Turing Machines.

States & Transitions:

Read write move left

$$q_1 \xrightarrow{a \to b,\ L} q_2$$

move right.

$$q_1 \xrightarrow{a \to b,\ R} q_2$$

The head at each transition (time step):

① Reads a symbol
② Writes a symbol
③ Moves left or right.

Head starts at the leftmost position of the input string

Turing Machines are deterministic. (no ε-transitions allowed).

The machine halts in a state if there is no transition to follow.

Accepting states have no outgoing transitions. $q_1 \longrightarrow q_2$ X not allowed.

If machine halts in an accept state: accept input string.

If machine halts in a non-accept state or if machine enters an infinite loop:

Reject input string.

In order to accept an input string, it is not necessary to scan all the symbols of the input string.

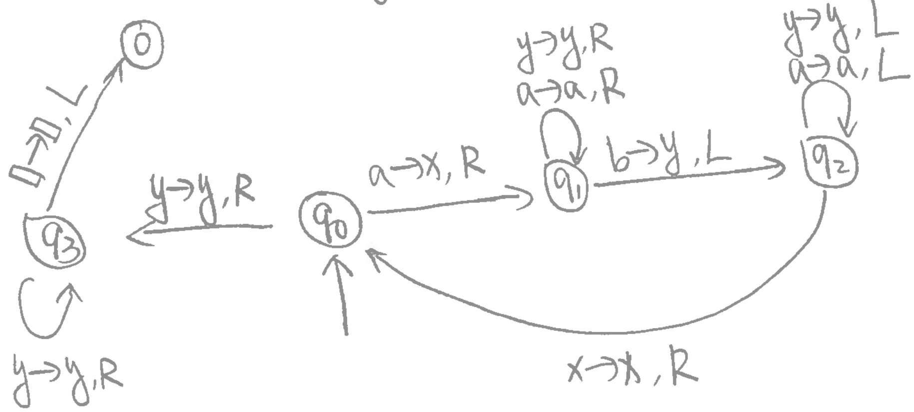
The accepted language: for any Turing Machine M:

$$L(M) = \{w : \underbrace{q_0}_{\text{initial state}} w \xrightarrow{*} x_1 \underbrace{q_f}_{\text{Accept state.}} x_2\}.$$

