

Gödel's Incompleteness

In any sufficiently powerful axiomatic system, there exist truths that cannot be proven.

$$\forall \exists \wedge \vee \neg$$

"This sentence is unprovable"

$$\forall x \exists y \dots$$

if \neg then cannot prove

if F and you prove this to be true then it is not false.

• Halting problem is undecidable.



Gödel's incompleteness

1. There are machines M_i that do not halt on inputs I_i \leftarrow this is true

~~Machines that halt on input~~

2. Machine M does not halt on input I \leftarrow sometimes true, sometimes false.

3. Suppose you could prove that machine M_k does not halt on input I_k
proof: finite string of symbols

4. \therefore Halting problem would be decidable: ^{search}
Run the TM and your proof algorithm at the same time.

1. if TM halts then HALT

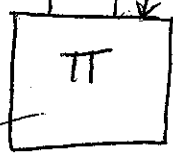
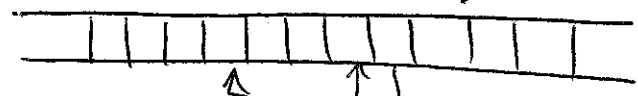
2. if finite proof determines no Halt, then No HALT.

Because the halting ~~is~~ problem is undecidable, there are machines that do not halt but you cannot prove this

things that are true

Random Access Machine

Array of registers, each capable of storing an arbitrarily large number



Finite state machine which has random read/write access to the registers

- READ R0
- STORE R1
- LOAD L1
- ADD R0, L1
- SUB R1, R2
- ⋮
- ⋮
- ⋮

Similar model to the computers of today!

equivalent to a TM

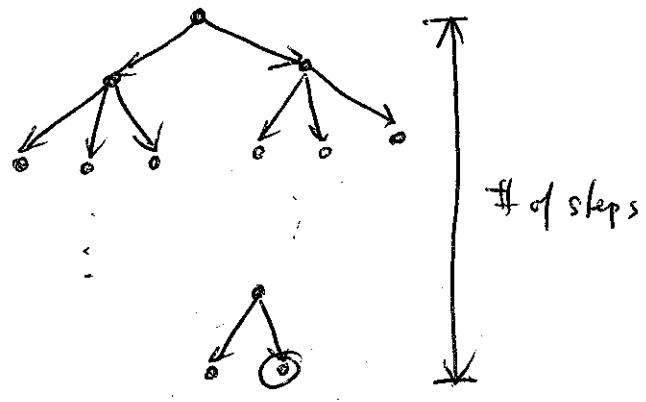
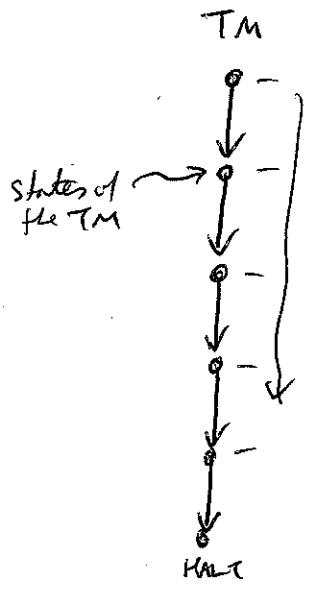
given enough time you can move back and forth and access everything!



Measuring time

of steps to get to the solution.

NDTM



$f(n)$ be a function of $n =$ size of the problem (n bits)

$TIME(f(n)) =$ set of problems we can solve on a TM (or general computer) in $O(f(n))$ time. polynomial

$SPACE(f(n)) =$ set of problems we can solve on a TM in $O(f(n))$ memory

$NTIME(f(n)) =$ 1) set of problems a non-deterministic TM can "solve" in $O(f(n))$ time

2) set of problems for which if the answer is "yes" there is a {proof, solution, witness, certificate} of this fact that we can check in $O(f(n))$ time

What a regular machine can do

$TIME(n)$ in $TIME(n)$, a TM can do in $TIME(n^2)$

if answer "yes", then \exists path for which the machine says yes
if answer "no", then \nexists path for which machine says yes.
 \forall paths say "no".
deterministic

DETERMINISTIC TM

$$P = \bigcup_k TIME(n^k)$$

$$PSPACE = \bigcup_k SPACE(n^k)$$

NON-DETERMINISTIC TM

$$NP = \bigcup_k NTIME(n^k)$$

$$NPSpace = \bigcup_k NSpace(n^k)$$

EX:

Traveling salesman problem (TSP)

Input: A list of cities with the distances between them.

Q: What is the shortest tour that visits each city once?

Is this in P? No! Is this in NP like I wrote it? **No!**

You cannot confirm that a given tour is the optimum? There might be a shorter one.

TSP decision version

Q: Is there a tour that visits each city once $< D$?

This is in NP. I can verify this in $O(n)$ time deterministically.

This algorithm can be used to find the shortest path

```

max_length = length(give_path) ← can be computed in O(n) time
for (i = 1; i < max_length; i++) {
  if (path = TSP-decision(i))
    return(path);
}
return(give_path)

```

Brute force TSP

n cities

$a_1, a_2, a_3, \dots, a_n$

Different orderings possible:

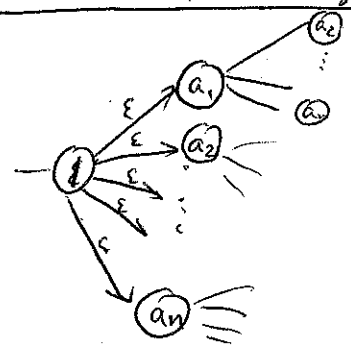
$$\underbrace{n}_{\text{---}} \underbrace{n-1}_{\text{---}} \underbrace{n-2}_{\text{---}} \dots \underbrace{1}_{\text{---}} = n!$$

So brute force algorithm is $O(n!)$

with dynamic programming $O(n^2 2^n)$

not a polynomial hence not in P!

Non-deterministic algorithm.



$a \vee b \Rightarrow$ disjunction
 $a \wedge b \Rightarrow$ conjunction

Conjunctive normal form $\Rightarrow (x_1 \vee x_2 \vee \dots) \wedge (a \vee b) \wedge (c \vee d)$

Ex! Satisfiability: Given a Boolean expression in conjunctive normal form, is it satisfiable? (SAT)

$$\phi = ((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3))$$

Brute force SAT:

```

for x3 in T, F
  for x2 in T, F
    for x1 in T, F {
      TEST EXPRESSION
    }
  }
}

```

$O(2^n)$ not polynomial time!

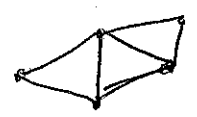
Suppose I have a verifier $V(x, w)$ which in TIME($f(n)$)

Question: $\left\{ \begin{array}{l} \text{input: } x \\ \text{question: does there exist a } w \text{ st. } V(x, w) \text{ returns "yes"?} \end{array} \right.$

if this takes non-poly time then this is NTIME($f(n)$)

Ex: Hamiltonian Path

Does there exist a path in a graph that visits each vertex exactly once?



$V(G, P)$

↑ ↑

graph path

given a path the verifier can check if on the graph.

$NTIME(f) \subseteq TIME(2^{O(f)})$

Why lump $\bigcup_K TIME(K^t)$ into P?

It allows us to abstract out the actual machine implementation.

is it a TM? is it a RAM?

It also gets rid of how we present the input.



The big question is $P = NP??$