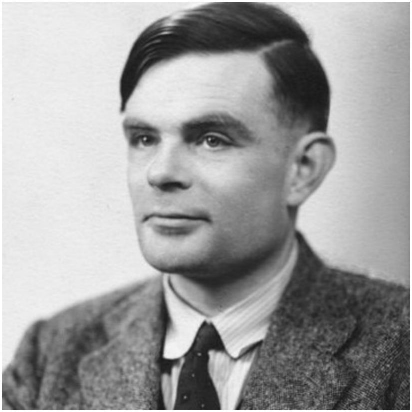
Introduction to Theory of Computation

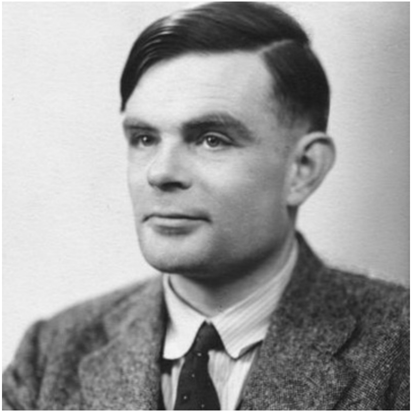
Chapters 4 and 5, Turing Machines and Decidability

November 28, 2018

Alan Turing

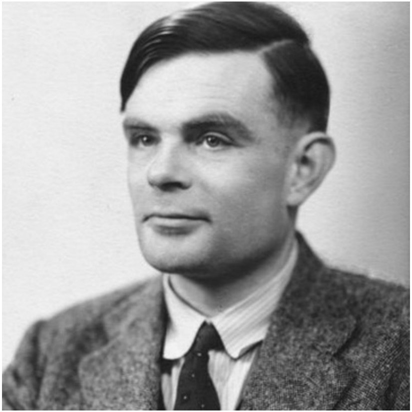


Alan Turing



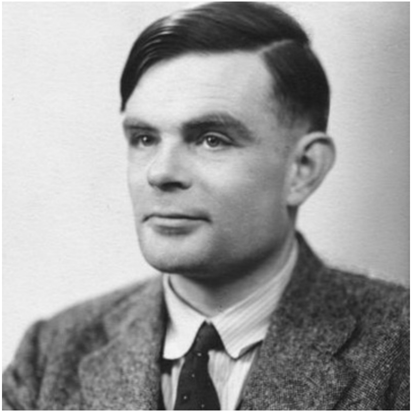
- Defeated Nazi Enigma code machine in WWII.

Alan Turing



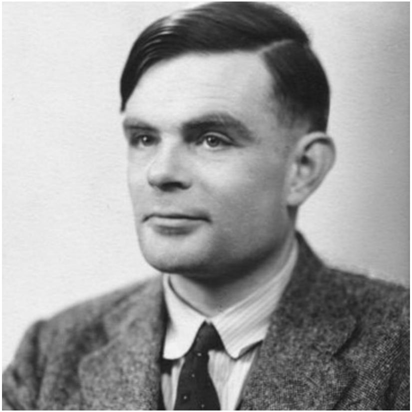
- Defeated Nazi Enigma code machine in WWII.
- Designed and built one of the first computers.

Alan Turing



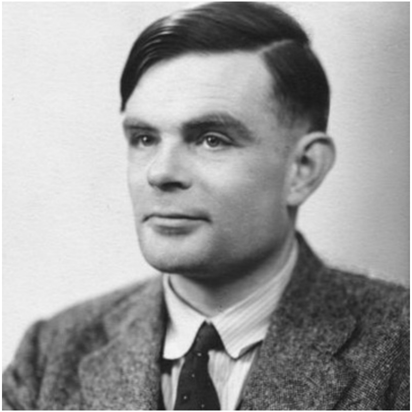
- Defeated Nazi Enigma code machine in WWII.
- Designed and built one of the first computers.
- Developed philosophy of artificial intelligence.

Alan Turing



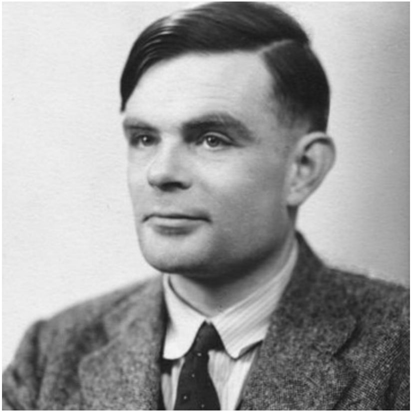
- Defeated Nazi Enigma code machine in WWII.
- Designed and built one of the first computers.
- Developed philosophy of artificial intelligence.
- Developed theory of computability.

Alan Turing



- Defeated Nazi Enigma code machine in WWII.
- Designed and built one of the first computers.
- Developed philosophy of artificial intelligence.
- Developed theory of computability.
- Invented roundhouse chess.