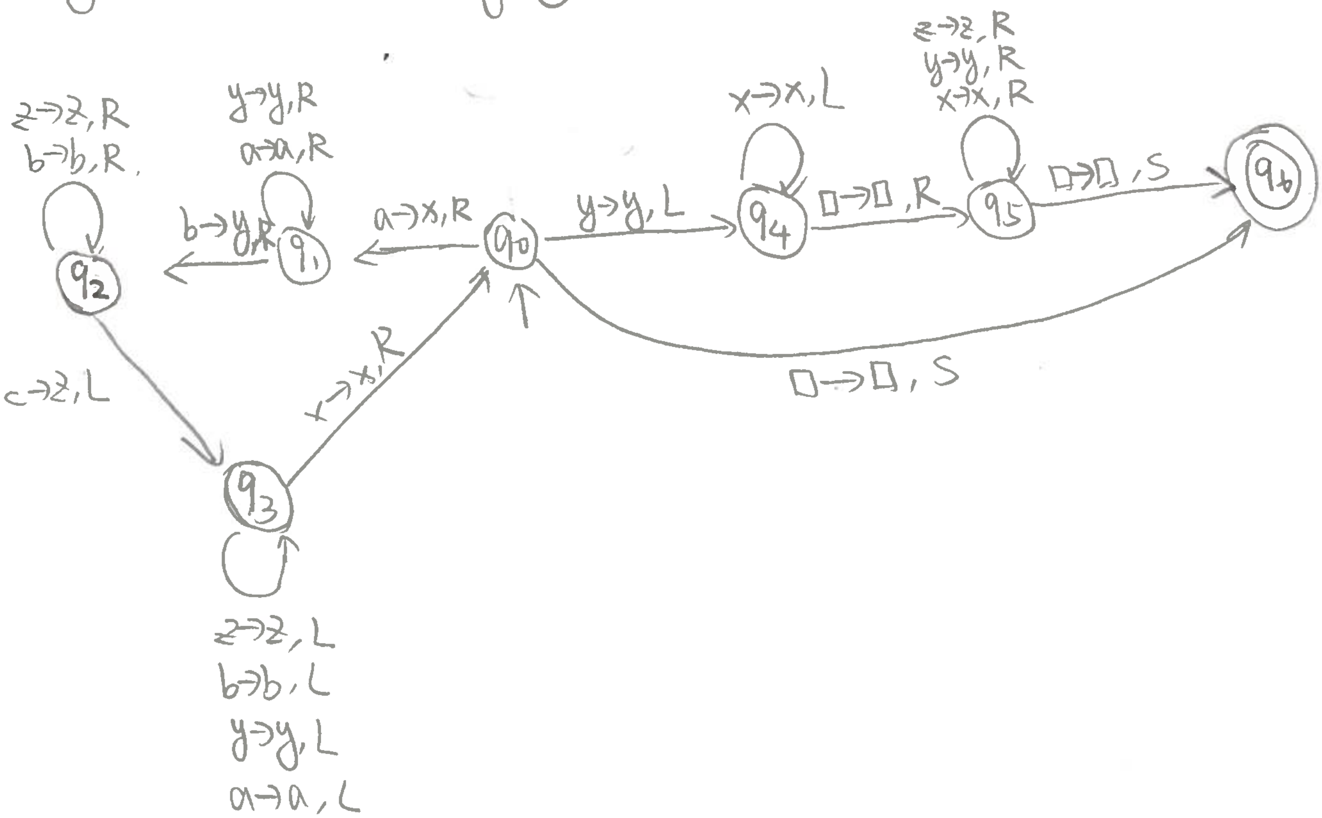
If a language L is accepted by a Turing machine M, then we say that L is: Turing recognizable or Turing Acceptable or Recursively Enumerable.

Turing Machine

Turing machine for the language $\{a^n b^n\}$, $n \geq 1$.



Turing machine for the language $\{a^n b^n c^n, \ n \geq 0\}$.



Formal Definitions for Turing Machines.

□□ c a b a □□ → instantaneous description: $c a q_1 b a$

$\uparrow$
q_1

For any turing machine M. $L(M) = \{w : q_0 w \overset{*}{\vdash} x_1 q_f x_2\}$

q_0: initial state
q_f: Accept state

Variations of the Turing Machine

An algorithm for a problem is a Turing Machine which solves the problem.

The standard model: $\begin{cases} \text{Infinite tape.} \\ \text{Read-Write Head (Left or Right)} \\ \text{Control units are deterministic} \end{cases}$

Variations of the standard Model:
① Stay-option
② Semi-Infinite Tape
③ Multitape
④ Multidimensional
⑤ Nondeterministic.

can simulate standard Turing machine and vice versa.

Each new class has the same power with standard Turing Machine.

M_1 M_2

$L(M_1) = L(M_2)$.

Same power of two machine classes: both classes accept the same set of language.

To proof the variations of the Turing machine have the same power with Standard Turing machines:

Step 1: variation machines simulate standard Turing machines

Step 2: Standard Turing machines simulate variation machines.

Theorem: stay-option machines have the same power with standard Turing machines.

Prof: step 1: stay-option machines simulate standard Turing machines.
This is trivial since any standard turing machine is also a stay-option machine.

Step 2: standard turing machines simulate stay-option machines.

The standard turing machines only have left and right head options. We need to use them to simulate the "stay" head option.

$$q_1 \xrightarrow{a \to b,\, S} q_2 \iff q_1 \xrightarrow{a \to b,\, L} \bigcirc \xrightarrow{x \to x,\, R} q_2$$

any possible x.

For other transitions nothing changes.

End of proof.

Theorem: Semi-infinite machines have the same power with Standard Turing machines.

Proof:

Step 1: Semi-infinite machines simulate standard turing machines.

Given a standard machine, find a reference point. i.e: □ a, b, c, d, e, □, □

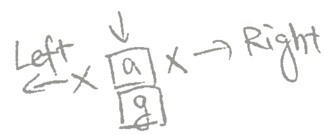
Ref point.

Then using semi-infinite tape machine with two tracks:

Right part: #, d, e, □, □, ···

Left part: # c, b, a, □, ···

Therefore, given standard machine with reference point: $q_1 \xrightarrow{a \to g,\, R} q_2$



Semi-infinite tape machines:

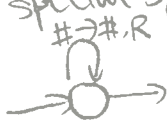
Right part: $q_1 \xrightarrow{(a,x) \to (g,x),\, R} q_2$

Left part: $q_1 \xrightarrow{(x,a) \to (x,g),\, L} q_2$ For all tape symbols x.

Step 2: Standard Turing machines simulate semi-infinite machines.

For standard machines, insert special symbol # at left of input string.

Then add a self-loop.



End of Proof.

Theorem: Multi-tape machines have the same power with standard Turing machines.

Proof: step 1: multi-tape machines simulate standard Turing machines.

This step is trivial, just use one tape.

step 2: Standard turing machines simulate multi-tape machines.

standard machines with multi-track tape.

Repeat for each multi-tape state transition:

① Return to reference point.

② Find current symbol in track 1 and update.

End of Proof.

same power doesn't imply same speed.

Theorem: Multidimensional machines have the same power with standard Turing machines.

Proof: step 1: Multidimensional machines simulate standard Turing machines

This step is trivial, just use one dimension.

Step 2: Standard Turing machines simulate multidimensional machines.

For a standard machine.

①. Use a two track tape.

②. store symbols in track 1

③. store coordinates in track 2.

Repeat for each transition followed in the 2-dimensional machine:

① update current symbol. ② compute coordinates of next position,

③ Find next position on tape.

end of proof

Theorem: Nondeterministic machines have the same power with standard Turing machines.

Proof: step 1. Nondeterministic machines simulate standard Turing machines.

Trivial, every deterministic machine is also nondeterministic.

step 2. Standard turing machines simulate nondeterministic machines.

For standard (Deterministic) machine:

① Uses a 2-dimensional tape, which has been proved to be equivalent to standard turing machine with one tape.

② Stores all possible computations of the non-deterministic machine on the 2-dimensional tape.

Then the deterministic turing machine simulates all possible computation paths: simultaneously, step-by-step, with breadth-first search.

until the input string is accepted or rejected in all configurations.

If the non-deterministic machine accepts the input string, then the deterministic machine accepts and halts, too.

If the non-deterministic machine does not accept the input string:

If all paths reach a halting state.

Case 1: The simulation halts if all paths reach a halting state.

or

Case 2: The simulation never terminates (infinite loop)

In either case, the deterministic machine rejects too, by halting, or by simulating the infinite loop.

End of proof.

A Universal Turing Machine.

Universal Turing Machine:

① Reprogrammable machine

②. Simulates any other Turing Machine.

We describe Turing machine M as a string of symbols: we encode M as a string of symbols.

A turing machine is described with a binary string of 0's and 1's.
Therefore, the set of turing machines form a language: each string of this language is the binary encoding of a Turing machine.

Infinite sets are either countable or uncountable.

Countable set: there is one-to-one correspondence of elements of the set to positive integers (1, 2, 3, ...).

Let S be a set of strings, an enumerator for S is a turing machine that generates (prints on tape) all the strings of S one-by-one, and each string is generated in finite time.

For a set of S, if there is an enumerator, then set S is countable.

Theorem: if S is an infinite countable set, then the powerset 2^S of S is uncountable.

Proof: since S in countable, we can list its elements in some order.

$$S = \{s_1, s_2, \cdots\}.$$

Then elements of the powerset 2^S have the form:

$\emptyset$
$\{s_1, s_2\}$ } they are subsets of S.

We encode each subset of S with a binary string of 0s and 1s.
Then every infinite binary string corresponds to a subset of S.

