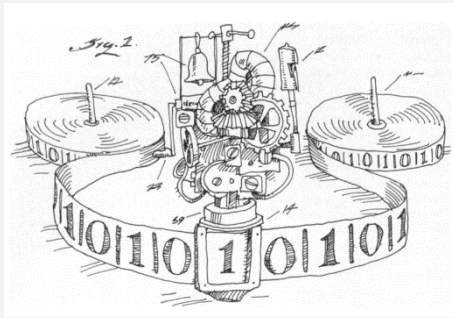
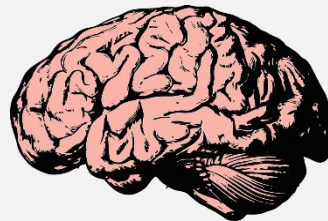
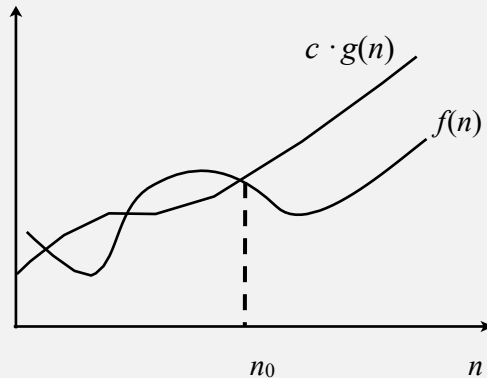


COMP 3602

Intro to Theory of Computation

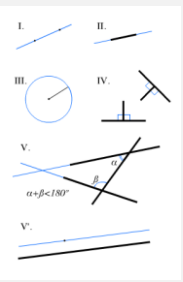
Lecture 3



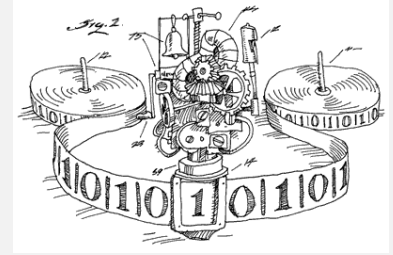
Logistics

- New room: EN-2007
 - For the rest of the semester
- Theory reading group
 - This week: Friday 1pm, from next week Monday 1pm.
- From Jan 19 to Feb 6, lectures will be 4pm to 5:15pm
- Office hours times: before lectures?

Questions/comments?

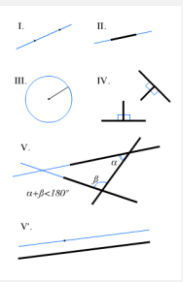


Hilbert's problems

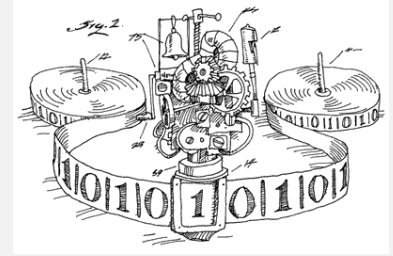


- In 1900, at the International Congress of Mathematicians in Paris, David Hilbert posed a list of 23 problems.
 - Problem 2: prove that mathematics contains no self-contradictions.
 - Problem 10: give an algorithm to find integer solutions to polynomial equations.
- In 1920, Hilbert extended it to what is now known as “Hilbert’s program”





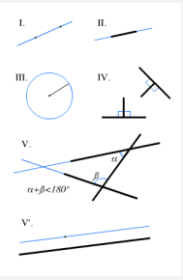
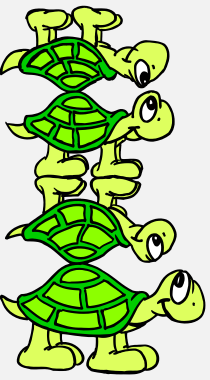
Hilbert's program



- Express all mathematics in a precise way
- Allowing a formal proof of all true statements
- With a proof, inside mathematics, that there is no self-contradiction
- And a procedure (an algorithm) for deciding, for any given mathematical statement, whether it is true or false.



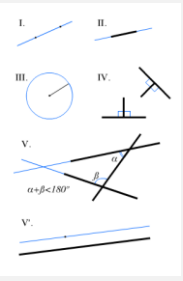
Gödel Incompleteness Theorem



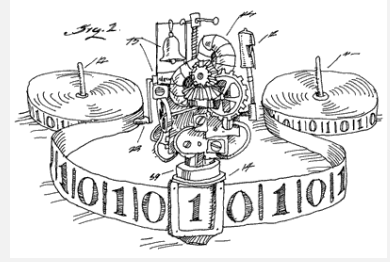
- Very informally:
 - If mathematics is not self-contradictory...
 - Then there are true statements that can't be proven!
 - Such as "I am not provable"

We will prove Gödel Incompleteness Theorem
later in this class





Church and Turing:

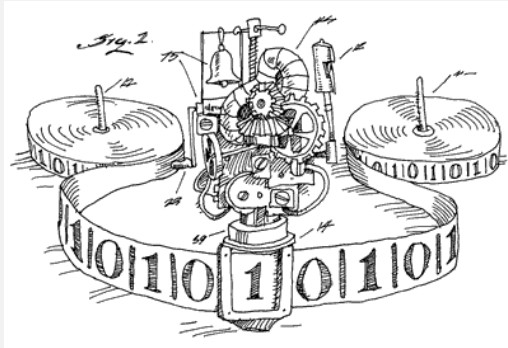


- Moreover, there is no procedure to decide if a given statement is true or false!
 - And to decide many other things...
- But what do we mean by a “procedure”?



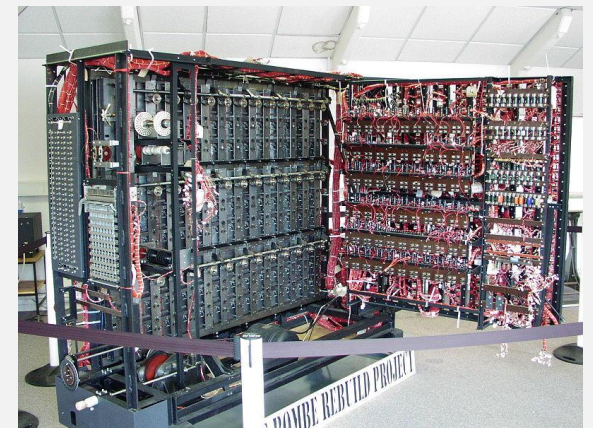
Turing machines

- Turing's idea: modeling how a person computes:
 - Write something on a piece of paper, put it aside, take next one...