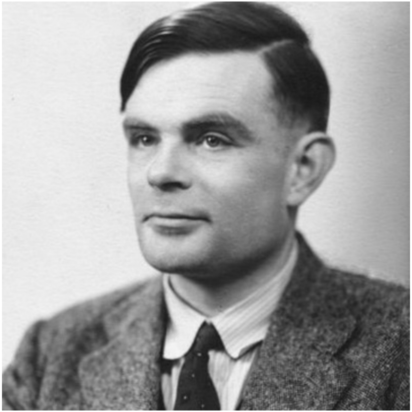
Alan Turing



- Defeated Nazi Enigma code machine in WWII.
- Designed and built one of the first computers.
- Developed philosophy of artificial intelligence.
- Developed theory of computability.
- Invented roundhouse chess.

- Convicted of homosexuality in 1952.

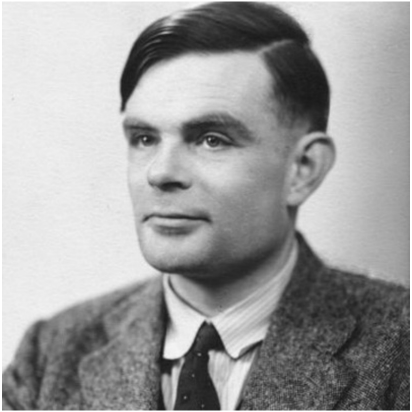
Alan Turing



- Defeated Nazi Enigma code machine in WWII.
- Designed and built one of the first computers.
- Developed philosophy of artificial intelligence.
- Developed theory of computability.
- Invented roundhouse chess.

- Convicted of homosexuality in 1952.
- Sentenced to chemical castration by synthetic estrogen.

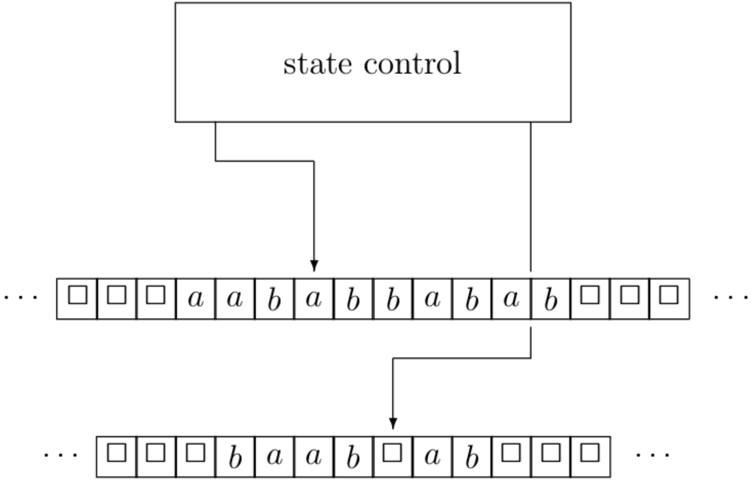
Alan Turing



- Defeated Nazi Enigma code machine in WWII.
- Designed and built one of the first computers.
- Developed philosophy of artificial intelligence.
- Developed theory of computability.
- Invented roundhouse chess.

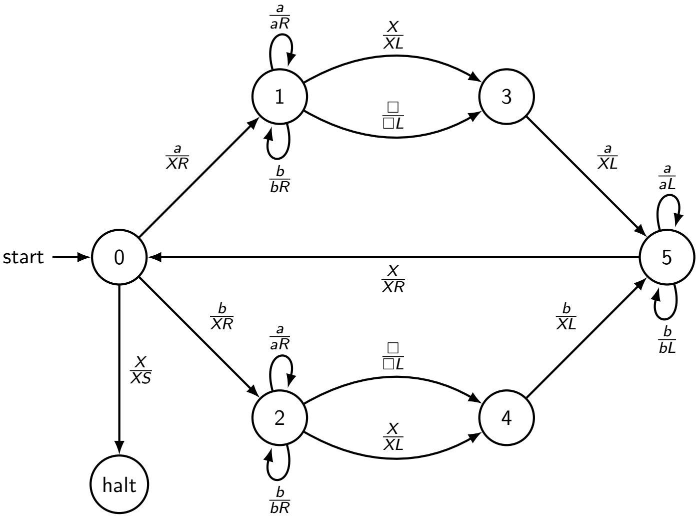
- Convicted of homosexuality in 1952.
- Sentenced to chemical castration by synthetic estrogen.
- Committed suicide in 1954, 16 days before 42nd birthday.

