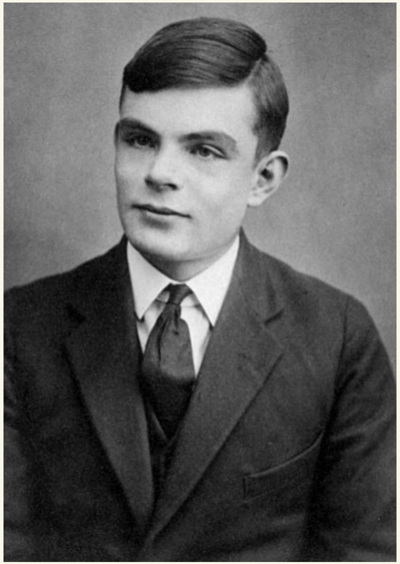
COMPUTABILITY

(Lecture notes: pp 54-65)

Turing Machines

"On computable Numbers, with an application to the Entscheidungsproblem"
1936



1912 - 1954

- Concept of 1st generally convincing general model of computation.
- Proved there is no algorithm for deciding truth in mathematics
- code breaking of Nazi ciphers WWII
- also worked in mathematical biology
- prosecuted in '52 for homosexuality

Turing Machines

$$M = \{Q, \Sigma, \Gamma, \delta, q_1, B, \{q_2\}\}$$

$Q = \{q_1, \ldots, q_k\}$ states, $k \geq 2$

Σ = finite input alphabet, including 0, 1

Γ = finite tape alphabet, $\Sigma \subseteq \Gamma$, includes "$b\!\!\!/$" (blank symbol)

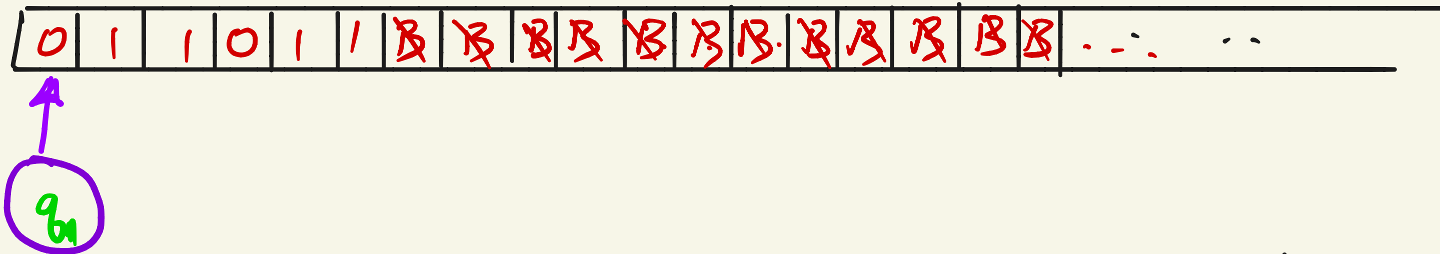
q_1 : start state

q_2 : halt state

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L \times R\}$

Turing Machines

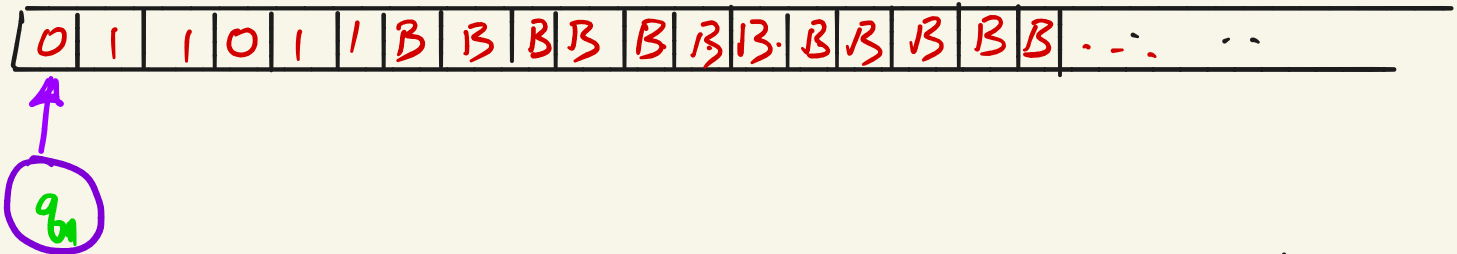
Input $x = 011011$



- Initially M is in start state q_1, input in 1st cells, then B's
- at any point in time, tape head points to some tape cell
  initally head points to left most cell

Turing Machines

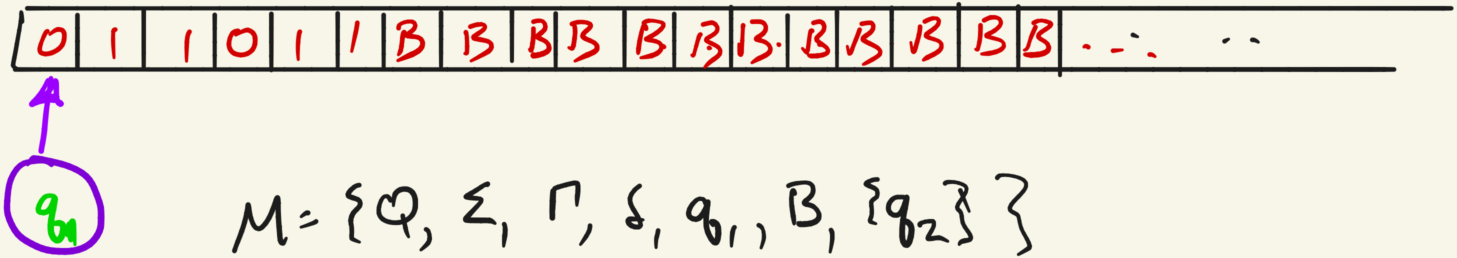
Input $x = 011011$



- Initially M is in start state q_1, input in 1st cells, then B's
- at any point in time, tape head points to some tape cell
  initally head points to left most cell
- at every time step, M makes one transition according to δ

