A deterministic finite automaton (DFA) is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

- $Q$ is the set of states (finite)
- $\Sigma$ is the alphabet (finite)
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

Let $w_1, \ldots, w_n \in \Sigma$ and $w = w_1 \ldots w_n \in \Sigma^*$
Then $M$ accepts $w$ if there are $r_0, r_1, \ldots, r_n \in Q$, s.t.

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$
Let $w \in \Sigma^*$ and suppose $w$ can be written as $w_1 \ldots w_n$ where $w_i \in \Sigma_\varepsilon$ ($\varepsilon =$ empty string).

Then $N$ accepts $w$ if there are $r_0, r_1, \ldots, r_n \in Q$ such that

1. $r_0 \in Q_0$
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for $i = 0, \ldots, n-1$, and
3. $r_n \in F$

$L(N) =$ the language recognized by $N$

$= \text{set of all strings machine } N \text{ accepts}$

A language $L$ is recognized by an NFA $N$ if $L = L(N)$. 
Let \( w \in \Sigma^* \) and suppose \( w \) can be written as \( w_1 \ldots w_n \) where \( w_i \in \Sigma_\varepsilon \) (recall \( \Sigma_\varepsilon = \Sigma \cup \{\varepsilon\} \)).

Then \( P \) accepts \( w \) if there are 
\[ r_0, r_1, \ldots, r_n \in Q \] and 
\[ s_0, s_1, \ldots, s_n \in \Gamma^* \] (sequence of stacks) such that 

1. \( r_0 = q_0 \) and \( s_0 = \varepsilon \) (\( P \) starts in \( q_0 \) with empty stack)

2. For \( i = 0, \ldots, n-1: \)
\[ (r_{i+1}, b) \in \delta(r_i, w_{i+1}, a), \] where \( s_i = a t \) and \( s_{i+1} = b t \) for some \( a, b \in \Gamma_\varepsilon \) and \( t \in \Gamma^* \)
(\( P \) moves correctly according to state, stack and symbol read)

3. \( r_n \in F \) (\( P \) is in an accept state at the end of its input)
THEOREM

For every regular language $L$, there exists a **UNIQUE** (up to re-labeling of the states) minimal DFA $M$ such that $L = L(M)$.
EXTENDING $\delta$

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$, extend $\delta$

to $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ as follows:

$$\hat{\delta}(q, \varepsilon) = q$$
$$\hat{\delta}(q, \sigma) = \delta(q, \sigma)$$
$$\hat{\delta}(q, w_1 \ldots w_{k+1}) = \delta(\delta(q, w_1 \ldots w_k), w_{k+1})$$

Note: $\delta(q_0, w) \in F \iff M$ accepts $w$

String $w \in \Sigma^*$ **distinguishes** states $q_1$ and $q_2$ iff exactly ONE of $\hat{\delta}(q_1, w), \hat{\delta}(q_2, w)$ is a final state
Fix $M = (Q, \Sigma, \delta, q_0, F)$ and let $p, q, r \in Q$

**Definition:**

$p \sim q$ iff $p$ is *indistinguishable* from $q$

$p \nslash \sim q$ iff $p$ is *distinguishable* from $q$

**Proposition:** $\sim$ is an *equivalence relation*

$p \sim p$ (reflexive)

$p \sim q \Rightarrow q \sim p$ (symmetric)

$p \sim q$ and $q \sim r \Rightarrow p \sim r$ (transitive)
Proposition: ~ is an *equivalence relation*

so ~ partitions the set of states of M into disjoint equivalence classes

\[ [q] = \{ p \mid p \sim q \} \]
TABLE-FILLING ALGORITHM

Input: DFA $M = (Q, \Sigma, \delta, q_0, F)$

Output: (1) $D_M = \{ (p,q) \mid p,q \in Q \text{ and } p \not\equiv q \}$

(2) $E_M = \{ [q] \mid q \in Q \}$

Base Case: $p$ accepts and $q$ rejects $\Rightarrow p \not\equiv q$

Recursion: if there is $\sigma \in \Sigma$ and states $p', q'$ satisfying

$\delta (p, \sigma) = p'$
$\not\equiv \Rightarrow p \not\equiv q$

$\delta (q, \sigma) = q'$

Repeat until no more new $D$’s
CONVERTING NFAs TO DFAs

Input: NFA $N = (Q, \Sigma, \delta, Q_0, F)$

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$

$Q' = 2^Q$

$\delta' : Q' \times \Sigma \rightarrow Q'$

$\delta'(R, \sigma) = \bigcup_{r \in R} \varepsilon(\delta(r, \sigma))$

$q_0' = \varepsilon(Q_0)$

$F' = \{ R \in Q' | f \in R \text{ for some } f \in F \}$

For $R \subseteq Q$, the $\varepsilon$-closure of $R$, $\varepsilon(R) = \{ q \text{ that can be reached from some } r \in R \text{ by traveling along zero or more } \varepsilon \text{ arrows} \}$
THE REGULAR OPERATIONS

Union: \( A \cup B = \{ w \mid w \in A \text{ or } w \in B \} \)

Intersection: \( A \cap B = \{ w \mid w \in A \text{ and } w \in B \} \)

Negation: \( \neg A = \{ w \in \Sigma^* \mid w \notin A \} \)

Reverse: \( A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \} \)

Concatenation: \( A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \} \)

Star: \( A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \} \)
REGULAR EXPRESSIONS

σ is a regexp representing \{\sigma\}

ε is a regexp representing \{\varepsilon\}

∅ is a regexp representing ∅

If R₁ and R₂ are regular expressions representing L₁ and L₂ then:

(R₁R₂) represents L₁ \cdot L₂

(R₁ \cup R₂) represents L₁ \cup L₂

(R₁)^* represents L₁^*
EQUIVALENCE

L can be represented by a regexp

⇔

L is a regular language
\[ q_0 \xrightarrow{\epsilon} q_1 \xrightarrow{b} q_2 \xrightarrow{\epsilon} q_3 \]

\[ R(q_0, q_3) = (a*b)(a∪b)^* \]

\[ (a*b)(a∪b)^* \]
How can we test if two regular expressions are the same?

Length $n$

$R_1$

$N_1$

$N_2$

$R_2$

$O(n)$ states

$N_1$

$N_2$

$O(2^n)$ states

$M_1$

$M_2$

$M_{1\,\text{MIN}}$

$=?=$

$M_{2\,\text{MIN}}$
A context-free grammar (CFG) is a tuple \( G = (V, \Sigma, R, S) \), where:

- \( V \) is a finite set of variables
- \( \Sigma \) is a finite set of terminals (disjoint from \( V \))
- \( R \) is set of production rules of the form \( A \rightarrow W \), where \( A \in V \) and \( W \in (V \cup \Sigma)^* \)
- \( S \in V \) is the start variable

\[ L(G) = \{ w \in \Sigma^* \mid S \Rightarrow^* w \} \] Strings Generated by \( G \)

A Language \( L \) is context-free if there is a CFG that generates precisely the strings in \( L \)
A context-free grammar is in **Chomsky normal form** if every rule is of the form:

- \( A \rightarrow BC \)  \( B \) and \( C \) aren’t start variables
- \( A \rightarrow a \)  \( a \) is a terminal
- \( S \rightarrow \varepsilon \)  \( S \) is the start variable

Any variable \( A \) that is not the start variable can only generate strings of length \( > 0 \)
**Theorem:** Any context-free language can be generated by a context-free grammar in Chomsky normal form.

**Theorem:** If $G$ is in CNF, $w \in L(G)$ and $|w| > 0$, then any derivation of $w$ in $G$ has length $2|w| - 1$.

**Theorem:** There is an $O(n^3 + \text{size } G)$ membership algorithm (CYK) any Chomsky normal form form $G$. 
Theorem: The set of PDAS that accept all strings is not r.e.
**Definition:** A (non-deterministic) PDA is a tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet
- $\Gamma$ is the stack alphabet
- $\delta : Q \times \Sigma \epsilon \times \Gamma \epsilon \rightarrow 2^{Q \times \Gamma \epsilon}$
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

$2^Q$ is the set of subsets of $Q$ and $\Sigma \epsilon = \Sigma \cup \{\epsilon\}$
A Language $L$ is generated by a CFG

$\iff$

$L$ is recognized by a PDA
THE PUMPING LEMMA
(for Context Free Grammars)
Let \( L \) be a context-free language with \( |L| = \infty \)

Then **there is an integer** \( P \) **such that**

if \( w \in L \) and \( |w| \geq P \)

then can write \( w = uv^ixyz \), where:

1. \( |vy| > 0 \)
2. \( |vxy| \leq P \)
3. \( uv^ixy^iz \in L \), for any \( i \geq 0 \)
TURING MACHINE

FINITE STATE CONTROL

INFINITE TAPE

$q_1$

INPUT

INFINITE TAPE
Definition: A Turing Machine is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \not\in \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$
CONFIGURATIONS

11010q₇001110

corresponds to:
A Turing Machine $M$ accepts input $w$ if there is a sequence of configurations $C_1, \ldots, C_k$ such that

1. $C_1$ is a *start* configuration of $M$ on input $w$, ie $C_1$ is $q_0w$

2. each $C_i$ *yields* $C_{i+1}$, ie $M$ can legally go from $C_i$ to $C_{i+1}$ in a single step

$ua \, q_i \, bv \, yields \, u \, q_j \, acv$ if $\delta(q_i, b) = (q_j, c, L)$

$ua \, q_i \, bv \, yields \, uac \, q_j \, v$ if $\delta(q_i, b) = (q_j, c, R)$
A Turing Machine $M$ accepts input $w$ if there is a sequence of configurations $C_1, \ldots, C_k$ such that

1. $C_1$ is a start configuration of $M$ on input $w$, ie $C_1$ is $q_0w$

2. each $C_i$ yields $C_{i+1}$, ie $M$ can legally go from $C_i$ to $C_{i+1}$ in a single step

3. $C_k$ is an accepting configuration, ie the state of the configuration is $q_{\text{accept}}$
A Turing Machine $M$ **rejects** input $w$ if there is a sequence of configurations $C_1, \ldots, C_k$ such that

1. $C_1$ is a *start* configuration of $M$ on input $w$, ie $C_1$ is $q_0w$

2. each $C_i$ *yields* $C_{i+1}$, ie $M$ can legally go from $C_i$ to $C_{i+1}$ in a single step

3. $C_k$ is a *rejecting* configuration, ie the state of the configuration is $q_{\text{reject}}$
A TM **decides** a language if it accepts all strings in the language and rejects all strings not in the language.

A language is called **decidable** or **recursive** if some TM decides it.

**Theorem:** $L$ decidable $\iff \neg L$ decidable

**Proof:** $L$ has a machine $M$ that accepts or rejects on all inputs. Define $M'$ to be $M$ with accept and reject states swapped. $M'$ decides $\neg L$. 
A TM **recognizes** a language if it accepts all and only those strings in the language.

A language is called **Turing-recognizable** or **recursively enumerable**, (or **r.e.** or **semi-decidable**) if some TM recognizes it.