Turing Machine

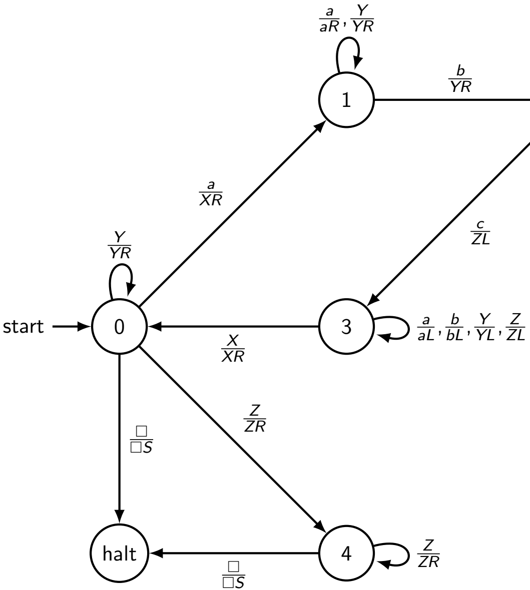


Tape Drives

Even Palindromes



$a^n b^n c^n$


start
0
1
2
3
4
halt
a/aR, Y/YR
b/bR, Z/ZR
b/YR
a/XR
c/ZL
Y/YR
a/aL, b/bL, Y/YL, Z/ZL
X/XR
Z/ZR
□/□S
Z/ZR
□/□S

Equivalent models:

1. One-tape Turing machines.
2. k-tape Turing machines.
3. Non-deterministic Turing machines.
4. Java programs.
5. Scheme programs.
6. `C++` programs.
7. ...

A Universal Turing Machine

- Any Turing machine T can be described by a string, $\langle T \rangle$.
- Another Turing machine U can simulate the operation of T on input string w, when given input $\langle T \rangle$ and w.
- A Turing machine, such as U, that can simulate any other Turing machine is called a **Universal Turing Machine**.

The Church-Turing Thesis

Every computational process that is intuitively considered to be an algorithm can be converted to a Turing machine.

The Church-Turing Thesis

Every computational process that is intuitively considered to be an algorithm can be converted to a Turing machine.

- Not a theorem.
- Can't be proved because it proposed as a *definition* of **algorithm**.

Decidability

A language A over Σ is *decidable* if there exists a Turing machine M such that for every string $w \in \Sigma^*$:

1. If $w \in A$ then M, started on w, halts in an accept state.
2. If $w \notin A$ then M, started on w, halts in a reject state.

- We will call a machine like this a **decider** for the language.

Enumerability

A language A over Σ is *enumerable* if there exists a Turing machine M such that for every string $w \in \Sigma^*$:

1. If $w \in A$ then M, started on w, halts in the accept state.
2. If $w \notin A$ then M, started on w, either halts in the reject state or loops forever.

- We will call a machine like this a **recognizer** for the language.
- A machine that produces each string in a language, one at a time, is an **enumerator** for the language.

Enumerability

A language A over Σ is *enumerable* if there exists a Turing machine M such that for every string $w \in \Sigma^*$:

1. If $w \in A$ then M, started on w, halts in the accept state.
2. If $w \notin A$ then M, started on w, either halts in the reject state or loops forever.

- We will call a machine like this a **recognizer** for the language.
- A machine that produces each string in a language, one at a time, is an **enumerator** for the language.
- We will prove shortly that the existence of a recognizer is equivalent to the existence of an enumerator.

Decidable *vs.* enumerable

- **Decidable** is also called
  - **computable**
  - **recursive**

- **Enumerable** is also called
  - **semi-decidable**
  - **recognizable**
  - **recursively enumerable**

Enumerator $\Rightarrow$ Recognizer

If we have an enumerator M_E for a language L,
we can construct a recognizer M_R for L.

Enumerator $\Rightarrow$ Recognizer

If we have an enumerator M_E for a language L,
we can construct a recognizer M_R for L.

- M_R:
  - On input w:
    - Start running M_E, producing series $s_1, s_2, s_3, \ldots$
    - If $w = s_i$ for any $i \in \mathbb{N}$, halt with **accept**.

Enumerator $\Rightarrow$ Recognizer

If we have an enumerator M_E for a language L,
we can construct a recognizer M_R for L.

- M_R:
  - On input w:
    - Start running M_E, producing series $s_1, s_2, s_3, \ldots$
    - If $w = s_i$ for any $i \in \mathbb{N}$, halt with **accept**.

- If $x \in L$ then on input x, M_R will halt with accept.
- If $x \notin L$ then on input x, M_R will run forever.

Recognizer $\Rightarrow$ Enumerator

If we have a recognizer M_R for a language L,
we can construct an enumerator M_E for L.

Recognizer $\Rightarrow$ Enumerator

If we have a recognizer M_R for a language L,
we can construct an enumerator M_E for L.

- This is tricky.
- We can build a machine to enumerate all possible strings:

$$\underbrace{\epsilon}_{0}, \underbrace{0, 1}_{1}, \underbrace{00, 01, 10, 11}_{2}, \underbrace{000, 001, 010, 011, 100, \ldots}_{3}, \underbrace{0000, \ldots}_{4}, \ldots$$

Recognizer $\Rightarrow$ Enumerator

If we have a recognizer M_R for a language L,
we can construct an enumerator M_E for L.

- This is tricky.
- We can build a machine to enumerate all possible strings:

$$\underbrace{\epsilon}_{0}, \underbrace{0, 1}_{1}, \underbrace{00, 01, 10, 11}_{2}, \underbrace{000, 001, 010, 011, 100, \ldots}_{3}, \underbrace{0000, \ldots}_{4}, \ldots$$

- We might suppose we just run M_R on each string, and output any that are accepted.
- That would be the machine, M_E, right?