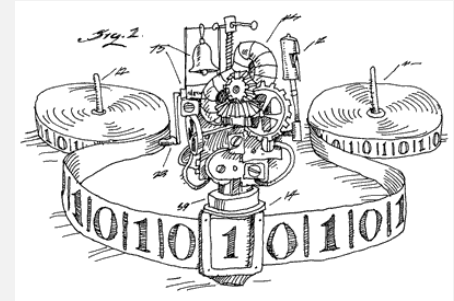
Digression: a bit about Alan Turing

- He is known for
 - The Turing machine
 - Breaking German's codes (Enigma machine) during the World War II
 - The Turing test (Artificial intelligence)
 - Equations describing patterns (like spots and stripes)
- He was prosecuted for homosexuality and died of cyanide poisoning in 1954...



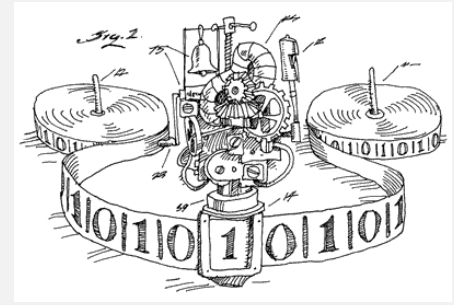
Turing machine

- A Turing machine (TM) has finitely many internal states, and (unlimited) memory, visualized as a tape
 - Or a stack of paper
- And takes very simple instructions:
 - Read a symbol
 - Write a symbol
 - Move one step left or right on the tape
 - Change internal state.



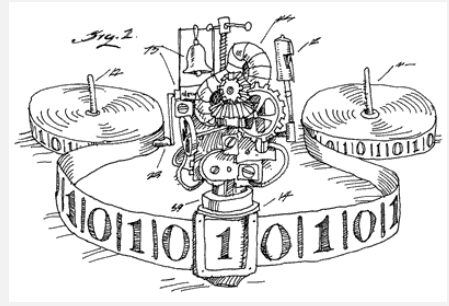
- A Turing machine has an (unlimited) memory, visualized as a tape
- And takes very simple instructions:
 - Read a symbol
 - Write a symbol
 - Move one step left or right on the tape
 - Change internal state.

Turing machine



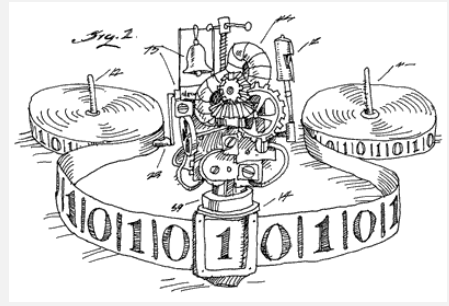
- Formally, to describe a Turing machine (TM) need to specify:
 - Its input alphabet Σ (Sigma, often $\{0,1\}^*$)
 - Its tape alphabet Γ (Gamma)
 - $\Gamma \supseteq \Sigma \cup \{\sqcup\}$, where $\sqcup$ is a special “blank” symbol not in Σ
 - Its set of states Q , including special states q_0 , q_{accept} , q_{reject}
 - q_0 is the start state
 - q_{accept} , q_{reject} are distinct halting states denoting “yes”/”no” answers.
 - Its transition table delta $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
 - Where L, R indicate that the machine moves to the left or to the right, respectively.

Turing machine



- A Turing machine computation starts with its tape blank except for the input
 - It starts in the start state q_0 looking at the first symbol of the input
 - Then keeps reading, writing and changing states according to δ
 - It stops (halts) if it reaches q_{accept} or q_{reject}
 - It may never stop

Computing with TMs



- A TM M *accepts (recognizes) a language* L if it stops in the state q_{accept} for all and only strings in L
 - It does not have to stop for strings not in L .
 - If it does (that is, rejects all strings not in L), then M *decides* L .
- For decision problems, we only care whether it stopped in the state q_{accept} (and “accepted its input”) or q_{reject} (“rejected its input”)
 - For search problems, what is written on the tape when it halts is the output. The head usually points to the first symbol of the output.

Example: even numbers in binary

- $L = \{w \in \{0,1\}^* \mid w \text{ encodes an even number in binary}\}$

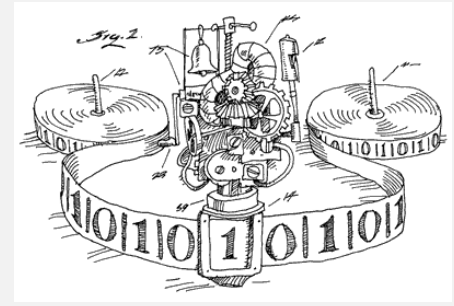
Example: even numbers in binary

- $L = \{w \in \{0,1\}^* \mid w \text{ encodes an even number in binary}\}$
- $\Sigma = \{0,1\}, \Gamma = \{0,1,\sqcup\}, Q = \{q_0, q_1, q_a, q_r\}$

• $\delta =$

Old State	Read	New State	Write	Move
q_0	0	q_0	0	R
q_0	1	q_0	1	R
q_0	$\sqcup$	q_1	$\sqcup$	L
q_1	0	q_a	0	R
q_1	1	q_r	1	R
q_1	$\sqcup$	q_r	$\sqcup$	L

Turing machine configurations



- A computation of a TM can be viewed as a sequence of *configurations*
 - Snapshots of the computation describing TM's state, head position and tape content.
 - Configuration is written as the non-blank part of the tape with the state name inserted before the symbol which it will be reading.
 - If the input string is $w_1 \dots w_n$, then the initial configuration (before a TM starts computing) is always $q_0 w_1 \dots w_n$

Example: even numbers in binary

- $L = \{w \in \{0,1\}^* \mid w \text{ encodes an even number in binary}\}$
- $\Sigma = \{0,1\}, \Gamma = \{0,1,\sqcup\}, Q = \{q_0, q_1, q_a, q_r\}$

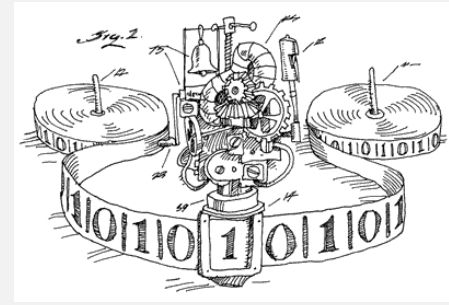
• $\delta =$

Old State	Read	New State	Write	Move
q_0	0	q_0	0	R
q_0	1	q_0	1	R
q_0	$\sqcup$	q_1	$\sqcup$	L
q_1	0	q_a	0	R
q_1	1	q_r	1	R
q_1	$\sqcup$	q_r	$\sqcup$	L

Example: even length palindrome

$$L = \{w \in \{b, c, d\}^* \mid \forall i \leq |w| \ w_i = w_{|w|-i+1} \text{ and } \exists k \in \mathbb{N} \ |w| = 2k\}$$

Turing machine



- A description of a TM consists of
 - Input alphabet Σ , tape alphabet $\Gamma, \Gamma \supseteq \Sigma \cup \{\sqcup\}$,
 - Set of states Q , including special states q_0 (start), q_a (accept), q_r (reject),
 - Transition table delta $\delta: (Q - \{q_a, q_r\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- A TM M *accepts* w if there is a sequence of configurations $C_0 \dots C_t$ such that
 - $C_0 = q_0 w_1 \dots w_{|w|}$,
 - for every i , C_{i+1} follows from C_i by applying a rule from δ (C_i *yields* C_{i+1})
 - C_t is an accepting configuration (contains q_a)
- A TM M *recognizes* a language L iff it accepts all and only $w \in L$.
 - Two TMs are *equivalent* if they recognize the same language.

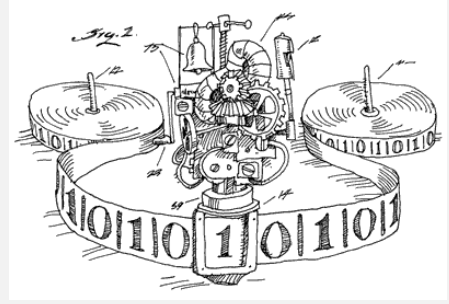
Showing equivalence

Theorem. Every TM for a two-way-infinite tape model can be converted into a TM which would accept exactly the same language, but on a one-way-infinite tape, and vice versa.

General proof structure:

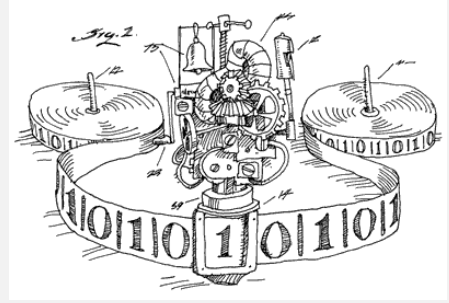
- **For each direction, give a construction:**
 - From an arbitrary M on one-way-infinite tape construct M' doing the same thing on two-way-infinite
 - And from an arbitrary M on two-way-infinite tape construct M' on one-way-infinite
- **For each construction, show that it works:**
 - That in both cases M' accepts exactly the same language as the corresponding M .
 - This is another “if and only if”.
- Always start by giving the informal **intuition** for what you are trying to accomplish.
 - What is your construction trying to achieve? Why would it work, informally?
- Sometimes, need to prove more:
 - eg, that M' always halts when M halts, or give a bound on how much slower M' would be than M .
 - This often can be phrased as a corollary of other statements you prove.
- It might help to prove little lemmas/claims, especially if you can use them more than once.

Showing equivalence



- First, show that everything that can be done on a one-way-infinite tape model can be done with a two-way-infinite tape.
 - Let $M = \{\Sigma, \Gamma, Q, \delta, q_0, q_a, q_r\}$ be a TM for one-way infinite tape.
 - We will convert it into M' which would work on a two-way tape.
- Idea: add a new symbol denoting the “beginning of the tape”. When that symbol is reached, go back to the right, keep the same state.
- Construction: $M' = \{\Sigma', \Gamma', Q', \delta', q'_0, q'_a, q'_r\}$ where
 - $\Sigma' = \Sigma$ (that is, M' has the same input alphabet as M)
 - $\Gamma' = \Gamma \cup \{\#\}$ for some $\# \notin \Gamma$ which will be our “tape beginning marker”
 - $q'_a = q_a$, $q'_r = q_r$, and new states q'_0 (new start state) and $q_\#$. $Q' = Q \cup \{q'_0, q_\#\}$.
 - δ' contains δ , plus, for every $q \in Q$, a rule $(q, \#) \rightarrow (q, \#, R)$
 - As well as new rules $(q'_0, s) \rightarrow (q_\#, s, L)$, $(q_\#, s) \rightarrow (q_0, \#, R)$ for $\forall s \in \Gamma'$.

One-way vs. 2-way infinite tape TM



- First, show that everything that can be done on a one-way-infinite tape model can be done with a two-way-infinite tape.
 - Let $M = \{\Sigma, \Gamma, Q, \delta, q_0, q_a, q_r\}$ be a TM for one-way infinite tape.
 - We will convert it into M' which would work on a two-way tape.
- Idea: add a new symbol denoting the “beginning of the tape”. When that symbol is reached, go back to the right, keep the same state.
- Construction: $M' = \{\Sigma', \Gamma', Q', \delta', q'_0, q'_a, q'_r\}$ where
 - $\Sigma' = \Sigma$ (that is, M' has the same input alphabet as M)
 - $\Gamma' = \Gamma \cup \{\#\}$ for some $\# \notin \Gamma$ which will be our “tape beginning marker”
 - $q'_a = q_a$, $q'_r = q_r$, and new states q'_0 (new start state) and $q_\#$. $Q' = Q \cup \{q'_0, q_\#\}$.
 - δ' contains δ , plus, for every $q \in Q$, a rule $(q, \#) \rightarrow (q, \#, R)$
 - As well as new rules $(q'_0, s) \rightarrow (q_\#, s, L)$, $(q_\#, s) \rightarrow (q_0, \#, R)$ for $\forall s \in \Gamma'$.

- Construction: $M' = \{\Sigma', \Gamma', Q', \delta', q'_0, q'_a, q'_r\}$ where
 - $\Sigma' = \Sigma$ (that is, M' has the same input alphabet as M)
 - $\Gamma' = \Gamma \cup \{\#\}$ for some $\# \notin \Gamma$ which will be our “tape beginning marker”
 - $q'_a = q_a$, $q'_r = q_r$, and new states q'_0 (new start state) and $q_\#$. $Q' = Q \cup \{q'_0, q_\#\}$.
 - δ' contains δ , plus, for every $q \in Q$, a rule $(q, \#) \rightarrow (q, \#, R)$
 - As well as new rules $(q'_0, s) \rightarrow (q_\#, s, L)$, $(q_\#, s) \rightarrow (q_0, \#, R)$ for $\forall s \in \Gamma'$.

Then M' has $\Sigma' = \Sigma = \{0\}$, $\Gamma' = \{0, \sqcup, \#\}$,
 $Q' = \{q'_0, q_\#, q_0, q_1, q_a, q_r\}$, $\delta' =$

- Example:
 - Let M be as follows: $\Sigma = \{0\}$, $\Gamma = \{0, \sqcup\}$, $Q = \{q_0, q_1, q_a, q_r\}$, $\delta =$

Old State	Read	New state	Write	Move
q_0	0	q_1	0	L
q_0	$\sqcup$	q_r	0	L
q_1	0	q_1	$\sqcup$	R
q_1	$\sqcup$	q_0	0	L

On input 00, the computation of

M

q_0 0 0
 (head stays)
 q_1 0 0
 $\sqcup$ q_1 0
 $\sqcup$ $\sqcup$ q_1 $\sqcup$
 $\sqcup$ q_0 $\sqcup$ 0
 q_r $\sqcup$ 0 0

M'

q'_0 0 0
 $q_\#$ $\sqcup$ 0 0
 $\#$ q_0 0 0
 q_1 $\#$ 0 0
 $\#$ q_1 0 0
 $\#$ $\sqcup$ q_1 0
 $\#$ $\sqcup$ $\sqcup$ q_1 $\sqcup$
 $\#$ $\sqcup$ q_0 $\sqcup$ 0
 $\#$ q_r $\sqcup$ 0 0

Old State	Read	New state	Write	Move
q'_0	s	$q_\#$	s	L
$q_\#$	s	q_0	#	R
q_0	0	q_1	0	L
q_0	$\sqcup$	q_r	0	L
q_0	#	q_0	#	R
q_1	0	q_1	$\sqcup$	R
q_1	$\sqcup$	q_0	0	L
q_1	#	q_1	#	R

Here first 2 rows should be 3 rows each, one for each value of $s \in \Gamma'$.

Claim: M' accepts a string on a 2-way infinite tape iff M does on one-way infinite tape.

Proof. We show two directions: if M accepts w , then so does M' , and the only way for M' to accept w is when M also does.

- Consider the first t steps of the computation of M on a string w
 - if M halted within t steps, assume, wlog, that it halted at the last step.
- Let $C_0 \dots C_t$ be a sequence of configurations of M .
 - Then the computation of M' starts with $C'_0 \dots C'_t$,
 - Where $C'_0 = q'_0 w_1 \dots w_n$, $C'_1 = q_{\#} \sqcup w_1 \dots w_n$, $C'_2 = \#q_0 w_1 \dots w_n$.

Claim: M' accepts a string on a 2-way infinite tape iff M does on one-way infinite tape.
Proof.

- Let $C_0 \dots C_t$ be a sequence of configurations of M .
 - Then the computation of M' starts with $C'_0 \dots C'_{t'}$,
 - Where $C'_0 = q'_0 w_1 \dots w_n$, $C'_1 = q_{\#} \sqcup w_1 \dots w_n$, $C'_2 = \#q_0 w_1 \dots w_n$.

- Lemma:* $C'_2 \dots C'_{t'}$ consists of $\#C_0 \dots \#C_{t-1}$ plus
- For every i where $C_i = qx_1 \dots x_m$, at most one more configuration $q\#x_1 \dots x_m$ before $\#C_i$
 - If $C'_t = \#C_t$ or $C'_t = q_h\#x_1 \dots x_m$, where $C_t = q_hx_1 \dots x_m$, for $q_h = q_a$ or $q_h = q_r$
- *Proof:* by induction on i

Corollary: The only possible halting configuration in the first t' steps of computation of M' on w is $C'_{t'}$, which is accepting iff C_t is, and rejecting iff C_t is.

Corollary 2: $t' \leq 2t + 2$

- M' is at most twice as slow as M .

Claim: M' accepts a string on a 2-way infinite tape **iff** M does on one-way infinite tape.

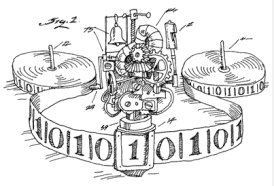
$\Leftarrow$ Suppose M accepts w .

- Then its computation is $C_0 \dots C_t$ with (only) C_t containing a halting state: q_a
- By the corollary, none of $C'_0 \dots C'_{t'-1}$ are halting states.
- By the lemma, $C'_{t'} = \#C_t$ or $q_a \#x_1 \dots x_n$.
- So $C'_0 \dots C'_{t'}$ is an accepting computation of M' on w .

$\Rightarrow$ Suppose M' accepts w .

- Then (only) $C'_{t'}$ is an accepting configuration of computation of M' on w .
- By the lemma, all $C_0 \dots C_{t-1}$ are in $C'_2 \dots C'_{t'-1}$ and so by the corollary, none of them is halting. Neither are C'_0, C'_1 by construction
- If $C'_{t'} = \#C_t$, done: then M has a transition from C_{t-1} to C_t , and C_t is accepting.
- Otherwise, $C'_{t'} = q_a \#x_1 \dots x_m$. By the lemma, $C'_{t'-1} = \#C_{t-1} = \#q y_1 x_2 \dots x_m$.
- M' got from $C'_{t'-1}$ to $C'_{t'}$ by applying $(q, y_1) \rightarrow (q_a, x_1, L)$ which, since $y_1 \neq \#$ and $q \neq q'_0$, was in δ
- In M , since the tape is one-way infinite, applying this rule to $q y_1 x_2 \dots x_m$ results in $q_a x_1 \dots x_m$.
- Therefore, M accepts w .

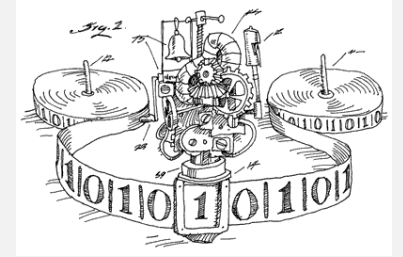
Other direction: from 2-way to 1-way inf.



- Now the other direction: show that everything that can be done on a two-way-infinite tape model can be done with a one-way-infinite tape.
 - Let $M = \{\Sigma, \Gamma, Q, \delta, q_0, q_a, q_r\}$ be a TM for two-way infinite tape.
 - We will convert it into M' which would work on a one-way tape.
- Idea: use every other tape cell for the “to the right” tape, and the ones in between for “to the left” tape (there, flip left/right in rules).
 - Start by putting in start marker at the first cell (to know where the tape starts), and spreading out the input by writing it in every other cell, keeping the rest blank.



Other variants

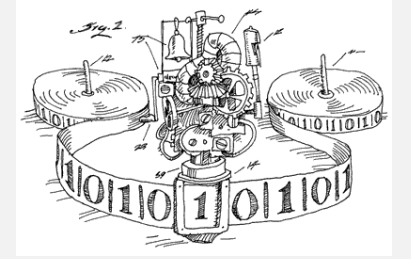


- Multi-tape TM:
 - There are k tapes and heads. Common state space.
 - $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
 - Here “S” means do not move on that tape.
- Non-deterministic TM:
 - $\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
 - Accepts w if there exists **some** valid accepting computation (sequence of configurations from initial to accepting where each configuration follows from previous).
- RAM
 - Random-access memory machine: a model closest to a standard computer

The Turing machine model we defined can simulate them all, though not necessarily efficiently.



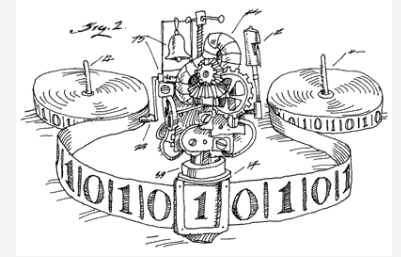
Multi-tape TM



- There are k tapes and heads. Common state space.
 - $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
- Example: palindrome



Multi-tape TM



Theorem: Single-tape and multi-tape models of TM are equivalent.

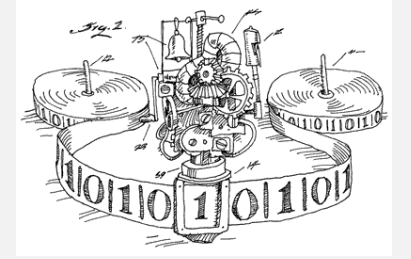
$\Rightarrow$ Single-tape TM is a special case of multi-tape, so trivially multi-tape can do all that single-tape can.

$\Leftarrow$ *Claim:* Single-tape TM can simulate multi-tape TM.

- Proof idea:
 - Write content of all tapes one after another, with some separator # between them.
 - For each symbol $s \in \Gamma$, introduce s^* denoting “s with a head on it”.
 - To simulate one step of M , go all the way to the left (to figure out what rule to apply), then all the way to the right to change the tape.
 - If need to write over blanks, move the whole tape content over.



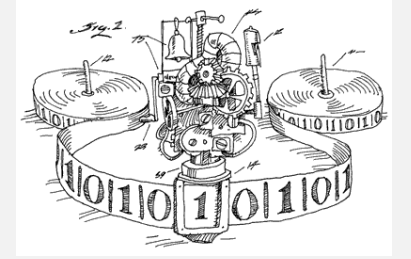
Non-deterministic TM



- $\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
 - Accepts w if there exists **some** accepting computation



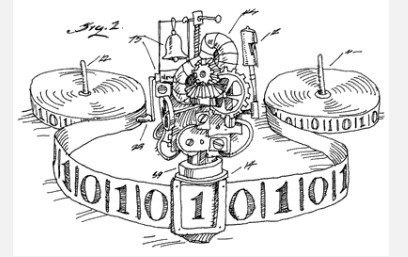
Non-deterministic TM



- $\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
 - Accepts w if there exists **some** accepting computation



Non-deterministic TM

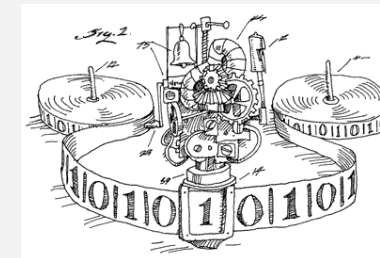


Claim: deterministic TM can simulate non-deterministic TM.

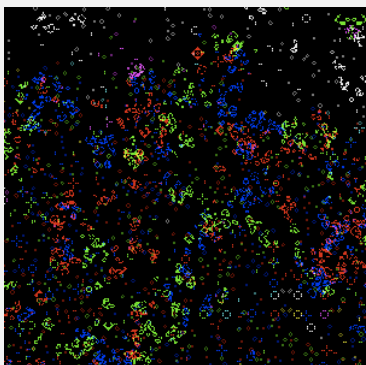
Proof idea: go through all possible sequences of choices of a rule in a breadth-first-search fashion.



Church-Turing thesis

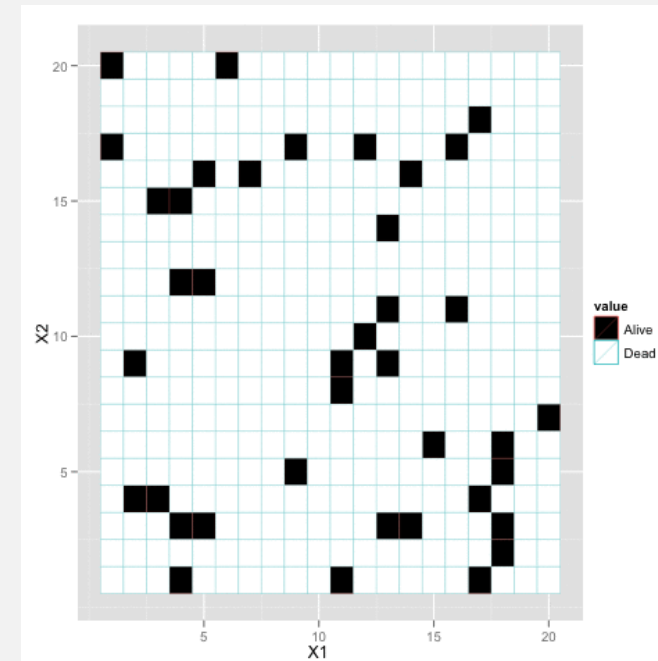


Everything we can call “computable” in any sense of this word is computable by a Turing machine.



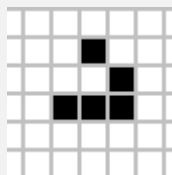
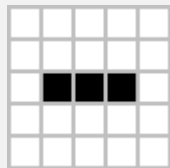
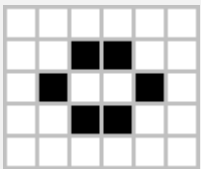
Conway's game of life

- Invented by John Conway
 - Mathematician, worked in Princeton
 - Proved “free will theorem” in quantum
 - Died in April 2020 from Covid...
- The game is a board (square grid) with cells some cells marked “live” (rest “dead”)
- At each iteration, cells may flip between “live” and “dead” according to what their neighbours were in the previous iteration

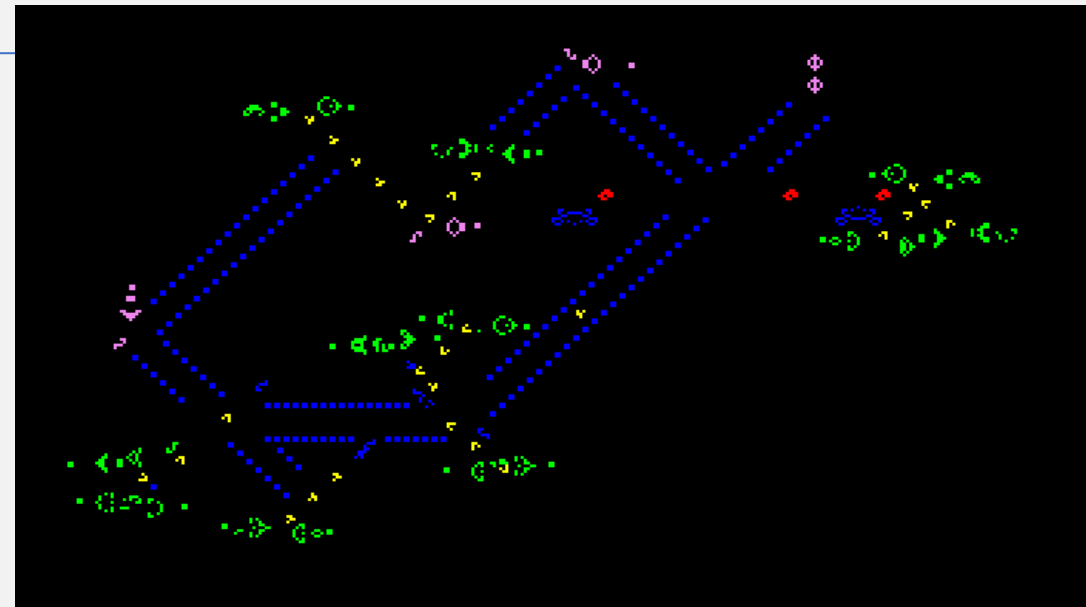


Conway's game of life

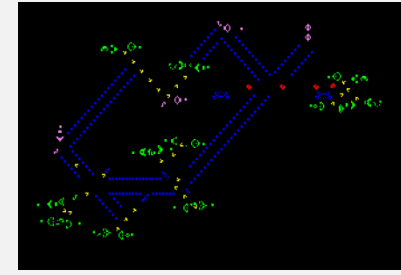
- Start with a few live cells
- See if cells somewhere else light up
- Make it so they only light up if some condition holds
- Just like a Turing machine going into accept state if some condition holds about its input



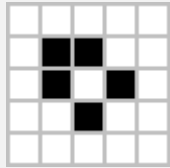
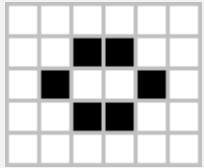
- Rules of the Game of Life:
 - Every cell can have up to 8 neighbours (above/below/left/right/diagonals)
- At every iteration
 - Every live cell with less than 2(live) neighbours dies
 - Every live cell with more than 3 neighbours dies
 - A dead cell with exactly 3 neighbours becomes alive
 - Rest stay the same



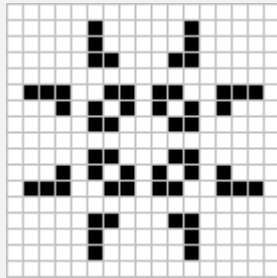
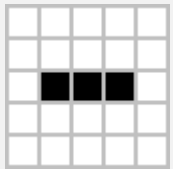
Conway's game of life: what can it do?