Turing Machines

Input $x = 011011$



$$M = \{Q, \Sigma, \Gamma, \delta, q_1, B, \{q_2\}\}$$

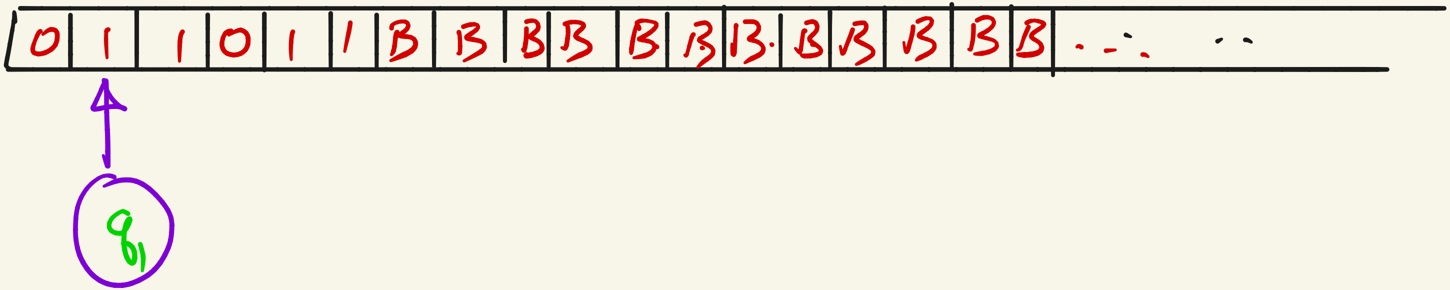
$$Q = \{q_1, q_2, q_3\}, \quad \Sigma = \{0, 1\}, \quad \Gamma = \{0, 1, B\}$$

δ:

$$(0, q_1) \to (0, q_1, R)$$
$$(1, q_1) \to (1, q_3, R)$$
$$(B, q_1) \to (B, q_1, R)$$
$$(0, q_3) \to (0, q_3, R)$$
$$(1, q_3) \to (1, q_2, R)$$
$$(B, q_3) \to (B, q_3, R)$$

Turing Machines

Input $x = 011011$

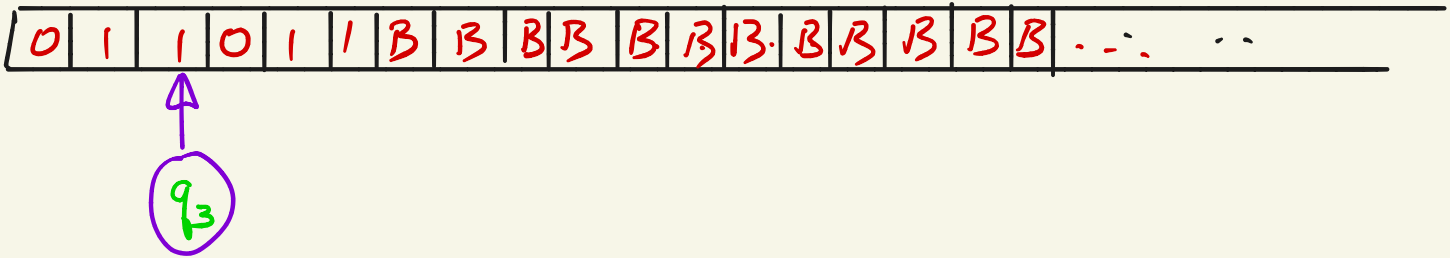


δ :

$$(0, q_1) \to (0, q_1, R)$$
$$(1, q_1) \to (1, q_3, R)$$
$$(B, q_1) \to (B, q_1, R)$$
$$(0, q_3) \to (0, q_3, R)$$
$$(1, q_3) \to (1, q_2, R)$$
$$(B, q_3) \to (B, q_3, R)$$

Turing Machines

Input $x = 011011$

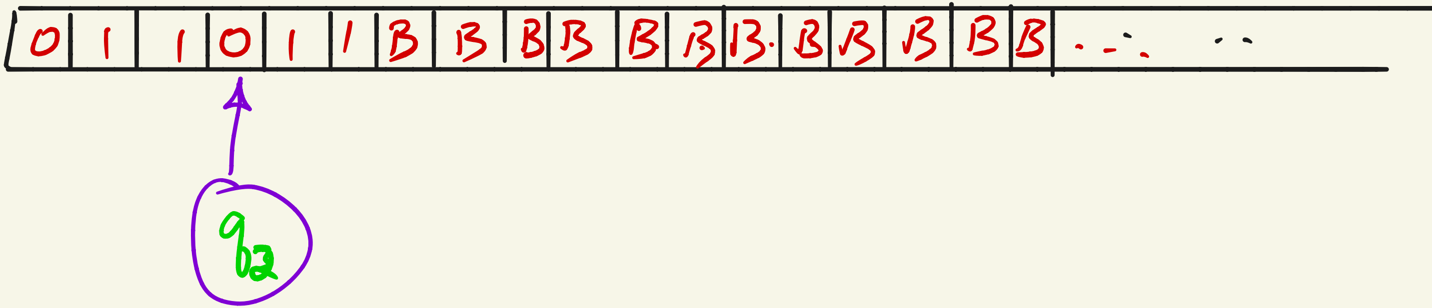


δ :

$$(0, q_1) \to (0, q_1, R)$$
$$(1, q_1) \to (1, q_3, R)$$
$$(B, q_1) \to (B, q_1, R)$$
$$(0, q_3) \to (0, q_3, R)$$
$$(1, q_3) \to (1, q_2, R)$$
$$(B, q_3) \to (B, q_3, R)$$

Turing Machines

Input $x = 011011$



δ :

$$(0, q_1) \to (0, q_1, R)$$
$$(1, q_1) \to (1, q_3, R)$$
$$(B, q_1) \to (B, q_1, R)$$
$$(0, q_3) \to (0, q_3, R)$$
$$(1, q_3) \to (1, q_2, R)$$
$$(B, q_3) \to (B, q_3, R)$$

Example : Count # 1's mod 2   Input $x = 011011$

0	1	1	0	1	1	B	B	B	B	B	B	B	B	B	B	B	B	. . . . . .