Recognizer $\Rightarrow$ Enumerator

If we have a recognizer M_R for a language L,
we can construct an enumerator M_E for L.

- This is tricky.
- We can build a machine to enumerate all possible strings:

$$\underbrace{\epsilon}_{0}, \underbrace{0, 1}_{1}, \underbrace{00, 01, 10, 11}_{2}, \underbrace{000, 001, 010, 011, 100, \ldots}_{3}, \underbrace{0000, \ldots}_{4}, \ldots$$

- We might suppose we just run M_R on each string, and output any that are accepted.
- That would be the machine, M_E, right?
- But we *can't* just run M_R on each of these strings!
- M_R might not halt!
- What to do?

Recognizer $\Rightarrow$ Enumerator

If we have a recognizer M_R for a language L,
we can construct an enumerator M_E for L.

Recognizer $\Rightarrow$ Enumerator

If we have a recognizer M_R for a language L,
we can construct an enumerator M_E for L.

- M_E:
  - Generate, one at a time, all possible $s \in \Sigma^*$: $s_1, s_2, s_3, \ldots$.
  - Keep them in a list.
  - After each string s_i is added to the list, run M_R on all strings in the list for i steps.
  - If any run of M_R accepts a string, output that string and remove it from the list.
  - If any run of M_R rejects a string, remove it from the list.

This machine will eventually run all possible strings for all possible number of steps. Hence, if M_R ever recognizes a string, this machine will output it. If a string is never recongized by M_R, it will never be output.

Describing machines and problems as strings

- We assume any machine (DFA, PDA, TM) can be described by a string M using some alphabet.
- The input to any machine is a string w using some alphabet.
- We can thus describe both a machine M and its input w, with a pair of strings: (M, w).
- This pair can be converted to a single string $\langle M, w \rangle$.
- For convenience, we assume $\langle M, w \rangle$ is encoded in binary.
- In general, $\langle x \rangle$ means: encode x as a binary string.
- We can now define a language A as the set of all strings $\langle M, w \rangle$ such that $w \in \mathcal{L}(M)$, the language of M.

The language A_{DFA} is decidable

$$A_{DFA} = \{\langle M, w\rangle : M \text{ is a DFA that accepts } w\}$$

Proof?

The language A_{DFA} is decidable

$$A_{DFA} = \{\langle M, w\rangle : M \text{ is a DFA that accepts } w\}$$

Proof?

- Given input $\langle M, w\rangle$:
  - Run M on w.
  - It must terminate.
  - If it accepts, accept, else reject.

The language A_{NFA} is decidable

$$A_{NFA} = \{\langle M, w\rangle : M \text{ is a NFA that accepts } w\}$$

Proof?

The language A_{NFA} is decidable

$$A_{NFA} = \{\langle M, w\rangle : M \text{ is a NFA that accepts } w\}$$

Proof?

- Given input $\langle M, w\rangle$:
  - Convert NFA M to DFA N.
  - This algorithm terminates.
  - Run N on w.
  - It must terminate.
  - If it accepts, accept, else reject.

The language A_{CFG} is decidable

$$A_{CFG} = \{\langle M, w\rangle : M \text{ is a CFG that accepts } w\}$$

Proof?

The language A_{CFG} is decidable

$$A_{CFG} = \{\langle M, w\rangle : M \text{ is a CFG that accepts } w\}$$

Proof?

- Given input $\langle M, w\rangle$:
  - Convert CFG M to Chomsky normal form CFG N.
  - This algorithm terminates.
  - Generate all derivations of length $2|w| - 1$ from N.
  - There are a finite number of these, so it must terminate.
  - If any derivation yields w, accept, else reject.

The language A_{TM} is not decidable.

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

Proof?

The language A_{TM} is not decidable.

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

Proof? By contradiction.

- Assume there is a TM H that decides this language.

The language A_{TM} is not decidable.

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

Proof? By contradiction.

- Assume there is a TM H that decides this language.
- Construct the following TM, D:

  D: On input $\langle M\rangle$:
  - Run H on $\langle M, \langle M\rangle\rangle$.
  - If H accepts, reject, else accept.
- If H accepts $\langle D, \langle D\rangle\rangle$, then D rejects $\langle D\rangle$.
  - Therefore, by definition $\langle D, \langle D\rangle\rangle \notin A_{TM}$.
- If H rejects $\langle D, \langle D\rangle\rangle$, then D accepts $\langle D\rangle$.
  - Therefore, by definition $\langle D, \langle D\rangle\rangle \in A_{TM}$.
- In either case, H does not decide A_{TM}.