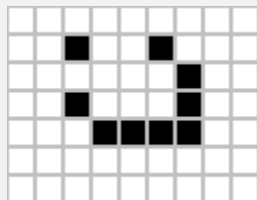
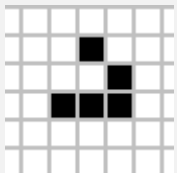
- Converge to a still pattern



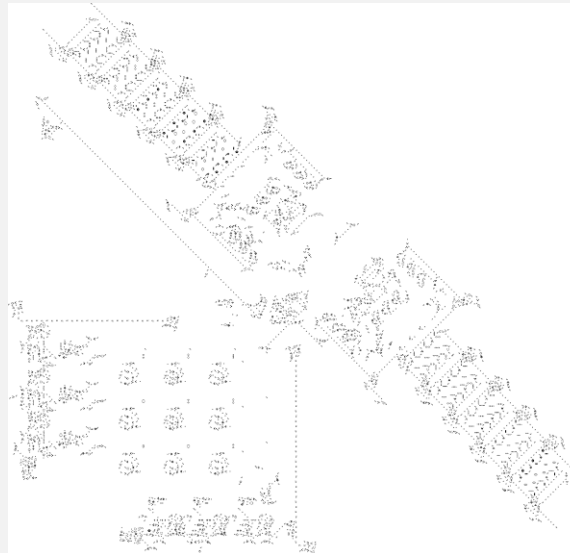
- Oscillate



- Create a moving pattern



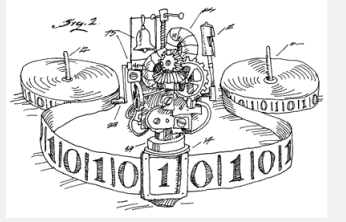
- Simulate a
Turing machine



- Rules of the Game of Life:
- At every step of the game:
 - Every live cell with less than 2 live neighbours dies
 - Every live cell with more than 3 live neighbours dies
 - A dead cell with exactly 3 live neighbours becomes alive
 - Rest stay the same



Everything we can call “computable” in any sense of this word is computable by a Turing machine.

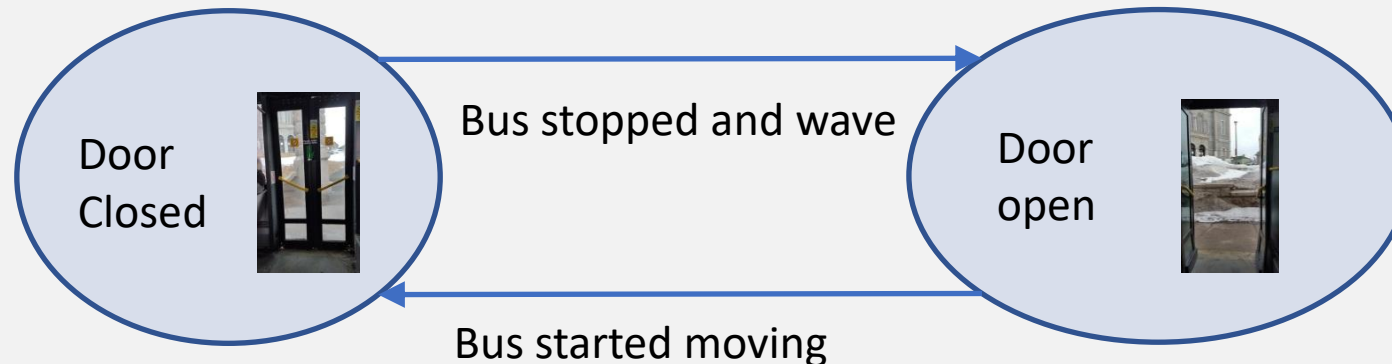
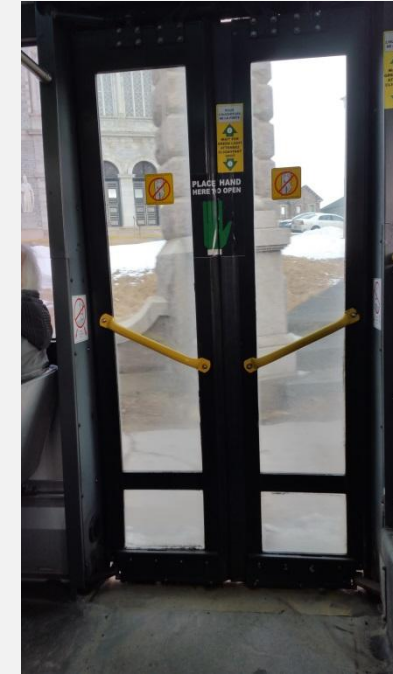


- But not everything is *equivalent* to a Turing machine
- Some models of computation are more limited:
 - Though they can be simulated **by** the Turing machine model, they cannot simulate TMs.



Finite state machine

- Metrobus door: wave to open.
 - Only works when bus has stopped.
 - Description of the system:
 - If bus is in motion then closed.
 - If bus is stopped then if wave received, open.
 - If bus is stopped and there is no wave, remain closed.
 - If the door is open, then if the bus started moving, close the door.





Finite state machine

- Finite state machine:

- States

- Including a start state and a set of finish states

- Inputs

- An input alphabet
 - Reads the input string symbol-by-symbol

- Transitions $States \times Inputs \rightarrow States$

- Sometimes also have outputs:
 - Then include output alphabet
 - Transitions to $States \times Outputs$

In the bus example

- Two states: closed and open.

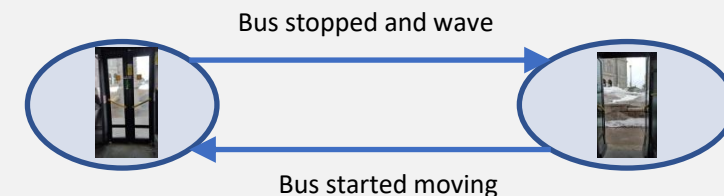
- Looks like closed is the start state.
 - In real life, probably more states needed.