1. Start state $= q_1$
2. Scan Right until we see B : if in q_1 : # 1's so far is even
   if in q_3 : # 1's so far is odd
3. When we hit B : if in $q_1 \to q_4$, $q_3 \to q_5$
4. If in q_4 or q_5 scan left until we hit left end of tape
   when we hit left end if in q_4 overwrite with 0 + halt
   " " q_5 " " 1 + halt

Example : Count # 1's mod 2   Input $x = 011011$

0	1	1	0	1	1	B	B	B	B	B	B	B	B	B	B	B	B	...

1. Start state $= q_1$
2. Scan Right until we see B : q_3 : # 1's so far is even
   q_4 : # 1's so far is odd
3. When we hit B : if in $q_3 \to q_5$, $q_4 \to q_6$
4. If in q_5 or q_6 scan left until we hit left end of tape
   when we hit left end if in q_5 overwrite with 0 + halt
   " " q_6 " " 1 + halt

---

start: q_1, halt: q_2

$Q = \{q_1, q_2, q_3, \ldots q_6\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, *, B\}$

δ:

$(0, q_1) \to (*, q_3, R)$

$(1, q_1) \to (*, q_4, R)$

$(B, q_1) \to (*, q_3, R)$

$(0, q_3) \to (B, q_3, R)$

$(1, q_3) \to (B, q_4, R)$

$(0, q_4) \to (B, q_4, R)$

$(1, q_4) \to (B, q_3, R)$

$(B, q_3) \to (B, q_5, L)$

$(B, q_4) \to (B, q_6, L)$

$(B, q_5) \to (B, q_5, L)$

$(B, q_6) \to (B, q_6, L)$

$(*, q_5) \to (0, q_2, R)$

$(*, q_6) \to (1, q_2, R)$

Turing Machines

Turing Machines compute n-ary partial (or total) functions from $\mathbb{N}^n \to \mathbb{N}$ by encoding input/output as strings over Σ

Encoding of $(a_1, \dots, a_n) \in \mathbb{N}^n$ example

$$(3, 10, 8) : \underbrace{11}_{a_1 \text{ in binary}}\, 2\, \underbrace{1010}_{a_2 \text{ in binary}}\, 2\, \underbrace{100}_{a_3 \text{ in binary}}$$

separated by "2"

Let $\langle a_1, \dots, a_n \rangle$ be the encoding of $(a_1, \dots, a_n)$

Turing Machines

Turing Machines compute n-ary partial (or total) functions from $\mathbb{N}^n \to \mathbb{N}$ by encoding input/output as strings over Σ

TM M on input x halts when it enters halt state (q_2)

If M halts on x, the output y is the longest string on tape, containing only 0's and 1's

Turing Machines

Let $f : \mathbb{N}^n \to \mathbb{N}$ be a total function

M <u>computes</u> f if for every n-tuple $(a_1 .. a_n) \in \mathbb{N}^n$
M on input $\langle a_1, .., a_n \rangle$ outputs $f(a_1 .. a_n)$
(in binary)

If there is a TM M that computes f,
then f is a <u>total</u> <u>computable</u> <u>function</u>

Turing Machines (omitted this slide)

Let $f : (\mathbb{N} \cup \{\infty\})^n \to \mathbb{N} \cup \{\infty\}$ be a <u>partial</u> functio[n]

(so $f(c_1 \ldots c_n) = \infty$ if any $c_i = \infty$)

M computes f if for all $(a_1 .. a_n)$ in domain of f

M on input $\langle a_1 .. a_n \rangle$ outputs $f(a_1 .. a_n)$

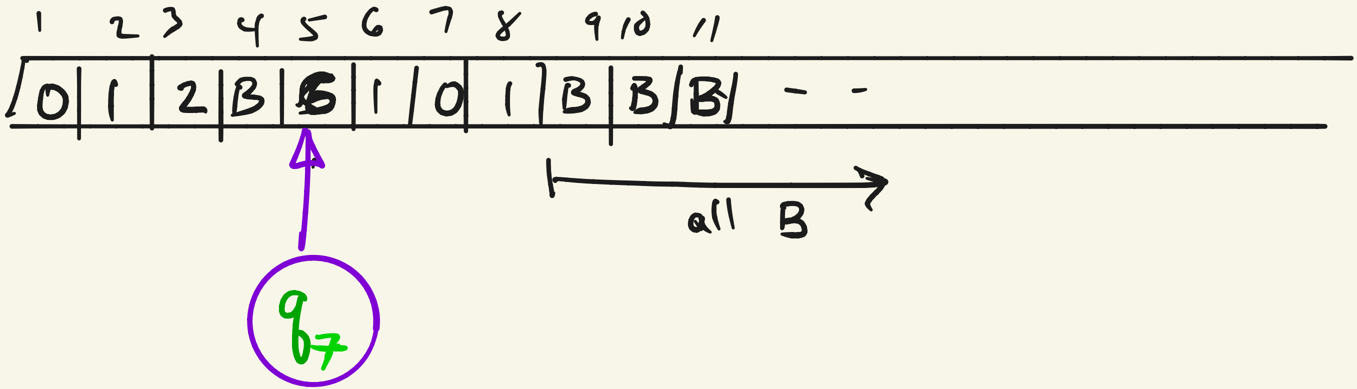
* M may not halt on inputs not in domain of f

If f (a partial function) is computed by some M

then f is a <u>computable partial function</u>

Turing Machine Configurations

- A configuration describes entire state of a TM at some point in time



Configuration : $0, 1, 2, B, (q_7, 5), 1, 0, 1$

Turing Machine Configurations

- A tableaux is a sequence of configurations describing running M on some input x

Turing Machine Configurations

- A tableaux is a sequence of configurations describing running M on some input x

$(0, q_1) \to (2, q_1, R)$

t=0	$(q_1, 0)$	0	1	1	0	2	B	. . .
t=1	2	$(q_1, 0)$	1	1	0	2	B	. . .
t=2	2	2	$(q_1, 1)$	1	0	2	B	. . .
t=3	2	2	2	$(q, 1)$	0	2	B	. .
t=4	2	2	2	2	$(q, 0)$	2	B	. .
t=5	2	2	2	2	2	$(q_1, 2)$	B	. .
t=6	2	2	2	2	2	2	(q_2, B)	. .