A TM **decides** a language if it accepts all strings in the language and rejects all strings not in the language.

A language is called **decidable** or **recursive** if some TM decides it.
A TM recognizes a language if it accepts all and only those strings in the language.

A language is called Turing-recognizable or recursively enumerable, (or r.e. or semi-decidable) if some TM recognizes it.

FALSE: \( L \) r.e. <-> \( \neg L \) r.e.

Proof: \( L \) has a machine \( M \) that accepts or rejects on all inputs. Define \( M' \) to be \( M \) with accept and reject states swapped. \( M' \) decides \( \neg L \).
A language is called Turing-recognizable or recursively enumerable (r.e.) or semi-decidable if some TM recognizes it.

A language is called decidable or recursive if some TM decides it.

Theorem: If $A$ and $\neg A$ are r.e. then $A$ is recursive.
Theorem: If $A$ and $\neg A$ are r.e. then $A$ is recursive.

Suppose $M$ accepts $A$. $M'$ accepts $\neg A$ decidable.
Use Odd squares/ Even squares simulation of $M$ and $M'$. If $x$ is accepted by the even squares reject it/ accepted by the odd squares then accept $x$. 
TURING MACHINE with WRITE ONLY output tape.

Outputs a sequence of strings separated by hash marks. Allows for a well defined infinite sequence of strings in the limit. The machine is said to enumerate the sequence of strings occurring on the tape.
Outputs a sequence of strings separated by hash marks. Allows for a well-defined infinite sequence of strings in the limit. The machine is said to enumerate the set of strings occurring on the tape.
From every TM $M$ accepting $A$, there is a TM $M'$ outputting $A$.

For $n = 0$ to forever do
{
    {Do $n$ parallel simulations of $M$ for $n$ steps for the first $n$ inputs}
    $M(0), M(1), M(2), M(3)$...
}
From every TM $M$ outputting $A$, there is a TM $M'$ accepting $A$.

$M''(X)$ run $M$, accept if $X$ output on tape.
Let $\mathbb{Z}^+ = \{1,2,3,4…\}$. There exists a bijection between $\mathbb{Z}^+$ and $\mathbb{Z}^+ \times \mathbb{Z}^+$ (or $\mathbb{Q}^+$)
Lex-order has an enumerator for strings of length 1, the length 2, ....

Pairs of binary strings have a lex-order enumerator for each $n > 0$ list all pairs of strings $a, b$ as $\#a\#b\#$ where total length of $a$ and $b$ is $n$.

Let $\text{BINARY}(w) = \text{pair of binary strings be any fixed way of encoding a pair of binary strings with a single binary string}$
THE ACCEPTANCE PROBLEM

\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

Theorem: \( A_{TM} \) is semi-decidable (r.e.)

but NOT decidable

\( A_{TM} \) is r.e.:

Define a TM \( U \) as follows:

On input \( (M, w) \), \( U \) runs \( M \) on \( w \). If \( M \) ever accepts, accept. If \( M \) ever rejects, reject.

NB. When we write “input \( (M, w) \)” we really mean “input code for (code for \( M, w \))”
MULTITAPE TURING MACHINES

\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k \]
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
We can encode a TM as a string of 0s and 1s

\[0^n10^m10^k10^s10^t10^r10^u1\ldots\]

- \(n\) states
- \(m\) tape symbols (first \(k\) are input symbols)
- start state
- accept state
- reject state
- blank symbol

\[( (p, a), (q, b, L) ) = 0^p10^a10^q10^b10\]
\[( (p, a), (q, b, R) ) = 0^p10^a10^q10^b11\]
UNDECEIDABLE PROBLEMS

THURSDAY Feb 13
There are languages over \{0,1\} that are not decidable
Languages over \{0,1\}

Turing Machines
Let \( L \) be any set and \( 2^L \) be the power set of \( L \).

**Theorem:** There is no onto map from \( L \) to \( 2^L \).

**Proof:** Assume, for a contradiction, that there is an onto map \( f : L \to 2^L \).

Let \( S = \{ x \in L \mid x \notin f(x) \} \).

If \( S = f(y) \) then \( y \in S \) if and only if \( y \notin S \).

Can give a more constructive argument!
**Theorem:** There is no onto function from the positive integers to the real numbers in $(0, 1)$.

**Proof:** Suppose $f$ is any function mapping the positive integers to the real numbers in $(0, 1)$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$ corresponding $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0.28347279...$</td>
</tr>
<tr>
<td>2</td>
<td>$0.88388384...$</td>
</tr>
<tr>
<td>3</td>
<td>$0.77635284...$</td>
</tr>
<tr>
<td>4</td>
<td>$0.11111111...$</td>
</tr>
<tr>
<td>5</td>
<td>$0.12345678...$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Let $r = 11121...$.

The $n$-th digit of $r$ is defined as follows:

$$\lfloor \text{n-th digit of } r \rfloor = \begin{cases} 
1 & \text{if } \lfloor \text{n-th digit of } f(n) \rfloor \neq 1 \\
2 & \text{otherwise}
\end{cases}$$

Then $f(n) \neq r$ for all $n$ (Here, $r = 11121...$).
THE MORAL:
No matter what L is, $2^L$ always has more elements than L.
Not all languages over \( \{0,1\} \) are decidable, in fact: not all languages over \( \{0,1\} \) are semi-decidable.

\{decidable languages over \( \{0,1\} \}\}

\{semi-decidable languages over \( \{0,1\} \}\}

\{Turing Machines\}

\{Strings of 0s and 1s\}

Set \( L \)

Set of all subsets of \( L \): \( 2^L \)
THE ACCEPTANCE PROBLEM

\[ A_{TM} = \{ (M, w) | M \text{ is a TM that accepts string } w \} \]

Theorem: \( A_{TM} \) is semi-decidable (r.e.)

but **NOT** decidable

\( A_{TM} \) is r.e. :

Define a TM \( U \) as follows:

On input \((M, w)\), \( U \) runs \( M \) on \( w \). If \( M \) ever accepts, accept. If \( M \) ever rejects, reject.

**NB.** When we write “input \((M, w)\)” we really mean “input code for (code for \( M, w \))”
THE ACCEPTANCE PROBLEM

$A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \}$

**Theorem:** $A_{TM}$ is semi-decidable (r.e.)

but **NOT** decidable

$A_{TM}$ is r.e. :

Define a TM $U$ as follows: $U$ is a *universal TM*

On input $(M, w)$, $U$ runs $M$ on $w$. If $M$ ever accepts, accept. If $M$ ever rejects, reject.

Therefore,

$U$ accepts $(M, w) \iff M$ accepts $w \iff (M, w) \in A_{TM}$

Therefore, $U$ recognizes $A_{TM}$
\( A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \)

**\( A_{TM} \) is undecidable:** (proof by contradiction)

Assume machine \( H \) decides \( A_{TM} \)

\[
H( (M, w) ) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Reject} & \text{if } M \text{ does not accept } w 
\end{cases}
\]

Construct a new TM \( D \) as follows: on input \( M \), run \( H \) on \((M, M)\) and output the opposite of \( H \)

\[
D( D ) = \begin{cases} 
\text{Reject} & \text{if } D \text{ accepts } D \\
\text{Accept} & \text{if } D \text{ does not accept } D 
\end{cases}
\]
<table>
<thead>
<tr>
<th></th>
<th>M_1</th>
<th>M_2</th>
<th>M_3</th>
<th>M_4</th>
<th>...</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_1</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>M_2</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td></td>
<td>reject</td>
</tr>
<tr>
<td>M_3</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>M_4</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Theorem:** $A_{TM}$ is r.e. but NOT decidable

**Cor:** $\neg A_{TM}$ is not even r.e.!
$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

$A_{TM}$ is undecidable: A constructive proof:

Let machine $H$ semi-decides $A_{TM}$ (Such $\exists$, why?)

$$H( (M,w) ) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Reject or No output} & \text{if } M \text{ does not accept } w
\end{cases}$$

Construct a new TM $D$ as follows: on input $M$, run $H$ on $(M,M)$ and output

$$D( D ) = \begin{cases} 
\text{Reject} & \text{if } H ( D, D ) \text{ Accepts} \\
\text{Accept} & \text{if } H ( D, D ) \text{ Rejects} \\
\text{No output} & \text{if } H ( D, D ) \text{ has No output}
\end{cases}$$

$$H( (D,D)) = \text{No output}$$

No Contradictions!
We have shown:

Given any machine $H$ for semi-deciding $A_{TM}$, we can effectively construct a TM $D$ such that $(D,D) \notin A_{TM}$ but $H$ fails to tell us that.

That is, $H$ fails to be a decider on instance $(D,D)$.

In other words,

Given any “good” candidate for deciding the **Acceptance Problem**, we can effectively construct an instance where the candidate fails.
THE classical HALTING PROBLEM

\[ \text{HALT}_{TM} = \{ (M, w) \mid M \text{ is a TM that halts on string } w \} \]

Theorem: \( \text{HALT}_{TM} \) is undecidable

Proof: Assume, for a contradiction, that TM \( H \) decides \( \text{HALT}_{TM} \)

We use \( H \) to construct a TM \( D \) that decides \( A_{TM} \)

On input \( (M, w) \), \( D \) runs \( H \) on \( (M, w) \):

- If \( H \) rejects then reject
- If \( H \) accepts, run \( M \) on \( w \) until it halts:
  - Accept if \( M \) accepts, ie halts in an accept state
  - Otherwise reject
MAPPING REDUCIBILITY

$f : \Sigma^* \rightarrow \Sigma^*$ is a computable function if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.

A language $A$ is mapping reducible to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B$$

$f$ is called a reduction from $A$ to $B$.

Think of $f$ as a “computable coding”
A is mapping reducible to B, \( A \leq_m B \), if there is a computable \( f : \Sigma^* \to \Sigma^* \) such that \( w \in A \iff f(w) \in B \).

Also, \( \neg A \leq_m \neg B \), why?
Theorem: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable

Proof: Let $M$ decide $B$ and let $f$ be a reduction from $A$ to $B$

We build a machine $N$ that decides $A$ as follows:

On input $w$:

1. Compute $f(w)$
2. Run $M$ on $f(w)$
**Theorem:** If $A \leq_m B$ and $B$ is (semi) decidable, then $A$ is (semi) decidable

**Proof:** Let $M$ (semi) decide $B$ and let $f$ be a reduction from $A$ to $B$

We build a machine $N$ that (semi) decides $A$ as follows:

On input $w$:

1. Compute $f(w)$
2. Run $M$ on $f(w)$
RICE’S THEOREM

Let $L$ be a language over Turing machines. Assume that $L$ satisfies the following properties:

1. For TMs $M_1$ and $M_2$, if $M_1 \equiv M_2$ then
   
   $M_1 \in L \iff M_2 \in L$

2. There are TMs $M_1$ and $M_2$, such that $M_1 \in L$ and $M_2 \not\in L$

Then $L$ is undecidable
THE PCP GAME

ba
---
a

a
---
ab

b
---
bcb

b
---
a
THE ARITHMETIC HIERARCHY
Is \((M, w)\) in \(A_{\text{TM}}\)?

**ORACLE TMs**

Oracle for \(A_{\text{TM}}\)

INPUT

INFINITE TAPE
ORACLE MACHINES

An ORACLE is a set $B$ to which the TM may pose membership questions “Is $w$ in $B$?”
(formally: TM enters state $q_?$) and the TM always receives a correct answer in one step
(formally: if the string on the “oracle tape” is in $B$, state $q_?$ is changed to $q_{YES}$, otherwise $q_{NO}$)

This makes sense even if $B$ is not decidable! (We do not assume that the oracle $B$ is a computable set!)
We say $A$ is semi-decidable in $B$ if there is an oracle TM $M$ with oracle $B$ that semi-decides $A$

We say $A$ is decidable in $B$ if there is an oracle TM $M$ with oracle $B$ that decides $A$
Language A "Turing Reduces" to Language B

if A is decidable in B, ie if there is an oracle TM M with oracle B that decides A

$A \leq_T B$
$\leq_T$ VERSUS $\leq_m$

**Theorem:** If $A \leq_m B$ then $A \leq_T B$

**Proof:**

If $A \leq_m B$ then there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B$$

We can thus use an oracle for $B$ to decide $A$

**Theorem:** $\neg_AT_{TM} \leq_T AT_{TM}$

**Theorem:** $\neg_AT_{TM} \leq_m AT_{TM}$
THE ARITHMETIC HIERARCHY

\[ \Delta^0_{n+1} = \{ \text{decidable sets} \} \quad \text{(sets} = \text{languages)} \]

\[ \Sigma^0_{n+1} = \{ \text{semi-decidable sets} \} \]

\[ \Sigma^0_n = \{ \text{sets semi-decidable in some } B \in \Sigma^0_n \} \]

\[ \Delta^0_{n+1} = \{ \text{sets decidable in some } B \in \Sigma^0_n \} \]

\[ \Pi^0_n = \{ \text{complements of sets in } \Sigma^0_n \} \]
Decidable Languages

Semi-decidable Languages

Co-semi-decidable Languages

∪ = \sum_0^1 \cap \Pi_0^1

Decidable Languages
Semi-decidable Languages

Decidable Languages

Co-semi-decidable Languages

\[ \sum_0^0 \cap \Pi_0^0 = \sum_1^1 \cap \Pi_1^1 \]
Theorem

\[ \sum_1^0 = \{ \text{semi-decidable sets} \} \]
\[ = \text{languages of the form } \{ x \mid \exists y \ R(x,y) \} \]

\[ \Pi_1^0 = \{ \text{complements of semi-decidable sets} \} \]
\[ = \text{languages of the form } \{ x \mid \forall y \ R(x,y) \} \]

\[ \Delta_1^0 = \{ \text{decidable sets} \} \]
\[ = \sum_1^0 \cap \Pi_1^0 \]

Where R is a decidable predicate
Theorem

$$\sum_2^0 = \{ \text{sets semi-decidable in some semi-dec. B } \}$$
= languages of the form \{ x | \exists y_1 \forall y_2 \ R(x,y_1,y_2) \}

$$\Pi_2^0 = \{ \text{complements of } \sum_2^0 \text{ sets} \}$$
= languages of the form \{ x | \forall y_1 \exists y_2 \ R(x,y_1,y_2) \}

$$\Delta_2^0 = \sum_2^0 \cap \Pi_2^0$$

Where R is a decidable predicate
Theorem

\[ \sum^0_n = \text{languages } \{ x \mid \exists y_1 \forall y_2 \exists y_3 \ldots Q y_n R(x, y_1, \ldots, y_n) \} \]

\[ \Pi^0_n = \text{languages } \{ x \mid \forall y_1 \exists y_2 \forall y_3 \ldots Q y_n R(x, y_1, \ldots, y_n) \} \]

\[ \Delta^0_n = \sum^0_n \cap \Pi^0_n \]

Where \( R \) is a decidable predicate
\[ \Sigma^0_1 = \text{languages of the form } \{ x \mid \exists y \ R(x,y) \} \]

We know that \( A_{\text{TM}} \) is in \( \Sigma^0_1 \) Why?

Show it can be described in this form:

\[ A_{\text{TM}} = \{ <(M,w)> \mid \exists t \ [M \text{ accepts w in t steps}] \} \]

Example

\[ A_{\text{TM}} = \{ <(M,w)> \mid \exists v \ (v \text{ is an accepting computation history of } M \text{ on } w) \} \]
Definition: A decidable predicate $R(x,y)$ is some proposition about $x$ and $y^1$, where there is a TM $M$ such that

for all $x, y$, $R(x,y)$ is TRUE $\implies$ $M(x,y)$ accepts

$R(x,y)$ is FALSE $\implies$ $M(x,y)$ rejects

We say $M$ “decides” the predicate $R$.

EXAMPLES:
$R(x,y) = \text{“} x + y \text{ is less than 100}\text{”}$
$R(<N>,y) = \text{“} N \text{ halts on } y \text{ in at most 100 steps}\text{”}$

Kleene’s T predicate, $T(<M>, x, y): M$ accepts $x$ in $y$ steps

1. $x, y$ are positive integers or elements of $\Sigma^*$
Definition: A decidable predicate \( R(x,y) \) is some proposition about \( x \) and \( y \), where there is a TM \( M \) such that

for all \( x, y \), \( R(x,y) \) is TRUE \( \Rightarrow \) \( M(x,y) \) accepts
\( R(x,y) \) is FALSE \( \Rightarrow \) \( M(x,y) \) rejects

We say \( M \) “decides” the predicate \( R \).

EXAMPLES:
\( R(x,y) = \) “\( x + y \) is less than 100”
\( R(<N>,y) = \) “\( N \) halts on \( y \) in at most 100 steps”
Kleene’s T predicate, \( T(<M>, x, y) \): \( M \) accepts \( x \) in \( y \) steps.

Note: \( A \) is decidable \( \iff \) \( A = \{ x | R(x,\varepsilon) \} \),
for some decidable predicate \( R \).
Theorem

$$\sum_0^0 = \text{languages } \{ x \mid \exists y_1 \forall y_2 \exists y_3 \ldots Q y_n R(x, y_1, \ldots, y_n) \}$$

$$\Pi_0^0 = \text{languages } \{ x \mid \forall y_1 \exists y_2 \forall y_3 \ldots Q y_n R(x, y_1, \ldots, y_n) \}$$

$$\Delta_0^0 = \sum_0^0 \cap \Pi_0^0$$

Where $R$ is a decidable predicate
Theorem: A language $A$ is semi-decidable if and only if there is a decidable predicate $R(x, y)$ such that: $A = \{ x | \exists y \ R(x,y) \}$

Proof:

(1) If $A = \{ x | \exists y \ R(x,y) \}$ then $A$ is semi-decidable

Because we can enumerate over all $y$’s

(2) If $A$ is semi-decidable, then $A = \{ x | \exists y \ R(x,y) \}$

Let $M$ semi-decide $A$

Then, $A = \{ x | \exists y \ T(<M>, x, y) \}$ (Here $M$ is fixed.)

where

Kleene’s $T$ predicate, $T(<M>, x, y)$: $M$ accepts $x$ in $y$ steps.
**THE PAIRING FUNCTION**

**Theorem.** There is a 1-1 and onto computable function $< , >: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ and computable functions $\pi_1$ and $\pi_2 : \Sigma^* \rightarrow \Sigma^*$ such that

$z = <w, t> \implies \pi_1 (z) = w$ and $\pi_2(z) = t$

**Proof:** Let $w = w_1 \ldots w_n \in \Sigma^*$, $t \in \Sigma^*$.

Let $a, b \in \Sigma$, $a \neq b$.

$<w, t> := a w_1 \ldots a w_n b t$

$\pi_1 (z) := \text{“if } z \text{ has the form } a w_1 \ldots a w_n b t, \text{ then output } w_1 \ldots w_n, \text{ else output } \varepsilon”$

$\pi_2(z) := \text{“if } z \text{ has the form } a w_1 \ldots a w_n b t, \text{ then output } t, \text{ else output } \varepsilon”$
Theorem

$\sum^0_1 = \{ \text{semi-decidable sets} \}$
$= \text{languages of the form } \{ x \mid \exists y \ R(x,y) \}$

$\Pi^0_1 = \{ \text{complements of semi-decidable sets} \}$
$= \text{languages of the form } \{ x \mid \forall y \ R(x,y) \}$

$\Delta^0_1 = \{ \text{decidable sets} \}$
$= \sum^0_1 \cap \Pi^0_1$

Where $R$ is a decidable predicate
Theorem

\[ \sum^0_2 = \{ \text{sets semi-decidable in some semi-dec. } B \} \]

= languages of the form \( \{ x | \exists y_1 \forall y_2 \ R(x, y_1, y_2) \} \)

\[ \Pi^0_2 = \{ \text{complements of } \sum^0_2 \text{ sets} \} \]

= languages of the form \( \{ x | \forall y_1 \exists y_2 \ R(x, y_1, y_2) \} \)

\[ \Delta^0_2 = \sum^0_2 \cap \Pi^0_2 \]

Where \( R \) is a decidable predicate
\[ \Sigma^0_1 = \text{languages of the form } \{ x \mid \exists y R(x,y) \} \]

We know that \( \text{A}_{\text{TM}} \) is in \( \Sigma^0_1 \) Why?

Show it can be described in this form:

\[ \text{A}_{\text{TM}} = \{ \langle M, w \rangle \mid \exists t [M \text{ accepts } w \text{ in } t \text{ steps}] \} \]

decidable predicate

\[ \text{A}_{\text{TM}} = \{ \langle M, w \rangle \mid \exists t T (\langle M \rangle, w, t) \} \]

\[ \text{A}_{\text{TM}} = \{ \langle M, w \rangle \mid \exists v (v \text{ is an accepting computation history of } M \text{ on } w) \} \]
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \sum_1 \cap \Pi_2 = \sum_2 \cap \Pi_2 \]

\( A_{TM} \)
\( \Pi^0_1 \) = languages of the form \( \{ x \mid \forall y \ R(x,y) \} \)

Show that \( \text{EMPTY} \) (ie, \( E_{TM} \)) = \( \{ M \mid L(M) = \emptyset \} \) is in \( \Pi^0_1 \)

\( \text{EMPTY} = \{ M \mid \forall w \forall t \ [M \ \text{doesn’t accept} \ w \ \text{in} \ t \ \text{steps}] \} \)

**two quantifiers??**

**decidable predicate**
\[ \Pi^0_1 = \text{languages of the form } \{ x | \forall y \ R(x,y) \} \]

Show that EMPTY (ie, \( E_{TM} \)) = \{ M | L(M) = \emptyset \} is in \( \Pi^0_1 \)

EMPTY = \{ M | \forall w \forall t \ [ \neg T(<M>, w, t) ] \} 

two quantifiers?? decidable predicate
THE PAIRING FUNCTION

Theorem. There is a 1-1 and onto computable function \( <, > : \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \) and computable functions \( \pi_1 \) and \( \pi_2 : \Sigma^* \rightarrow \Sigma^* \) such that

\[ z = <w, t> \implies \pi_1(z) = w \text{ and } \pi_2(z) = t \]

\[ \text{EMPTY} = \{ M \mid \forall w \forall t \ [M \text{ doesn't accept } w \text{ in } t \text{ steps}] \} \]

\[ \text{EMPTY} = \{ M \mid \forall z \ [M \text{ doesn't accept } \pi_1(z) \text{ in } \pi_2(z) \text{ steps}] \} \]

\[ \text{EMPTY} = \{ M \mid \forall z [ \neg T(<M>, \pi_1(z), \pi_2(z)) ] \} \]
Theorem. There is a 1-1 and onto computable function < , >: Σ* x Σ* → Σ* and computable functions π₁ and π₂ : Σ* → Σ* such that

\[ z = <w, t> \implies \pi_1(z) = w \text{ and } \pi_2(z) = t \]

Proof: Let \( w = w_1...w_n \in \Sigma^* \), \( t \in \Sigma^* \). Let \( a, b \in \Sigma, a \neq b \).

\[ <w, t> := a w_1... a w_n b t \]

\[ \pi_1(z) := \text{"if } z \text{ has the form } a w_1... a w_n b t, \text{ then output } w_1... w_n, \text{ else output } \varepsilon" \]

\[ \pi_2(z) := \text{"if } z \text{ has the form } a w_1... a w_n b t, \text{ then output } t, \text{ else output } \varepsilon" \]
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \sum_1^0 \cup \Pi_2^0 = \sum_2^0 \cap \Pi_2^0 \]

\[ \Delta_0^3 \]

\[ \Delta_2^2 \]

\[ \Delta_1^0 \]

\[ A_{TM} \]
\[ \Pi^0_2 = \text{languages of the form } \{ x \mid \forall y \exists z \ R(x,y,z) \} \]

Show that \( \text{TOTAL} = \{ M \mid M \text{ halts on all inputs} \} \)
is in \( \Pi^0_2 \)

\[
\text{TOTAL} = \{ M \mid \forall w \ \exists t \ [M \text{ halts on } w \text{ in } t \text{ steps}] \}
\]

decidable predicate
\[ \Pi^0_2 = \text{languages of the form } \{ x | \forall y \exists z R(x,y,z) \} \]

Show that \( \text{TOTAL} = \{ M | M \text{ halts on all inputs} \} \) is in \( \Pi^0_2 \)

\[ \text{TOTAL} = \{ M | \forall w \exists t \left[ T(<M>, w, t) \right] \} \]

decidable predicate
Decidable languages

Semi-decidable languages

$\sum_1^0 \cap \Pi_2^0 = \sum_2^0 \cap \Pi_2^0$

$\Delta_2^0$

$\Delta_0^3$

$\sum_3^0$

$\Pi_3^0$

$\Pi_2^0$

$\Pi_1^0$

Co-semi-decidable languages

$A_{TM}$

TOTAL

EMPTY
Show that \( \text{FIN} = \{ M \mid L(M) \text{ is finite} \} \) is in \( \sum_2^0 \)

\[
\text{FIN} = \{ M \mid \exists n \forall w \forall t \ [\text{Either } |w| < n, \text{ or } M \text{ doesn't accept } w \text{ in } t \text{ steps}] \}
\]

\[
\text{FIN} = \{ M \mid \exists n \forall w \forall t ( |w| < n \lor \neg T(<M>,w,t) ) \}
\]
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \Sigma^0_3 \]

\[ \Sigma^0_2 \]

\[ \Sigma^0_1 \]

\[ \Delta^0_3 \]

\[ \Delta^0_2 \]

\[ \Delta^0_1 \]

\[ \Pi^0_3 \]

\[ \Pi^0_2 \]

\[ \Pi^0_1 \]

\[ \text{FIN} \]

\[ \text{TOTAL} \]

\[ \text{EMPTY} \]

\[ \text{A}_{TM} \]

\[ = \Sigma^0_2 \cap \Pi^0_2 \]
Show that $\text{COF} = \{ M \mid \text{L(M) is cofinite} \}$ is in $\sum^0_2$

$$\text{COF} = \{ M \mid \exists n \forall w \exists t \left[ |w| > n \implies \text{M accept w in t steps} \right] \}$$
Decidable languages

Semi-decidable languages

COF

\[ \sum_0^3 \]

\[ \sum_0^2 \]

FIN

\[ \sum_0^1 \]

\[ \Delta^0_2 \cap \Pi^0_2 = \sum_0^2 \cap \Pi^0_2 \]

\[ \Delta^0_1 \]

\[ \Pi^0_3 \]

\[ \Pi^0_2 \]

\[ \Pi^0_1 \]

\[ \text{Co-semi-decidable languages} \]

\[ \text{EMPTY} \]

\[ \text{TM} \]

\[ \text{TOTAL} \]
Decidable languages

Semi-decidable languages

\[ \sum_0^3 \]

\[ \Delta_0^3 \]

\[ \Pi_0^3 \]

\[ \sum_2^0 \]

\[ \sum_1^0 \]

\[ \Delta_2^0 \]

\[ \Pi_2^0 \]

\[ \sum_2^0 \cap \Pi_2^0 = \sum_2^0 \]

\[ A_{TM} \]

\[ \emptyset \]

\[ \text{REG} \]

\[ \text{FIN} \]