Assume the powerset 2^S is countable. Then we can list the elements of the powerset in some order.

$$2^S = \{t_1, t_2, t_3, \cdots\}.$$

$$\begin{array}{cccccccc}
t_1 & 1 & 0 & 0 & 0 & 0 & \cdots \\
t_2 & 1 & 1 & 0 & 0 & 0 & \cdots \\
t_3 & 1 & 1 & 0 & 1 & 0 & \cdots \\
 & - & - & - & & &
\end{array}$$

Let t = the binary string whose bits are the complement of the diagonal.

$= 001\cdots$

Then t should correspond to a subset of S : $t \in 2^S \Leftrightarrow t = t_i$ for som

But $t \neq t_i$ for every i since they differ in the ith bit.

We got contradiction here.

Therefore the powerset 2^S is uncountable.

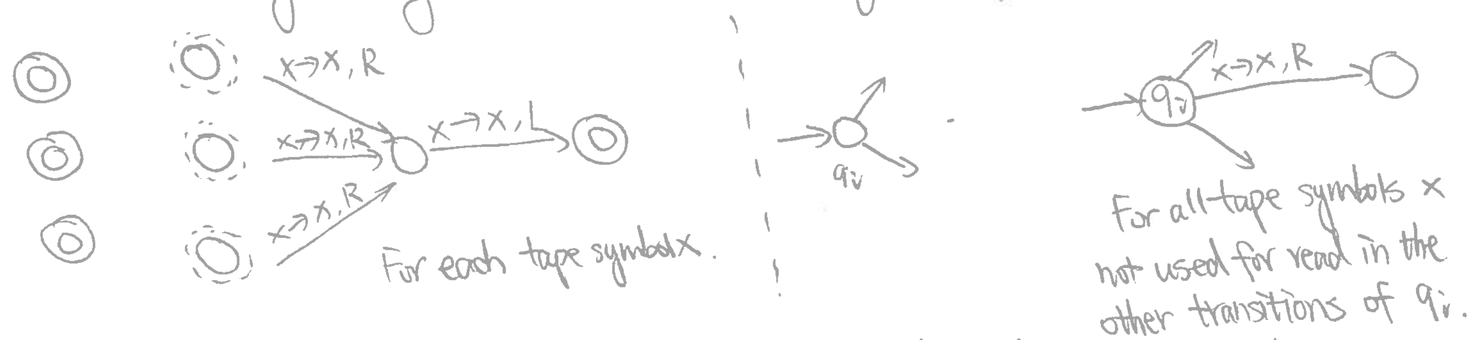
End of proof.

Decidable Languages.

A language L is <u>decidable</u> if there is a Turing machine M which accepts L and halts on every input string.

Every decidable language is Turing-Acceptable.

We can convert any Turing machine to have single accept and reject states.



A computational problem is decidable if the corresponding language is decidable.

Examples:

① Problem: is number x prime?

Decidable: divide x with all possible numbers between 2 and $\sqrt{x}$, if any of them divides x, then reject, else accept.

② Problem: Does DFA M accept the empty language $L(M) = \phi$?

Decidable: determine whether there is a path from the initial state to any accept state.

③ Problem: Does DFA M accept a finite language?

Decidable: check if there is a walk with a cycle from the initial state to an accepting state.

④ Problem: Does DFA M accept string w?

Decidable: Run DFA M on input string w.

⑤ Problem: Does DFAs M_1 and M_2 accept the same language?

Decidable: construct $L(M) = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$, determine if $L(M) = \phi$

Theorem: if a language L is decidable, then its complement $\bar{L}$ is decidable, too.

Proof: Build a turing machine M that accepts $\bar{L}$ and halts on every input string.

Tranform accept state to reject, and vice versa.

On each input string w do:

1. Let M be the decider for L.
2. Run M with input string w.
   if M accepts then reject.
   if M rejects then accept.

Therefore, it accepts $\bar{L}$ and halts on every input string.

End of Proof.

Undecidable Languages.

An undecidable language has no decider: any turing machine that accepts L does not halt on some input string.

There is a language which is Turing-Acceptable and undecidable.

We will prove that there is a language L, such that:

① L is Turing-Acceptable

② $\bar{L}$ is not Turing-Acceptable.

From ②, since we know the complement of a decidable language is decidable Therefore, L is undecidable.

Proof: consider alphabet $\{a\}$.

$L =$ strings of $\{a\}^+$, then L is countable since there is an enumerator that generates them.

Give binary representation to $L(M_i)$ as following:

	a^1	a^2	a^3	...
$L(M_1)$	0	1	0	...
$L(M_2)$	1	0	1	...
$L(M_3)$	1	1	0	...
⋮	...	...	...	

Let $L = \{a^i : a^i \in L(M_i)\}$. L consists of the 1s in the diagonal.

Consider $\bar{L} = \{a^i : a^i \notin L(M_i)\}$, then $\bar{L}$ consists of 0s in the diagonal.

Now we want to prove $\bar{L}$ is not turing acceptable.

Assume $\bar{L}$ is turing acceptable. Let M_k be the turing Machine that accepts $\bar{L}$: $L(M_k) = \bar{L}$

But $M_k \neq M_i$ for any i, since $\begin{cases} a^i \in L(M_k) \\ a^i \notin L(M_i) \end{cases}$ or $\begin{cases} a^i \notin L(M_k) \\ a^i \in L(M_i) \end{cases}$

Therefore, M_R cannot exist, $\bar{L}$ is not Turing Acceptable.

End of Proof.

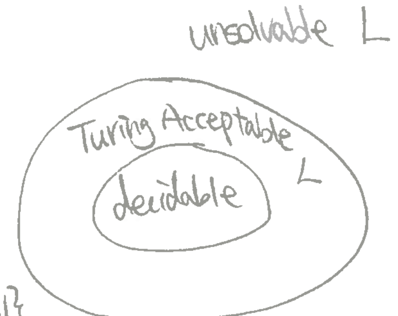
Undecidable Problems.

A language L is decidable if there is a Turing machine M that accepts L and halts on every input string.

Undecidable Language L: there is no decider for L, no turing machine which accepts L and halts on every input string.

Or there is no turing machine that gives an answer for every input instance.

Two unsolvable problems: ?



1. Membership problem: Does turing machine M accept string w? $w \in L(M)$?

Corresponding language $A_{TM} = \{ \langle M, w \rangle : M \text{ accept } w \}$.

Theorem: A_{TM} is undecidable. (The membership problem is unsolvable).

Proof: Suppose A_{TM} is decidable, then for input $\langle M, w \rangle$, there is a decider for A_{TM}.

Then given an arbitrary turing recognizable language, let M_L be the turing machine that accepts L, there is a decider for L.

Therefore, for any turing-acceptable language, it is decidable.

But we know there is a turing-Acceptable language which is undecidable.

We got contradiction here.

End of Proof.

A_{TM} is Turing-Acceptable.

$\langle M, w \rangle \longrightarrow$
1. Run M on input w
2. if M accepts w then accept $\langle M, w \rangle$.

2. Halting Problem: Does turing Machine M halt while processing input string w

Let $HALT_{TM} = \{ \langle M, w \rangle : M$ is a Turing Machine that halts on input string $w \}$.

Theorem: $HALT_{TM}$ is undecidable.

Proof: Suppose that $HALT_{TM}$ is decidable, then there is a decider for $HALT_{TM}$ with input $\langle M, w \rangle$.

Let L be an arbitary Turing-Acceptable language and M_L be the Turing Machine that accepts L.

Then we can build a decider for L. therefore, L is decidable.

Therefore, every Turing-Acceptable language is decidable.

But we got contradiction here since there is a Turing-Acceptable language which is undecidable.

End of Proof.

$HALT_{TM}$ is Turing-Acceptable.

$\langle M, w \rangle \longrightarrow$
1. Run M on input w.
2. if M halts on w then accept $\langle M, w$