Turing Machine Configurations

- A tableaux is a sequence of configurations describing running M on some input x

At time $t = m$, tableaux is $m \times m$

$t=0$	$(q_1,0)$	0	1	1	0	2	B	. .
$t=1$	2	$(q_1,0)$	1	1	0	2	B	. . .
$t=2$	2	2	$(q_1,1)$	1	0	2	B	. . .
$t=3$	2	2	2	$(q,1)$	0	2	B	. .
$t=4$	2	2	2	2	$(q,0)$	2	B	. .
$t=5$	2	2	2	2	2	$(q_1,2)$	B	. .
$t=6$	2	2	2	2	2	2	(q_2,B)	. .

Encoding Turing Machines

$M = (\Sigma, Q, \Gamma, \delta, q_1, B, \{q_2\})$

Let $\Sigma = \{0, 1, 2\}$

$Q = \{q_1, q_2, \dots q_n\}$

$\Gamma = \{X_1, X_2, \dots X_k\}$ where $X_1 = 0$ $X_2 = 1$ $X_3 = 2$ $X_4 = B$

$D_1 = \text{left}$ $D_2 = \text{right}$

We represent transition $\delta(q_i, X_j) \to (q_k, X_l, D_m)$ by

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

Code for M: $111\, \text{code}_1\, 11\, \text{code}_2\, 11 \dots 11\, \text{code}_r\, 111$

where $\text{code}_1, \dots, \text{code}_r$ are the codes for transition function

Encoding Turing Machines

Example. $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, B\}$

$\delta(q_1, 1) = (q_3, 0, R)$ $\quad 0^1 1 0^2 1 0^3 1 0^1 1 0^2 \leftarrow c_1$

$\delta(q_3, 0) = (q_1, 1, R)$ $\quad 0^3 1 0 1 0^1 1 0^2 1 0^2 \leftarrow c_2$

$\delta(q_3, 1) = (q_2, 0, R)$ $\quad 0^3 1 0^2 1 0^2 1 0^1 1 0^2 \leftarrow c_3$

$\delta(q_3, B) = (q_3, 1, L)$ $\quad 0^3 1 0^3 1 0^3 1 0^2 1 0^1 \leftarrow c_4$

$M = 111 c_1 11 c_2 11 c_3 11 c_4 111$

$(M, 110110)$ encoded as $\underbrace{111 \overbrace{c_1 11 c_2 11 c_3 11 c_4}^{M} 111 \overbrace{110110}^{x}}_{\#(M, x)}$

* uniquely decodable

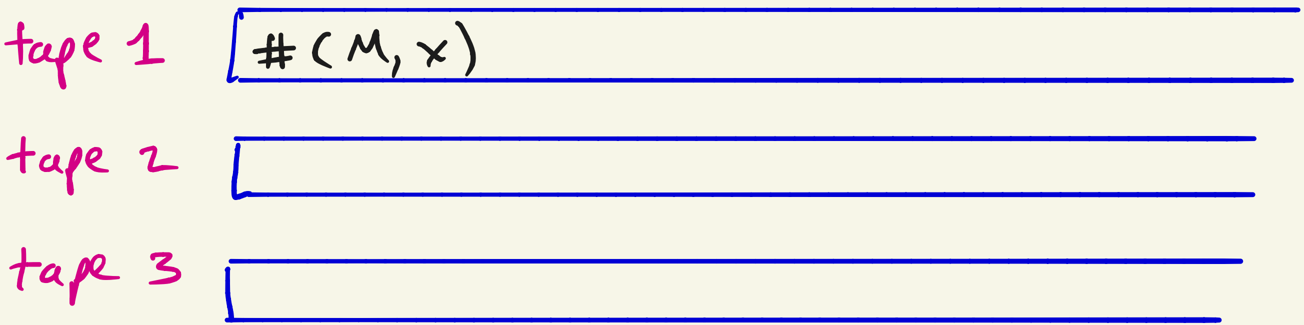
Universal Turing Machines

U: Takes as input $\#(M,x)$ and outputs y if
M on x halts and outputs y
if M does not halt on x, U does not halt on $\#(M,x)$

Universal Turing Machines

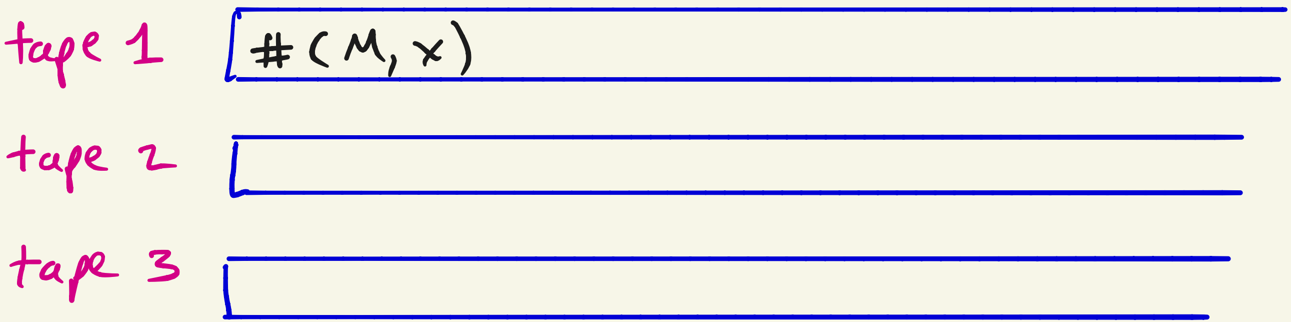
$\mathcal{U}$: Takes as input $\#(M,x)$ and outputs y if
M on x halts and outputs y
if M does not halt on x, $\mathcal{U}$ does not halt on $\#(M,x)$

We describe a 3-tape TM (at a high level) for $\mathcal{U}$.
(3-tapes can be simulated by one tape)



Universal Turing Machines

1. initial state



check that contents of tape 1 is legal encoding of M, x

Universal Turing Machines

(2)

tape 1 | $11 \$ \text{code}_1 \, 11 \, \text{code}_2 \, 11 \ldots 11 \, \text{code}_r \, 111$ | encoding of M

tape 2 | $\$ \, \underbrace{\dot{0}110}_{x}$ | contents of M's tape at start

tape 3 | $\$0$ | initial state of M

Initialize tapes 1 + 2 as above

and tape 3 to contain $\$0$

$\uparrow$ q_1 in binary

Universal Turing Machines

2

tape 1 $\quad$ 11 \$ $code_1$ 11 $code_2$ 11 ... 11 $code_r$ 111

tape 2 $\quad$ \$ x

tape 3 $\quad$ \$ 0

Loop

If tape 3 contains \$00 (halt state) halt and output contents of tape 2 (to 1st "B")

OW simulate next state:
Store contents of tape 2 head and current state of M in U's state. Scan tape 1 to find corresponding code, Modify tapes 2,3 accordingly

Universal Turing Machines

2

tape 1 | $11 \$ \text{code}_1\ 11\ \text{code}_2\ 11 \ldots 11\ \text{code}_r\ 111$

tape 2 | $\$\ 0\ 0\ 1\ 2\ 1\ 0\ 1\ B\ B \ldots$

tape 3 | $\$\ 0\ 0\ B\ B \ldots$

Say $\delta(q_2, 1) \rightarrow (q_3, 0, R)$

Universal Turing Machines

(2)

tape 1 | 11 \$ $code_1$ 11 $code_2$ 11 ... 11 $code_r$ 111

tape 2 | \$ 0 0 1 2 0 0 1 B B . . .

tape 3 | \$ 0 0 0 B . . . .

Say $\delta(q_2, 1) \rightarrow (q_3, 0, R)$

Notation

$\{x\}$ = Turing machine M such that $\#M = x$

$\{x\}_1$ = the unary function computed by x

$\{x\}_n$ = the n-ary function computed by x

(can generalize earlier so M takes n inputs instead of 1)

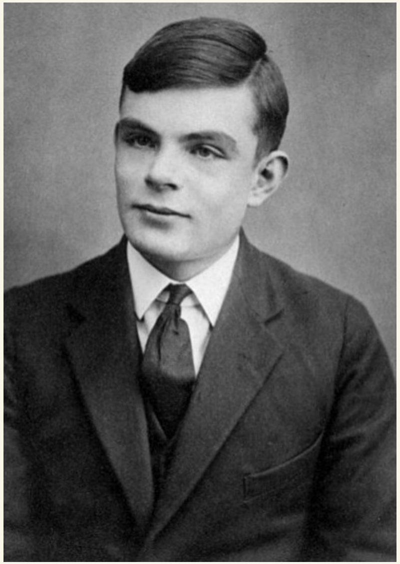
A **set** is a subset of $\mathbb{N}^n$ (usually $n=1$)

a set / relation / 0-1 valued total function:

$$A \subseteq \mathbb{N} \text{ then } A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

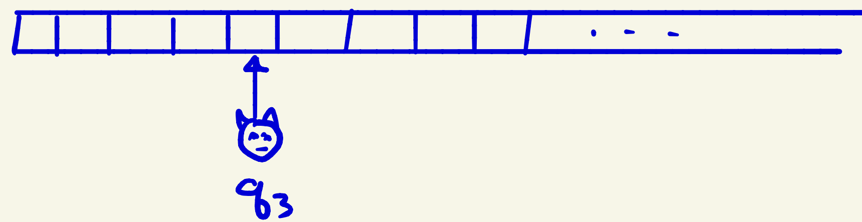
Turing Machines

"On computable Numbers, with an application to the Entscheidungsproblem"
1936



1912 - 1954

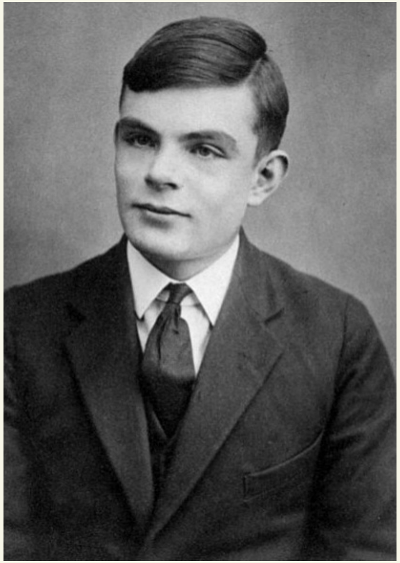
Turing Machines:



$$M = (Q, \Sigma, \Gamma, \delta, q_1, B, \{q_2\})$$

Turing Machines

"On computable Numbers, with an application to the Entscheidungsproblem"
1936



1912 - 1954

Church-Turing Thesis

A function/predicate is computable/realizable in physical world $\Longrightarrow$ it is computable by a TM

Notation

$\{x\}$ = Turing machine M such that $\#M = x$

$\{x\}_1$ = the unary function computed by x

$\{x\}_n$ = the n-ary function computed by x

(can generalize earlier so M takes n inputs instead of 1)

Today

What is computable and what isn't?

We will mostly focus on unary relations or languages – $L \subseteq \{0,1\}^*$

all finite length strings over $\{0,1\}$

$\{0,1\}^*$ = all binary strings of finite length
$= \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

Definition Let M be a TM, $\Sigma = \{0,1\}$

$\mathcal{L}(M) \subseteq \{0,1\}^*$ is the set of all (finite-length) strings $x \in \{0,1\}^*$ such that $M(x)$ halts and outputs 1

the language accepted by M

Recursive / RE Sets

recognizable / semi-computable

A language $L \subseteq \{0,1\}^*$ is recursively enumerable if there exists a TM M such that $\mathcal{L}(M) = L$

So $\forall x \in \{0,1\}^*$

$x \in L \Rightarrow$ M on x halts and outputs "1"

$x \notin L \Rightarrow$ M on x halts and does not output 1

or M does not halt on x

Recursive / RE Sets

A language $L \subseteq \{0,1\}^*$ is recursively enumerable if there exists a TM M such that $\mathcal{L}(M) = L$

So $\forall x \in \{0,1\}^*$

$x \in L \Rightarrow M$ on x halts and outputs "1"

$x \notin L \Rightarrow M$ on x halts and does not output 1

or M does not halt on x

recursively enumerable (r.e.) also called semidecidable, partial computable

Recursive / RE Sets

A language $L \subseteq \{0,1\}^*$ is recursive (computable / decidable) if there exists a TM M such that $\mathcal{L}(M) = L$ and M always halts

So $\forall x \in \{0,1\}^*$

$x \in L \Rightarrow$ M on x halts and outputs "1"

$x \notin L \Rightarrow$ M on x halts and does not output 1

(without loss of generality, $x \notin L \Rightarrow M(x)$ halts & outputs "0")

Recursive / RE Sets

A language $L \subseteq \{0,1\}^*$ is <u>recursive</u> if there exists a TM M such that $\mathcal{L}(M) = L$ and M always halts

So $\forall x \in \{0,1\}^*$

$x \in L \Rightarrow M$ on x halts and outputs "1"

$x \notin L \Rightarrow M$ on x halts and does not output 1

(without loss of generality, $x \notin L \Rightarrow M(x)$ halts & outputs "0")

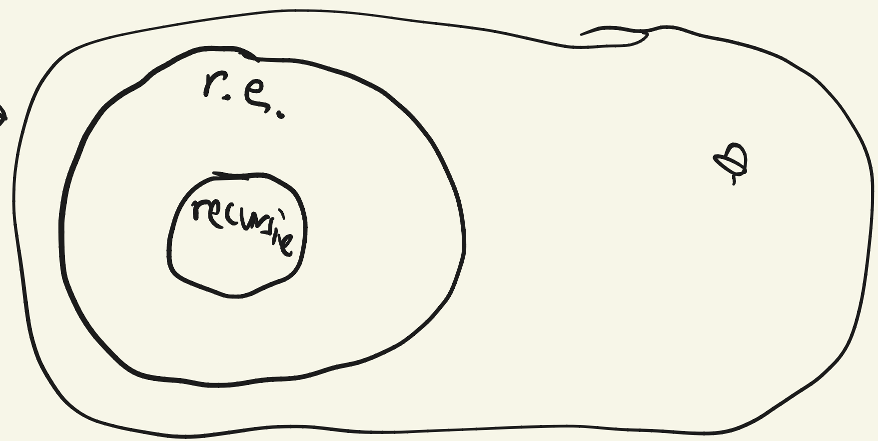
recursive also called decidable, computable

Recursive / RE Sets

A function $f: \{0,1\}^* \to \{0,1\}^*$ (or $f: \mathbb{N}^n \to \mathbb{N}$) is **total computable** if there exists a TM M such that $\forall x \in \{0,1\}^*$ $M(x)$ halts and outputs $f(x)$.

---

all $L \subseteq \{0,1\}^*$
all subsets of $\{0,1\}^*$ $\rightarrow$



CLOSURE PROPERTIES

1. L recursive $\Rightarrow$ L r.e.

2. Total computable functions closed under composition:

$$f, g \text{ computable} \Rightarrow f \circ g = f(g(x)) \text{ is computable}$$

CLOSURE PROPERTIES

1. L recursive $\Longrightarrow$ L r.e.

2. Total computable functions closed under composition:

   f, g computable $\Longrightarrow$ $f \circ g = f(g(x))$ is computable

3. Closure of recursive languages under $\cap, \cup, \neg$:

   L_1, L_2 recursive $\Longrightarrow$ $L_1 \cup L_2$, $L_1 \cap L_2$, $\neg L_1$, $\neg L_2$ are recursive

CLOSURE PROPERTIES

(1) L recursive $\Rightarrow$ L r.e.

(2) Total computable functions closed under composition:

f, g computable $\Rightarrow$ $f \circ g = f(g(x))$ is computable

(3) Closure of recursive languages under $\cap, \cup, \neg$:

L_1, L_2 recursive $\Rightarrow$ $L_1 \cup L_2$, $L_1 \cap L_2$, $\neg L_1$, $\neg L_2$ are recursive

(3') Closure of r.e. languages under $\cap, \cup$

L_1, L_2 r.e. $\Rightarrow$ $L_1 \cup L_2$, $L_1 \cap L_2$ are r.e.

use dovetailing ie for $i = 1, \ldots$
run M_1 for i steps, M_2 for i steps

CLOSURE PROPERTIES

1. L recursive $\Rightarrow$ L r.e.

2. Total computable functions closed under composition:

   f, g computable $\Rightarrow$ $f \circ g = f(g(x))$ is computable

3. Closure of recursive languages under $\cap, \cup, \neg$:

   L_1, L_2 recursive $\Rightarrow$ $L_1 \cup L_2$, $L_1 \cap L_2$, $\neg L_1$, $\neg L_2$ are recursive

3'. Closure of r.e. languages under $\cap, \cup$

   L_1, L_2 r.e. $\Rightarrow$ $L_1 \cup L_2$, $L_1 \cap L_2$ recursive

   use dovetailing re for $i = 1, \ldots$
   run M_1 for i steps, M_2 for i steps

What about closure of r.e. under $\neg$?

CLOSURE PROPERTIES, cont'd

④ L r.e., and $\overline{L}$ r.e. $\Rightarrow$ L is recursive

$\{x \mid x \notin L\}$

* <u>Note</u>: Often $L \subseteq \{0,1\}^*$ is a set of encodings. Example $L = \{x \mid \{x\}_1 \text{ accepts input "111"}\}$ then we usually think of $\overline{L}$ as $\{x \mid \{x\}_1 \text{ does not accept } 111\}$ although technically

$\overline{L} = \{x \mid x \text{ is not a legal encoding or } \{x\}_1 \text{ does not accept } 111\}$

$L \subseteq \{0,1\}^*$

$\{0,1\}^*$ = set of all strings over 0/1 of finite length

$\{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots \ldots\}$

$\overline{L} = \{ y \in \{0,1\}^* \mid y \notin L \}$

CLOSURE PROPERTIES, cont'd

④ L r.e., and $\bar{L}$ r.e. $\Rightarrow$ L is recursive

Proof: (Dovetailing)

Let M_1 be a TM st $\mathcal{L}(M) = L$,
M_2 be a TM st $\mathcal{L}(M) = \bar{L}$

New TM M on x:

For $i = 1, 2, 3, \ldots$

Run M_1 on x for i steps
- if M_1 accepts x halt + accept

Run M_2 on x for i steps
- if M_2 accepts x, halt + reject

CLOSURE PROPERTIES, cont'd

④ L r.e., and $\bar{L}$ r.e. $\Rightarrow$ L is recursive

Proof: (Dovetailing)

Let M_1 be a TM st $\mathcal{L}(M) = L$,
M_2 be a TM st $\mathcal{L}(M) = \bar{L}$

New TM M on x:

For $i = 1, 2, 3, \ldots$

Run M_1 on x for i steps
if M_1 accepts x halt + accept

Run M_2 on x for i steps
if M_2 accepts x, halt + reject

- M on x eventually halts since x accepted by exactly one of M_1, M_2
- $x \in L \Rightarrow M_1$ accepts $x \Rightarrow M$ accepts x
- $x \notin L \Rightarrow M_2$ accepts $x \Rightarrow M$ halts and rejects x

Many Languages aren't Recursively Enumerable!

Intuition: $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

- Every TM M maps uniquely to a string in $\{0,1\}^*$, corresponding to the Language $L \subseteq \{0,1\}^*$ accepted by M
- Since the set of all TMs is countable, so are the re languages $L \subseteq \{0,1\}^*$
- On the other hand, the set of all languages over $\{0,1\}^*$ is <u>uncountable</u>

  (so there exist many languages $L \subseteq \{0,1\}^*$ that are not r.e.)

Many Languages are Not r.e.

Proof : Diagonalization

Main idea : There are many more Languages (subsets of $\{0,1\}^*$) than there are TMs.

Proof very similar to Cantor's argument showing that there is NO 1-1 mapping from the Real numbers to the Natural numbers

Many Languages are Not r.e.

Proof : Diagonalization

- Fix an enumeration of all TMs with $\Sigma = \{0,1\}$

  $\{x_1\}, \{x_2\}, \{x_3\}, \ldots$

- Make a 2-way infinite (but countable) table

  rows correspond to $\{x_1\}, \{x_2\}, \ldots$

  columns correspond to enumeration of encodings of Turing machines $x_1, x_2, \ldots$

- Entry (i,j): 0 if $\{x_i\}$ accepts x_j

  1 otherwise

Many Languages are Not r.e.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\cdots$
M_1 $\{x_1\}$	0	1	1	0	1	0	0	$\cdots$
M_2 $\{x_2\}$	0	0	1	1	0	1	1	
$\{x_3\}$	1	1	1	1	1	0	1	
	1	1	0	0	0	0	1	
	0	0	0	0	1	1	1	
	0	1	0	1	0	1	0	
	$\vdots$							

Many Languages are Not r.e.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\cdots$
M_1	1	1	1	0	1	0	0	$\cdots$
M_2	0	0	1	1	0	1	1	
M_3	1	1	0	1	1	0	1	
M_4	1	1	0	1	0	0	1	
M_5	0	0	0	0	0	1	1	
$\vdots$	0	1	0	1	0	0	0	

Diagonal Language D

$$D = \{x_j \mid \{x_j\}_1 \text{ does not accept } x_j\}$$

ie $D = \{x \in \{0,1\}^* \mid x \text{ is a legal encoding of a TM and } \{x\}_1 \text{ does not accept } x\}$

Theorem D is not r.e.

Proof By construction: For all TMs M_i,

$\{x_i\}(x_i) \neq D(x_i)$ so $\mathcal{L}(M_i) \neq D$

$\therefore D$ not r.e.

Theorem D is not r.e.

Proof By construction: For all TMs M_i,

$\{x_i\}(x_i) \neq D(x_i)$ so $\mathcal{L}(M_i) \neq D$

$\therefore$ D not r.e.

$\bar{D} = \{x \in \{0,1\}^* \mid$
either x is not a legal coding of a TM or $\{x\}$, accepts $\{x\}$

Theorem $\bar{D} = \{x \in \{0,1\}^* \mid x \notin D\}$

$\bar{D}$ is r.e.

(Weak) Incompleteness of PA

- PA has a recursive set of axioms, so the set of sentences in PA is r.e. (by completeness theorem, which gives an r.e. procedure for deciding if $A \in PA$)

- Given $x \in \{0,1\}^*$ Let $num(x) \in \mathbb{N}$ be the number encoded by x and conversely let $bin(n) \in \{0,1\}^*$ be the binary encoding of n

- Let $F_D(n) \subseteq \mathbb{N}$ be the relation corresponding to D (our "diagonal language")

  $F_D(n) = 1$ iff $D(bin(n)) = 1$ iff $\{bin(n)\}_1$ does not accept $bin(n)$

- For all $n \in \mathbb{N}$, there is a sentence A_n (over $\mathcal{L}_{PA}$) such that $A_n \in TA \iff F_D(n) = 1$

  $\neg A_n \in TA \iff F_D(n) = 0$

since $\overline{D}$ is r.e.

claim $\exists n \in \mathbb{N}$ s.t. either $A_n \in TA$, but $A_n \notin PA$

or $\neg A_n \in TA$ but $\neg A_n \notin PA$

(Weak) Incompleteness of PA

- PA has a recursive set of axioms, so the set of sentences in PA is r.e. (by completeness theorem, which gives an r.e. procedure for deciding if $A \in PA$)
- Given $x \in \{0,1\}^*$ Let $num(x) \in \mathbb{N}$ be the number encoded by x and conversely let $bin(n) \in \{0,1\}^*$ be the binary encoding of n
- Let $F_D(n) \subseteq \mathbb{N}$ be the relation corresponding to D (our "diagonal language")

  $F_D(n) = 1$ iff $D(bin(n)) = 1$ iff $\{bin(n)\}_1$ does not accept $bin(n)$
- For all $n \in \mathbb{N}$, there is a sentence A_n (over $\mathcal{L}_{PA}$) such that

  $A_n \in TA \iff F_D(n) = 1$

  $\neg A_n \in TA \iff F_D(n) = 0$

$$\boxed{\textbf{claim} \quad \exists n \in \mathbb{N} \text{ s.t. either } A_n \in TA, \text{ but } A_n \notin PA \text{ or } \neg A_n \in TA \text{ but } \neg A_n \notin PA}$$

otherwise D is r.e.

