BB(2) = 6$
- $BB(3) = 21$
- $BB(4) = 107$

Busy Beaver Numbers

A few busy beaver numbers:

- $BB(1) = 1$
- $BB(2) = 6$
- $BB(3) = 21$
- $BB(4) = 107$
- $BB(5) \stackrel{?}{=} 47\,176\,870$

Busy Beaver Numbers

A few busy beaver numbers:

- $BB(1) = 1$
- $BB(2) = 6$
- $BB(3) = 21$
- $BB(4) = 107$
- $BB(5) \stackrel{?}{=} 47\,176\,870$
- $BB(6) \geq 7.4 \times 10^{36534}$

Busy Beaver Numbers

A few busy beaver numbers:

- $BB(1) = 1$
- $BB(2) = 6$
- $BB(3) = 21$
- $BB(4) = 107$
- $BB(5) \stackrel{?}{=} 47\,176\,870$
- $BB(6) \geq 7.4 \times 10^{36534}$
- $BB(7) \geq 10^{10^{10^{10^{18705353}}}}$

An exaFLOPS computer system can perform $\sim 10^{18}$ floating-point operations per second.

Why do Busy Beaver numbers matter?

Theorem (Matiyasevich, O'Rear 2016)

There is a 744-state Turing machine that halts if and only if the Riemann Hypothesis is false.

Why do Busy Beaver numbers matter?

Theorem (Matiyasevich, O'Rear 2016)

There is a 744-state Turing machine that halts if and only if the Riemann Hypothesis is false.

Theorem (Anonymous GitHub user 2015)

There is a 27-state Turing machine that halts if and only if Goldbach's conjecture is false.

Acknowledgements

First and foremost, we would like to thank to our phenomenal, cool, intelligent mentor Alexandra!
Thanks to the amazing Primes Circle coordinators Marisa and Mary!
Thanks to Sarah's computer science teacher, Tom, Michael Sipser for writing such an informative book, and Scott Aaronson for being a g.
And thank you all for coming to our presentation!

References

- Aaronson, Scott. (2020). *The Busy Beaver Frontier* https://www.scottaaronson.com/papers/bb.pdf
- Kun, Jeremy. (2012). *Busy Beavers, and Quest for Big Numbers* https://jeremykun.com/2012/02/08/busy-beavers-and-the-quest-for-big-numbers/
- Mullins, Robert. (2012). *What is a Turing machine?* [illustration] https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/turing-machine/one.html
- Sipser, M. (2013). *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning.