Theorem

Let S be an infinite countable set. Then 2^S (powerset of S, i.e., $P(S)$) is uncountable.

$$P(S) = 2^S$$

We can use either of these notations to denote the powerset of S.

Proof: we will prove this using diagonalization.

Let $S = \{s_1, s_2, s_3, \ldots\}$

I need to show $P(S)$ is not countable.

Assume 2^S is countable.

We encode each element $t \in 2^S$ as an infinite sequence of 0's and 1's, with 1 at the position if and only if $s_i \in t$.

$t_1, t_2, t_3, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, \in 2^S$

$t_1 = 1\,0\,0\,1\,0\,1 \ldots ; \{s_1, s_4, s_6, \ldots$

$t_2 = 1\,0\,0\,0\,0\,0 \ldots ; \{s_1, \ldots$

$t_3 = 1\,0\,1\,1\,0\,0 \ldots ; \{s_1, s_3, s_4$

Since we assumed that 2^S is countable, I can map the set of natural numbers to 2^S using a bijection

i	$f(i)$
1	$\underline{1}\,0\,0\,1\,1\,0\,0 - - - -$
2	$0\,\underline{1}\,0\,1\,1\,0\,1 - - - -$
3	$1\,1\,\underline{0}\,0\,0\,1\,0 - - - -$

I will create element t_k where each entry of t_k is constructed by taking the complement of the diagonal in this table.

$t_k = 0\,0\,1 - - - - -$

$\therefore$ t_k does not have a mapping in this bijection

$\therefore$ 2^S is not countable.

We can use this idea to show that there are some languages not Turing-Recognizable

Theorem: Some languages are not Turing-Recognizable.

Proof Idea:

Set of all possible turing machines are countable.

- TM M can be encoded into a string $\langle M \rangle$.
- This string can be further encoded into a binary string.
- The set of all binary strings is countable
  - $\Sigma = \{0, 1\}$
  - Σ^* is countable
    - Simply list the strings in Σ^* in lexicographical order
    - $\{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$

- you can assign a unique natural number to each string in this set.

- Observe that each TM recognizes a language $L \subseteq \Sigma^*$
- A language is a set of strings.
- $L \in P(\Sigma^*)$
- Note that Σ^* is contably infinite.
- We showed that $P(\Sigma^*)$ is not countable.
- Therefore, we do not have enough turing machines to recognize all the possible languages.

Undecidability

We will consider several languages.

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

$E_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

we will look at A_{TM}

let us create an algorithm for A_{TM}

1. Simulate M on w
2. If M stops at the accept state, accept;
   If M stops at a reject state, reject

//otherwise loop, looping is hard to distinguish from taking a long time to run.

This is a recognizer for A_{TM} but not a decider

- A_{TM} is not the problem of asking you to write a program that accepts w.

- Suppose you have a program that someone else wrote, which you do not know how it works. Now you need to check whether this program works as intended. For example you need to check whether specific input is accepted by the program. what you can do feed the program and the input to A_{TM}, and it will say whether that specific input is accepted by the program or not

Theorem 4.11 A_{TM} is undecidable.

Proof: We prove this theorem by contradiction.

Assume A_{TM} is decidable.

Let TM H is a decider for the A_{TM} language.

$$H(\langle M, w \rangle) = \begin{cases} \text{accept}, & \text{If } M \text{ accepts } w \\ \text{reject}, & \text{If } M \text{ does not accept } w. \end{cases}$$

Then I am going to construct TM D, where it will take $\langle K \rangle$; K is a TM, as the input.

D = "on $\langle K \rangle$, where K is a TM

1. Run H on $\langle K, \langle K \rangle \rangle$
2. output the opposite of what H outputs on $\langle K, \langle K \rangle \rangle$

Now, I am going to feed $\langle D \rangle$ to D.

$$D(\langle D \rangle) = \begin{cases} \text{accept, if D does not accept } \langle D \rangle \\ \text{reject, if D accept } \langle D \rangle \end{cases}$$

This is a contradiction.

$\therefore A_{TM}$ is not decidable.

Let's look into this proof bit more carefully.

- This proof implicitly involves diagonalization.
  - H accepts $\langle M, w \rangle$ when M accepts w.
  - D rejects $\langle K \rangle$ when K accepts $\langle K \rangle$
  - D rejects $\langle D \rangle$ when D accepts $\langle D \rangle$

Let us create a table that describes behaviorrs of TM when we input TM as inputs to them.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$ ----------
M_1	accept		accept	
M_2		accept		accept
M_3				
M_4	accept	accept		
⋮				

each entry is accept if the machine accepts the input and blank if it rejects or loop on that input.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$ ----------
M_1	accept		accept	
M_2		accept		accept
M_3				
M_4	accept	accept		
⋮				

Note that entry i,j is the value of H on the input $\langle M_i, \langle M_j \rangle \rangle$

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	.....	D	.. ..
M_1	accept		accept			⋮	
M_2		accept		accept		⋮	
M_3						⋮	
M_4	accept	accept				⋮	
⋮						⋮	
D	reject	reject	accept		.....	?	

I cannot put a value here.

This proof looks very hard.

Seems like it's very hard to prove something is undecidable using these type of argument.

* There is a easier way.

Reduction

$A \propto B$ $(A \leq B)$

A, B are problems

A reduction is a way of converting one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem.

If $A \propto B$, then we can use an algorithm that solves problem B to solve problem A.

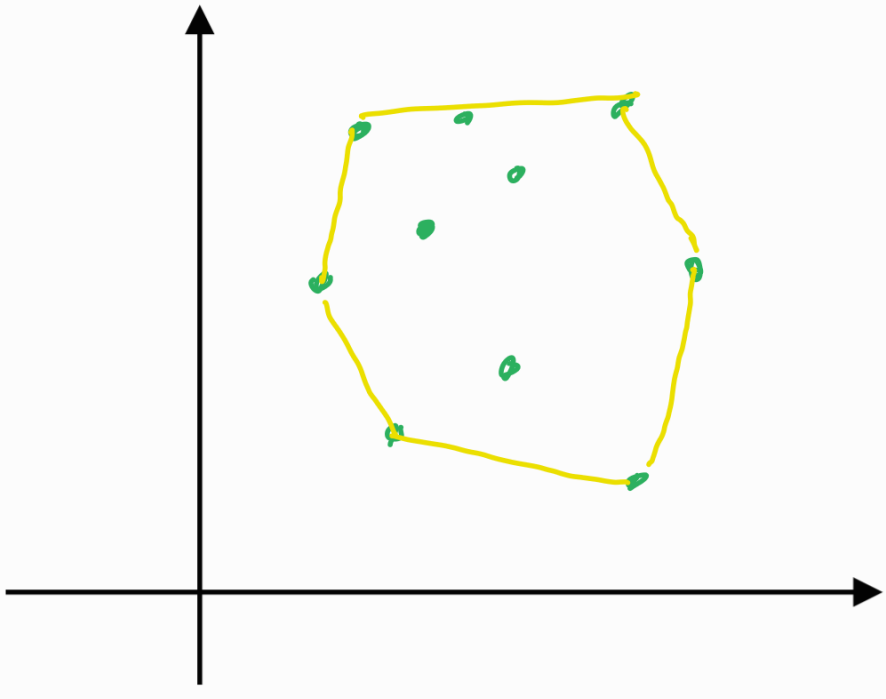
* If you already know problem A is very hard to solve, and If A is reducible to B, then you can use a hypothetical algo that solves B to solve problem A.

Then Problem B must be at least as hard as problem A.

Example of a reduction.

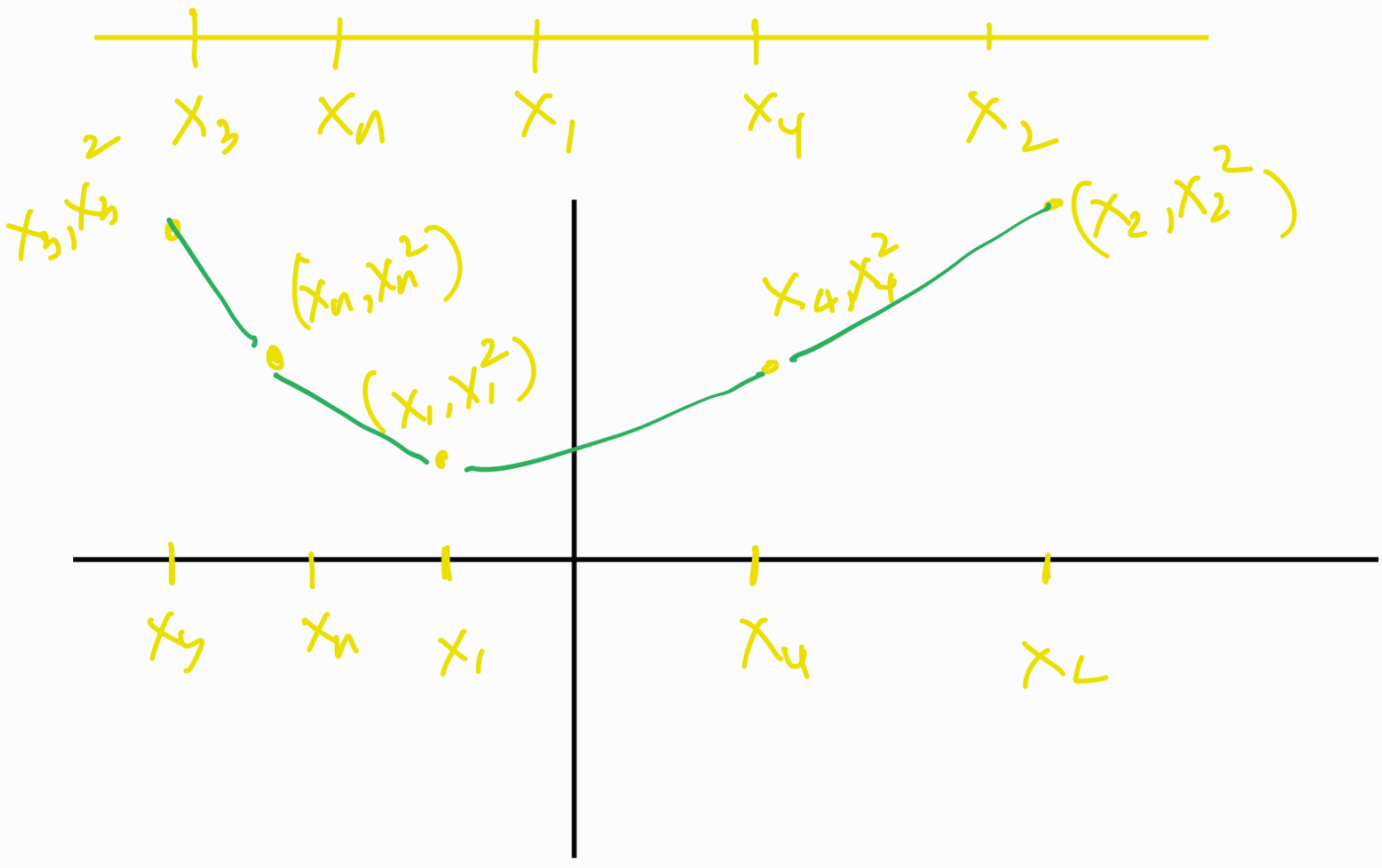
Sorting n real numbers $x_1, x_2, x_3, \ldots, x_n$ $\leq$ 2D convex hull problem.

Given set of n points in 2D plane, find the convex hull.



Given $x_1, x_2, x_3, \ldots x_n$, convert these points to set of 2D points, where

$x_i \rightarrow (x_i, x_i^2)$



$x_1, x_2, \ldots x_n$ → (x_i, x_i^2)

Then we compute the CH of new points

Start from the left most vertex in the CH and output points → $x_3, x_n, x_1, \ldots$

Sorting is $\Omega(n \log n)$, therefore CH algorithm must have $\Omega(n \log n)$

Reducibility

- Reduction: convert problem A to B s.t the solution to B can be used to solve A.

$(A \leq B, A \propto B)$

How can we use this to show the undecidability?

Right now we proved A_{TM} problem is undecidable.

We are going to prove $HALT_{TM}$ is undecidable.

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on } w \}$$

$HALT_{TM}$ and A_{TM} are two different languages.

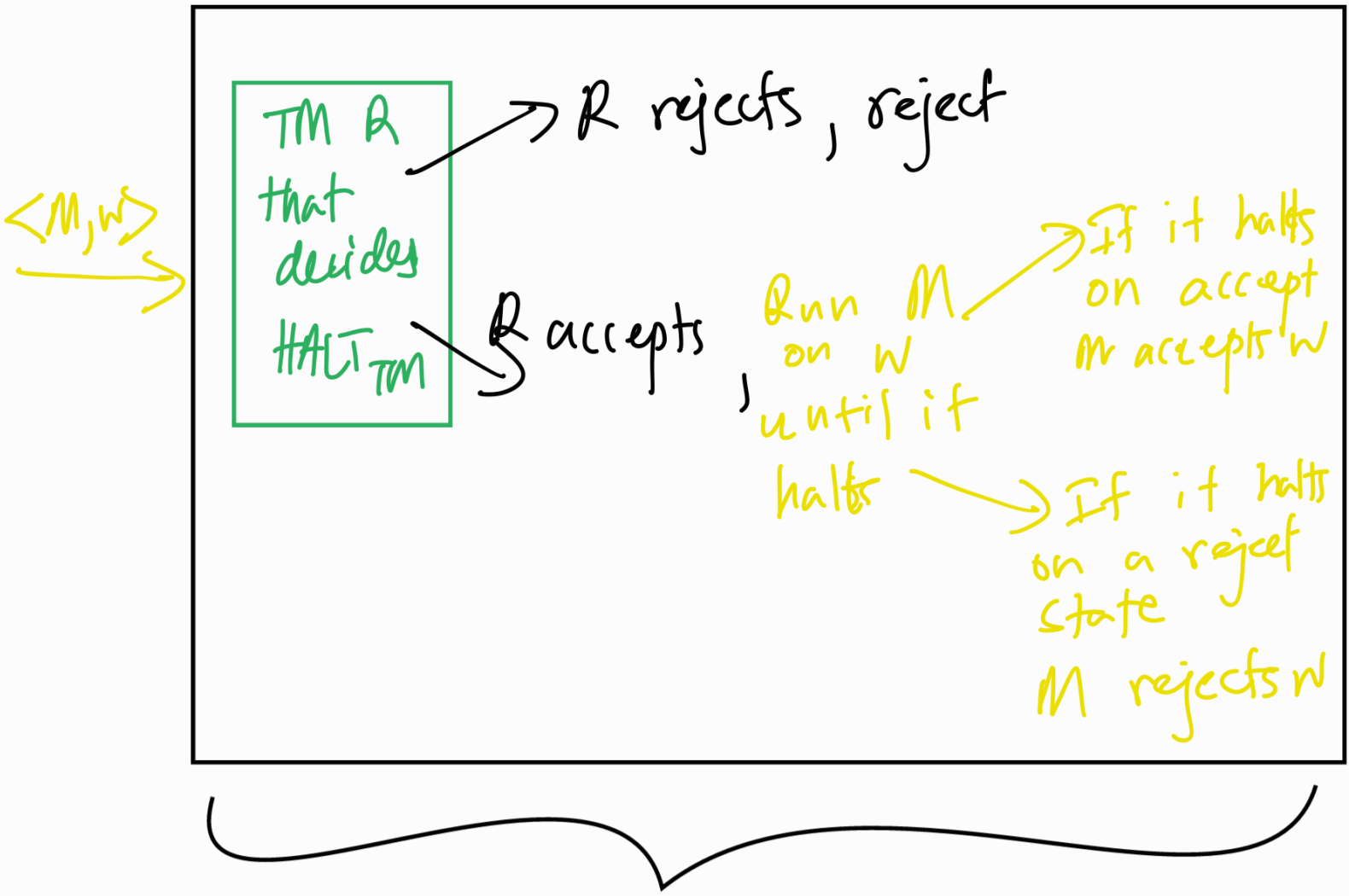
We are going to prove $HALT_{TM}$ is undecidable.

We will show that

$$A_{TM} \leq HALT_{TM}$$

(Note that order of the reduction matter)

Assume $HALT_{TM}$ is decidable, let TM R be the decider for $HALT_{TM}$.



A decider for A_{TM}

Clearly this is an algorithm that decides A_{TM}. But this cannot happen as A_{TM} is undecidable.

∴ Our initial assumption is incorrect.

$HALT_{TM}$ is undecidable.