- Input alphabet

- Bus moving/stopping, wave.

- Transitions:

- If closed and stoped and sensed a wave, go to open
 - If open and started moving, go to closed.





Definitions of DFAs

- Formally: a **deterministic finite automaton (DFA)** consists of $(\Sigma, Q, q_0, F, \delta)$, where
 - Σ is the input alphabet
 - Q is the set of states with
 - q_0 the start state
 - F a set of accepting (final) states.
 - $\delta: Q \times \Sigma \rightarrow Q$ a transition function
- A DFA **accepts** a string $w = w_1 \dots w_n$ iff there is a sequence of states $r_0 \dots r_n$ in Q where
 - $r_0 = q_0$ is the start state
 - $\forall i < n \quad \delta(r_i, w_{i+1}) = r_{i+1}$
 - $r_n \in F$
- A DFA **recognizes** (accepts) a language L iff it accepts all and only strings in L .



Definition of DFAs

- Example: binary strings with even number of 1s.

- $D = (\{0,1\}, \{s_1, s_2\}, s_1, \{s_1\}, \delta)$

- Input alphabet is $\Sigma = \{0,1\}$

- States $Q = \{s_1, s_2\}$

- s_1 is a start state

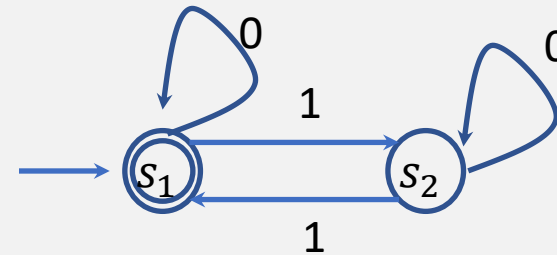
- Denoted by an arrow 

- s_1 is a single accepting state

- Denoted by a double circle 

- δ :

	0	1
s_1	s_1	s_2
s_2	s_2	s_1



Pattern matching

- Suppose we have a DNA string:
 - AAGATTCATTAATAAATACGCTTACA
 - And a gene string ATAC
 - How do we check if the string contains the match?



AAGATTCATATAATAAATACGCTTACA
ATAC

- Could just move along checking each letter, and if mismatch, shifting by 1 character...
 - There is a faster way: finite state machines.

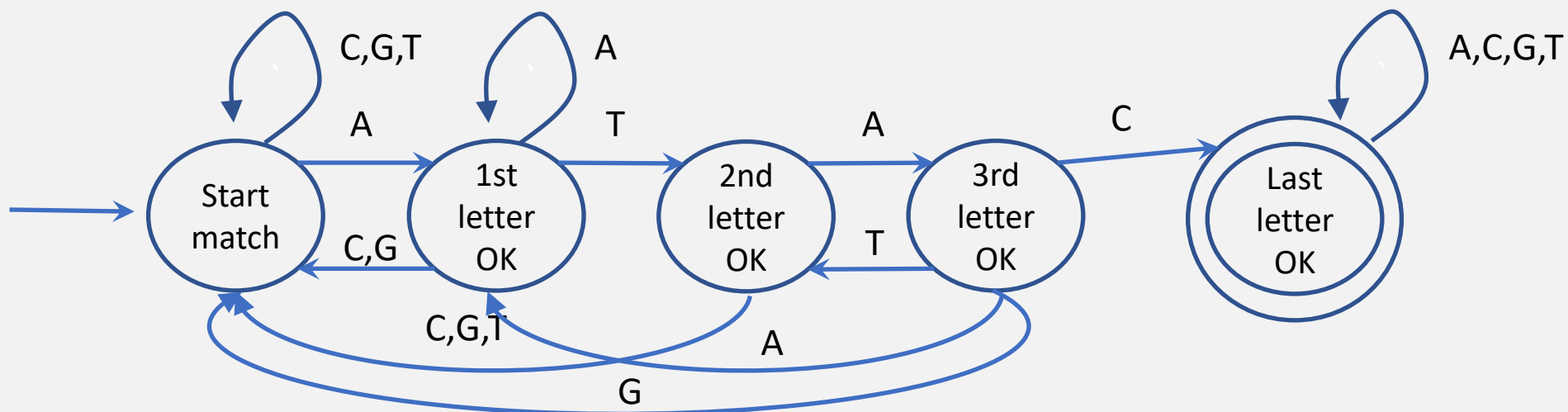


Matching with finite state machines

- Faster matching idea:

AAGATTTCATATAATAAATACGCTTACA
ATAC

- If mismatch T instead of C, know that shifting by 2 would be good enough; no need to re-match ATA





More examples of DFAs

- Strings containing a substring “00”
- Strings containing a subsequence “00”
-

Regular operations

- Build new languages out of existing ones with **regular operations**:
 - **Union** of two languages L_1 and L_2 is $L_1 \cup L_2$ (languages are sets of strings)
 - **Concatenation** of L_1 and L_2 is $L_1 \circ L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$
 - **Kleene star** of a language L is $L^* = \{x_1 \dots x_k \mid k \geq 0 \wedge \forall i \leq k \ x_i \in L\}$

Regular expressions

- A regular expression R is defined recursively as follows:
 - Basis:
 - R is a symbol a for some $a \in \Sigma$, or
 - $R = \lambda$, or
 - $R = \emptyset$
 - Recursion: if R_1, R_2 are regular expressions, then so are
 - $R_1 \cup R_2$,
 - $R_1 \circ R_2$,
 - R_1^* .

When writing regular expressions, often skip $\circ$, just write $R_1 R_2$. Also, use $|$ for $\cup$.

Regular expressions

- A regular expression recognizes (or describes) a language (matches strings that can be built using this recursive definition).
 - The class of languages recognized by regular expressions is called **regular languages**.
- Theorem: the class of languages recognized by DFAs is exactly the class of regular languages.
 - We will do a proof later.



- XKCD comics <https://xkcd.com/1313>