Diagonal argument

- Machine H that decides A_{TM} can fill in this table:

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	...
M_0	**accept**	*accept*	*accept*	*reject*	*accept*	*reject*	...
M_1	*accept*	**reject**	*accept*	*accept*	*accept*	*reject*	...
M_2	*accept*	*reject*	**accept**	*accept*	*accept*	*reject*	...
M_3	*accept*	*accept*	*reject*	**reject**	*accept*	*accept*	...
M_4	*reject*	*accept*	*accept*	*reject*	**accept**	*accept*	...
M_5	*reject*	*reject*	*accept*	*accept*	*accept*	**reject**	...
...	...	...	...	...	...	...	...

- D uses H to give the opposite answer on the diagonal.
- H must give the wrong answer somewhere on machine D.

The language A_{TM} is enumerable but not decidable.

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

Proof?

The language A_{TM} is enumerable but not decidable.

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

Proof?

- Given input $\langle M, w\rangle$:
  - Simulate the operation of M on w.
  - If this terminates with accept, accept.

The language *Halt* is not decidable.

$$Halt = \{\langle M, w\rangle : M \text{ is a TM that terminates on } w\}$$

Proof?

The language *Halt* is not decidable.

$$Halt = \{\langle M, w \rangle : M \text{ is a TM that terminates on } w\}$$

Proof?

By contradiction. Assume there is a TM H that decides this language. Construct the following TM, Q:

Q: On input $\langle M \rangle$:

- while $H(\langle M, \langle M \rangle \rangle)$ do end;

- What happens if we run Q on itself?
- If H says $Q(\langle Q \rangle)$ terminates, $Q(\langle Q \rangle)$ does not terminate.
- If H says $Q(\langle Q \rangle)$ does not terminate, $Q(\langle Q \rangle)$ terminates.
- $Q(\langle Q \rangle)$ terminates iff $Q(\langle Q \rangle)$ does not terminate.

The language *Halt* is not decidable.

$$Halt = \{\langle M, w\rangle : M \text{ is a TM that terminates on } w\}$$

Proof?

By contradiction. Assume there is a TM H that decides this language. Construct the following TM, Q:

> Q: On input $\langle M\rangle$:
> - while $H(\langle M, \langle M\rangle\rangle)$ do end;

- What happens if we run Q on itself?
- If H says $Q(\langle Q\rangle)$ terminates, $Q(\langle Q\rangle)$ does not terminate.
- If H says $Q(\langle Q\rangle)$ does not terminate, $Q(\langle Q\rangle)$ terminates.
- $Q(\langle Q\rangle)$ terminates iff $Q(\langle Q\rangle)$ does not terminate.
- Can also use a diagonal argument.

The language M_a is not decidable.

$$M_a = \{\langle M \rangle \mid \mathcal{L}(M) = \{a\}\}$$

Proof?

The language M_a is not decidable.

$$M_a = \{\langle M \rangle \mid \mathcal{L}(M) = \{a\}\}$$

Proof?

By contradiction.

- Suppose TM A decides M_a.
- Construct the following TM, H:

> H: On input $\langle M, w \rangle$:
> - Construct TM D:
>   > D: On input $\langle s \rangle$:
>   > - Run M on w.
>   > - If $s = a$ accept, else reject.
> - Run A on D. If it accepts, accept, else reject.

The language M_a is not decidable.

$$M_a = \{\langle M \rangle \mid \mathcal{L}(M) = \{a\}\}$$

Proof?

By contradiction.

- Suppose TM A decides M_a.
- Construct the following TM, H:

> H: On input $\langle M, w \rangle$:
> - Construct TM D:
>
>   > D: On input $\langle s \rangle$:
>   > - Run M on w.
>   > - If $s = a$ accept, else reject.
>
> - Run A on D. If it accepts, accept, else reject.

- $\mathcal{L}(D) = \{a\}$ iff M halts on w.

The language M_a is not decidable.

$$M_a = \{\langle M \rangle \mid \mathcal{L}(M) = \{a\}\}$$

Proof?

By contradiction.

- Suppose TM A decides M_a.
- Construct the following TM, H:

  H: On input $\langle M, w \rangle$:
  - Construct TM D:

    D: On input $\langle s \rangle$:
    - Run M on w.
    - If $s = a$ accept, else reject.
  - Run A on D. If it accepts, accept, else reject.
- $\mathcal{L}(D) = \{a\}$ iff M halts on w.
- H decides the language *Halt*. But that's impossible!

The language $M_\emptyset$ is not decidable.

$$M_\emptyset = \{\langle M \rangle \mid \mathcal{L}(M) = \emptyset\}$$

Proof?

The language $M_\emptyset$ is not decidable.

$$M_\emptyset = \{\langle M \rangle \mid \mathcal{L}(M) = \emptyset\}$$

Proof?

By contradiction.

- Suppose TM A decides $M_\emptyset$.
- Construct the following TM, H:

  H: On input $\langle M, w \rangle$:

  - Construct TM D:

    D: On input $\langle s \rangle$:

    - Run M on w.
    - Accept.
  - Run A on D. If it accepts, reject, else accept.

The language $M_\emptyset$ is not decidable.

$$M_\emptyset = \{\langle M \rangle \mid \mathcal{L}(M) = \emptyset\}$$

Proof?