Using Reductions to show other (more natural) languages/functions are not computable/recursive/r.e.

High Level:

1. Say we know L_1 not recursive
   To show L_2 not recursive, design a TM M_1 always halts & $\mathcal{L}(M_1) = L_1$, assuming a TM M_2 that always halts & $\mathcal{L}(M_2) = L_2$

2. Suppose L_1 not r.e.
   To show L_2 not r.e., construct M_1 st $\mathcal{L}(M_1) = L_1$ assuming a TM M_2 st $\mathcal{L}(M_2) = L_2$

K and HALT are r.e. but not recursive

$$\bar{D} \approx K \stackrel{d}{=} \{ x \mid \text{TM } \{x\} \text{ halts on input } x \}$$

$$\text{HALT} \stackrel{d}{=} \{ \langle x, y \rangle \mid \text{TM } \{x\} \text{ halts on input } y \}$$

<u>Claim</u> HALT, K are both r.e.

<u>Pf</u>: simply run $\{x\}$ on y. Accept if simulation halts.

Theorem K is not recursive

$$K \stackrel{d}{=} \{ x \mid \text{TM } \{x\} \text{ halts on input } x \}$$

<u>Proof</u> Let $L_1 = D$. We know L_1 is not r.e.
Assume $L_2 = K$ is recursive, + Let M_2 always halt + $\mathcal{L}(M_2) = L_2$

<u>Construction of TM M_1 for D on input x:</u>

Run M_2 on x

- If M_2 accepts x then
  Run $\{x\}$ on x and output 1 iff $\{x\}(x) \neq 1$
- If M_2 halts + does not accept x then output 1

Theorem K is not recursive

$$K \stackrel{d}{=} \{ x \mid TM\ \{x\} \text{ halts on input } x \}$$

Proof Let $L_1 = D$. We know L_1 is not r.e.

Assume $L_2 = K$ is recursive, + let M_2 always halt + $\mathcal{L}(M_2) = L_2$

Construction of TM M_1 for D on input x:

Run M_2 on x

- If M_2 accepts x then
  Run $\{x\}$ on x and output 1 iff $\{x\}(x) \neq 1$
- If M_2 halts + does not accept x then output 1

- M_1 halts on all x
- $x \in D \Rightarrow \{x\}(x) \neq 1 \Rightarrow M_1(x) = 1$
- $x \notin D \Rightarrow \{x\}(x) = 1 \Rightarrow M_1(x) \neq 1$

Theorem HALT is Not recursive

$$K \overset{d}{=} \{ x \mid \text{TM } \{x\} \text{ halts on input } x \}$$

$$HALT \overset{d}{=} \{ \langle x, y \rangle \mid \text{TM } \{x\} \text{ halts on input } y \}$$

Proof $\frac{\text{idea}}{K}$ is a special case of HALT & K Not recursive

show K reduces to HALT:

Let $L_1 = K$, $L_2 = HALT$. Assume M_2 always halts and accepts L_2.

Construct M_1 for L_1:

$\underline{M_1 \text{ on } x:}$

Run M_2 on $\langle x, x \rangle$. Accept iff M_2 accepts

Theorem $\overline{K}$ is not r.e. $\longleftarrow \overline{\mathbb{C}} \approx D$

$$K \stackrel{d}{=} \{ x \mid \text{TM } \{x\} \text{ halts on input } x \}$$

Pf We saw already that K is r.e. but not recursive

If $\overline{K}$ is r.e., then both $K, \overline{K}$ are r.e.,

so by property (4) $\Rightarrow$ K recursive

$\therefore$ $\overline{K}$ not r.e.

Tips

(1.) Try obvious algorithms to see if you think language is recursive, re, or neither

(2.) To show L not r.e., sometimes it helps to work with $\bar{L}$

(ie. if $\bar{L}$ r.e., + $\bar{L}$ not recursive then L not r.e.)

(3) get reduction in correct direction.

Many times constructed TM M_1 will ignore its own input

SUMMARY SO FAR

1. We saw $D = \{x \mid \{x\}_1(x) \text{ does not accept}\}$

is not r.e. by diagonalization

2. Using reductions we have:

- K, Halt are not recursive (but both are r.e.)

- $\overline{K}$, $\overline{\text{HALT}}$ are not r.e.

Another example: $L = \{ x \mid \{x\} \text{ accepts at least one input} \}$

L is r.e. but not recursive.

$\boxed{L \text{ is r.e.:}}$ Enumerate all strings in $\{0,1\}^*$

$\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$

$w_1 \quad w_2 \quad w_3$

Dovetail Procedure for L on input x:

For $i = 1, 2, 3, \ldots$

For $j = 1, \ldots i$

Simulate $\{x\}$, on w_j for i steps

If any of the simulations accepts, HALT & accept

Another example: $L = \{ x \mid \{x\} \text{ accepts at least one input} \}$

$\boxed{\text{L is not recursive:}}$

$L_1 = K = \{ y \mid \{y\}(y) \text{ halts} \}$

Assume $L_2 = L$ is recursive & let M_2 be TM $\mathcal{L}(M_2) = L$
and M_2 always halts

M_1 on input y:

Construct encoding z of TM $\{z\}$ where
$\{z\}$ on input x: Ignores x & runs $\{y\}$ on y
and accepts x if $\{y\}(y)$ halts

Run M_2 on z and accept y iff $M_2(z)$ accepts

Claim $\mathcal{L}(M_1) = K$ and M_1 always halts

$y \in K \Rightarrow \{y\}(y)$ halts $\Rightarrow \{z\}$ accepts all inputs $\Rightarrow M_2(z) = 1 \Rightarrow M_1(y) = 1$

$y \notin K \Rightarrow \{y\}(y)$ doesn't halt $\Rightarrow \{z\}$ accepts No input $\Rightarrow M_2(z) \neq 1 \Rightarrow M_1(y) \neq 1$