By contradiction.

- Suppose TM A decides $M_\emptyset$.
- Construct the following TM, H:

  H: On input $\langle M, w \rangle$:
  - Construct TM D:

    D: On input $\langle s \rangle$:
    - Run M on w.
    - Accept.
  - Run A on D. If it accepts, reject, else accept.
- $\mathcal{L}(D) = \emptyset$ iff M does not halt on w.

The language $M_\emptyset$ is not decidable.

$$M_\emptyset = \{\langle M \rangle \mid \mathcal{L}(M) = \emptyset\}$$

Proof?

By contradiction.

- Suppose TM A decides $M_\emptyset$.
- Construct the following TM, H:

> H: On input $\langle M, w \rangle$:
> - Construct TM D:
>   > D: On input $\langle s \rangle$:
>   > - Run M on w.
>   > - Accept.
> - Run A on D. If it accepts, reject, else accept.

- $\mathcal{L}(D) = \emptyset$ iff M does not halt on w.
- H decides the language *Halt*. But that's impossible!

Rice's Theorem

Let $\mathcal{T}$ be the set of all binary encoded TMs.
Let $\mathcal{P}$ be a subset of $\mathcal{T}$ such that

1. $\mathcal{P} \neq \emptyset$
2. $\mathcal{P} \neq \mathcal{T}$
3. If $L(M_1) = L(M_2)$, then either both or neither is in $\mathcal{P}$.

Then $\mathcal{P}$ is undecidable.

Rice's Theorem Examples

1. $\{\langle M \rangle \mid M$ accepts only inputs in the language $a^*b^*\}$
2. $\{\langle M \rangle \mid M$ accepts only input of length $n^2\}$
3. $\{\langle M \rangle \mid M$ accepts only input of length $10\}$
4. $\{\langle M \rangle \mid M$ accepts all inputs$\}$
5. $\{\langle M \rangle \mid M$ rejects all inputs$\}$

None of these is decideable.

Rice's Theorem Examples

1. $\{\langle M \rangle \mid M$ accepts only inputs in the language $a^* b^*\}$
2. $\{\langle M \rangle \mid M$ accepts only input of length $n^2\}$
3. $\{\langle M \rangle \mid M$ accepts only input of length $10\}$
4. $\{\langle M \rangle \mid M$ accepts all inputs$\}$
5. $\{\langle M \rangle \mid M$ rejects all inputs$\}$

None of these is decideable.

- Note that it is easy to find a machine in each of these sets.
- It is also easy to find a machine not in each of these sets.
- What *cannot* be done is find a machine that will decide if *another* machine is in one of these sets.

Hilbert's 10th problem is enumerable but not decidable

$$\mathit{Hilbert} = \{\langle p \rangle \quad : \quad p \text{ is a polynomial with integer coefficients that has an integral root}\}$$

$$15x^3y^2 + 12xy^2 - 17x^9y^2 + 2x - 5y + 3 = 0$$

Post Correspondence Problem is enumerable but not decidable

- Given a finite set of dominoes with strings on the top and the bottom, and an unlimited supply of each domino, does there exist a sequence of these dominoes such that the string at the top matches the string at the bottom?
- For example, given the set of three dominos:

a	ab	bba
baa	aa	bb

- We can find a sequence:

bba	ab	bba	a
bb	aa	bb	baa

- Where the top and bottom rows are both: bbaabbbaa

Uncomputable real number

- A *computable real number* is one for which there is a Turing machine which, given *n* on its initial tape, terminates with the *n*th digit of the decimal expansion of that number encoded on its tape.
  - π and $\sqrt{2}$ are computable, even though their decimal expansions never end.

Uncomputable real number

- A *computable real number* is one for which there is a Turing machine which, given n on its initial tape, terminates with the nth digit of the decimal expansion of that number encoded on its tape.
    - π and $\sqrt{2}$ are computable, even though their decimal expansions never end.
- All possible Turing machines can be enumerated, since each is represented by a unique string. Let the ith Turing machine be denoted by T_i.

Uncomputable real number

- A *computable real number* is one for which there is a Turing machine which, given n on its initial tape, terminates with the nth digit of the decimal expansion of that number encoded on its tape.
  - π and $\sqrt{2}$ are computable, even though their decimal expansions never end.
- All possible Turing machines can be enumerated, since each is represented by a unique string. Let the ith Turing machine be denoted by T_i.
- Let x be the real number between 0 and 1 with the following decimal expansion:
  The ith digit of x is 1 if $\mathcal{L}(T_i) = \emptyset$, otherwise 0.

Uncomputable real number

- A *computable real number* is one for which there is a Turing machine which, given n on its initial tape, terminates with the nth digit of the decimal expansion of that number encoded on its tape.
  - π and $\sqrt{2}$ are computable, even though their decimal expansions never end.
- All possible Turing machines can be enumerated, since each is represented by a unique string. Let the ith Turing machine be denoted by T_i.
- Let x be the real number between 0 and 1 with the following decimal expansion:
  The ith digit of x is 1 if $\mathcal{L}(T_i) = \emptyset$, otherwise 0.
- x cannot be computable, because its solution would solve the halting problem (see above).

A language such that both A and $\overline{A}$ are not enumerable.

$$EQ_{TM} = \{\langle M_1, M_2 \rangle : \mathcal{L}(M_1) = \mathcal{L}(M_2)\}$$

EQ_{TM} is not enumerable

$$EQ_{TM} = \{\langle M_1, M_2\rangle : \mathcal{L}(M_1) = \mathcal{L}(M_2)\}$$

- Suppose EQ_{TM} is recognizable by TM $M_=$.
- Recall that $\overline{Halt}$ is not recognizable.
- For any M and w, define the following TM:

M_{Mw}: on input s: - Run M on w. - Accept

- Also define:

$M_\emptyset$: on input s, reject.

M_{Σ^*}: on input s, accept.

- Run $M_=$ on $\langle M_\emptyset, M_{Mw}\rangle$.
  - This accepts iff $\langle M, w\rangle \in \overline{Halt}$.
  - Therefore it is a recognizer for $\overline{Halt}$.

$\overline{EQ_{TM}}$ is not enumerable

$$\overline{EQ_{TM}} = \{\langle M_1, M_2\rangle : \mathcal{L}(M_1) \neq \mathcal{L}(M_2)\}$$

- Suppose $\overline{EQ_{TM}}$ is recognizable by TM $M_{\neq}$.
- Recall that $\overline{Halt}$ is not recognizable.
- For any M and w, define the following TM:

M_{Mw}: on input s: ▶ Run M on w. ▶ Accept

- Also define:

$M_{\emptyset}$: on input s, reject.

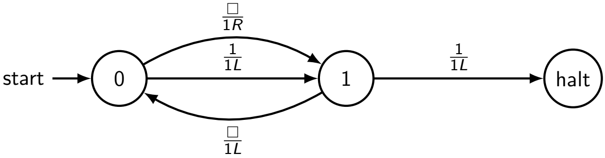
M_{Σ^*}: on input s, accept.

- Run $M_{\neq}$ on $\langle M_{\Sigma^*}, M_{Mw}\rangle$.
  - This accepts iff $\langle M, w\rangle \in \overline{Halt}$.
  - Therefore it is a recognizer for $\overline{Halt}$.

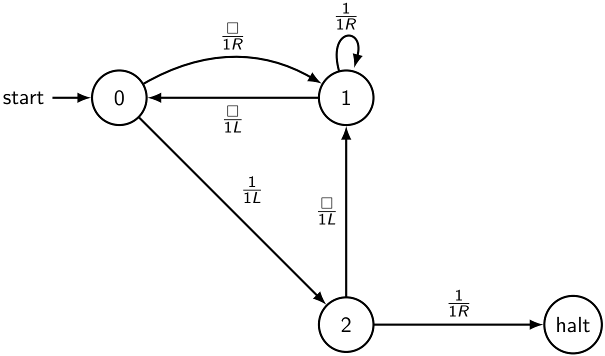
Busy beavers are not enumerable

The nth busy beaver number is the largest (finite) number of 1s that can be output by a Turing machine with n states when started on a blank tape.

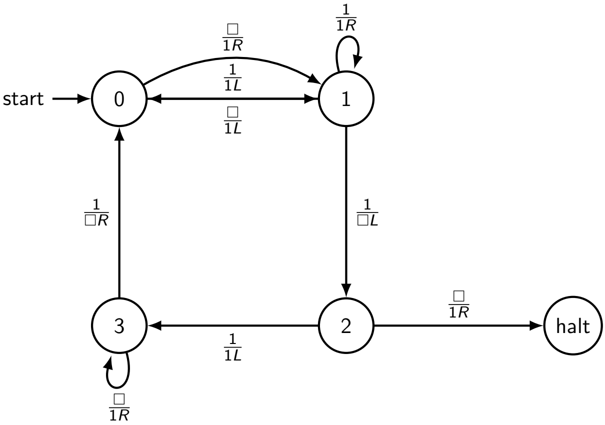
2 State Busy Beaver: four 1s



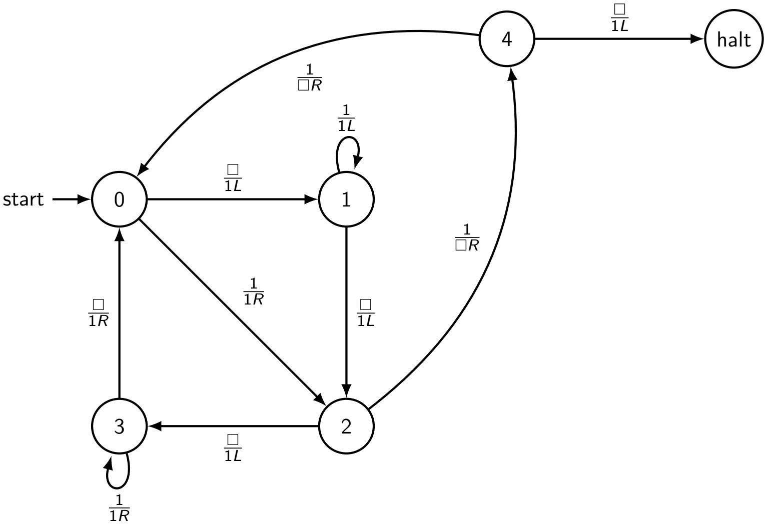
3 State Busy Beaver: six 1s



4 State Busy Beaver: thirteen 1s



5 State Busy Beaver (?): 4098 1s



Current Busy Beaver Records

$bb(2) = 4$

$bb(3) = 6$

$bb(4) = 13$

$bb(5) \geq 4098$ discovered in 1989

$bb(6) \geq 3.515 \times 10^{18267}$ discovered in 2010

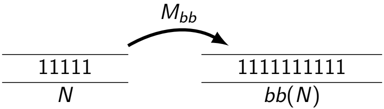
$bb(7) \geq 10^{10^{10^{10^{18705353}}}}$ actually, much bigger

Note: there are about 10^{80} atoms in the universe!

Proof Busy Beaver function is not computable

Proof by contradiction.

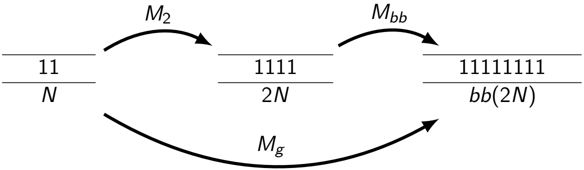
- Let $bb(n)$ be the largest (finite) number of 1's output by a Turing Machine with n states.
- Suppose there is a Turing Machine M_{bb} that computes $bb(n)$, that is, starting with n on the tape, the machine halts with $bb(n)$ on the tape.



- Note: this is a new use of TMs, computing a function from input to output, not recognizing a language.

Busy Beaver proof

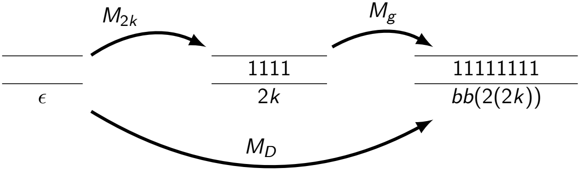
- Let $g(n) = bb(2n)$. We can build a TM for g by starting with a machine that doubles the input, and then runs the machine M_{bb}.



- Suppose the machine for g, M_g has k states.

Busy Beaver proof

- Build a machine M_{2k} with $2k$ states that does nothing but put $2k$ 1s on a blank tape.
- Now build a machine M_D that starts by putting $2k$ 1's on the tape, and then runs the M_g machine.



- M_D can be built with $3k$ states.
- The output of M_D is $g(2k) = bb(2(2k)) = bb(4k)$ 1s.
- Do you see the problem?

An Old Philosophical Problem

This sentence is false.

Quine's Paradox

"Yields falsehood when preceded by its quotation" yields falsehood when preceded by its quotation.

Self Reproducing Sentences

Print two copies of the following, the second one in quotes:
"Print two copies of the following, the second one in quotes:"

Self Reproducing Programs: "Quines"

```
(define data "Put the program below here,
so long as it doesn't have any strings in it.")
(define (display-as-data data)
  (display (integer->char 40))
  (display 'define)
  (display (integer->char 32))
  (display 'data)
  (display (integer->char 32))
  (display (integer->char 34))
  (display data)
  (display (integer->char 34))
  (display (integer->char 41))
  (newline))
(display-as-data data)
(display data)
```

The Recursion Theorem

Let T be a Turing machine that computes a function

$$t : \Sigma^* \times \Sigma^* \to \Sigma^*$$

There is a Turing machine R that computes a function

$$r : \Sigma^* \to \Sigma^*$$

where, for every $w \in \Sigma^*$,

$$r(w) = t(w, \langle R \rangle)$$

The Recursion Theorem

Let T be a Turing machine that computes a function

$$t : \Sigma^* \times \Sigma^* \to \Sigma^*$$

There is a Turing machine R that computes a function

$$r : \Sigma^* \to \Sigma^*$$

where, for every $w \in \Sigma^*$,

$$r(w) = t(w, \langle R \rangle)$$

- In other words, given any computation with two inputs, we can assume that it is given only one input and obtains a description of itself for the second input.

The language A_{TM} is not decidable: EASY PROOF!

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

Proof?

The language A_{TM} is not decidable: EASY PROOF!

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

Proof? By contradiction.

- Assume there is a TM H that decides this language.
- Construct the following TM, B:

  B: On input $\langle w \rangle$:
  - Obtain own description, $\langle B \rangle$.
  - Run H on $\langle B, w \rangle$.
  - If H accepts, reject, else accept.
- Running B on input w does the opposite of what H says.
- Therefore, H is wrong about B.
- H does not decide A_{TM}.