\[ \text{TOTAL} \]

\[ \text{Co-semi-decidable languages} \]
Decidable languages

Semi-decidable languages

Δ₀¹

∑₀³

Δ₀³

Π₀³

Σ₀²

FIN

TOTAL

Π₀²

Δ₀²

Σ₀² ∩ Π₀²

A_TM

EMPTY

Decidable languages

Co-semi-decidable languages

Δ₀¹

Π₀¹
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

CFL

\[ \sum^0_3 \]

\[ \sum^0_2 \]

\[ \sum^0_1 \]

\[ \Delta^0_3 \]

\[ \Delta^0_2 \]

\[ \Delta^0_1 \]

\[ \Pi^0_3 \]

\[ \Pi^0_2 \]

\[ \Pi^0_1 \]

\[ \sum^0_2 \cap \Pi^0_2 = \sum^0_2 \cap \Pi^0_2 \]

\[ \text{TOTAL} \]

\[ \text{EMPTY} \]
Each is $m$-complete for its level in hierarchy and cannot go lower (by next Theorem, which shows the hierarchy does not collapse).

$L$ is $m$-complete for class $C$ if
i) $L \in C$ and
ii) $L$ is $m$-hard for $C$,

ie, for all $L' \in C$, $L' \leq_{m} L$
$A_{TM}$ is m-complete for class $C = \sum_0^1$

i) $A_{TM} \in C$

ii) $A_{TM}$ is m-hard for $C$,

Suppose $L \in C$. Show: $L \leq_m A_{TM}$

Let $M$ semi-decide $L$. Then Map

$\sum^* \rightarrow \sum^*$

where $w \rightarrow (M, w)$.

Then, $w \in L \iff (M, w) \in A_{TM}$

QED
FIN is m-complete for class \( C = \Sigma^0_2 \)

i) \( \text{FIN} \in C \)

ii) \( \text{FIN} \) is m-hard for \( C \),

Suppose \( L \in C \). Show: \( L \leq_m \text{FIN} \)

Suppose \( L = \{ w \mid \exists y \forall z \ R(w,y,z) \} \)
where \( R \) is decided by some TM \( D \)

Map \( \Sigma^* \rightarrow \Sigma^* \)
where \( w \rightarrow N_{D,w} \)
Suppose \( L \in \Sigma_2^0 \) i.e. \( L = \{ w \mid \exists y \forall z \ R(w,y,z) \} \)
where \( R \) is decided by some TM \( D \)

Show: \( L \leq_m \text{FIN} \)

Map \( \Sigma^* \rightarrow \Sigma^* \)
where \( w \rightarrow N_{D,w} \)

Define \( N_{D,w} \) on input \( s \):

1. Write down all strings \( y \) of length \( |s| \)
2. For each \( y \), try to find a \( z \) such that \( \neg R(w, y, z) \) and accept if all are successful

So, \( w \in L \iff N_{D,w} \in \text{FIN} \)
ORACLES not all powerful

The following problem cannot be decided, even by a TM with an oracle for the Halting Problem:

SUPERHALT = \{ (M,x) \mid M, \text{ with an oracle for the Halting Problem, halts on } x \}

Can use diagonalization here!
Suppose H decides SUPERHALT (with oracle)
Define \( D(X) = \text{“if } H(X,X) \text{ accepts (with oracle) then LOOP, else ACCEPT.”} \)
\( D(D) \) halts \( \iff \) \( H(D,D) \) accepts \( \iff \) \( D(D) \) loops…
ORACLES not all powerful

**Theorem:** The arithmetic hierarchy is strict. That is, the nth level contains a language that isn’t in any of the levels below n.

**Proof IDEA:** Same idea as the previous slide.

SUPERHALT^0 = HALT = \{ (M,x) | M halts on x \}.

SUPERHALT^1 = \{ (M,x) | M, with an oracle for the Halting Problem, halts on x \}

SUPERHALT^n = \{ (M,x) | M, with an oracle for SUPERHALT^{n-1}, halts on x \}
**Definition:** Let \( x \) in \( \{0,1\}^* \). The **shortest description** of \( x \), denoted as \( d(x) \), is the lexicographically shortest string \( \langle M, w \rangle \) s.t. \( M(w) \) halts with \( x \) on tape.

**Definition:** The **Kolmogorov complexity** of \( x \), denoted as \( K(x) \), is \( |d(x)| \).

**How to code \( \langle M, w \rangle \)?**

Assume \( w \) in \( \{0,1\}^* \) and we have a binary encoding of \( M \).
Theorem: There is a fixed $c$ so that for all $x$ in $\{0,1\}^*$, $K(x) \leq |x| + c$

Proof: Define $M = \text{"On input } w, \text{ halt."}$

On any string $x$, $M(x)$ halts with $x$ on its tape!

This implies

$$K(x) \leq |<M,x>| \leq 2|M| + |x| + 1 \leq |x| + c$$

(Note: $M$ is fixed for all $x$. So $|M|$ is constant)
**Theorem:** For all \( n \), there is an \( x \in \{0,1\}^n \) such that \( K(x) \geq n \)

“There are incompressible strings of every length”

**Proof:** (Number of binary strings of length \( n \)) = \( 2^n \)

(Number of **descriptions** of length < \( n \))

\( \leq \) (Number of **binary strings** of length < \( n \))

= \( 2^n - 1 \).

Therefore: there’s at least one \( n \)-bit string that doesn’t have a description of length < \( n \)
Theorem: For all $n$ and $c$, $\Pr_{x \in \{0,1\}^n}[ K(x) \geq n-c ] \geq 1 - 1/2^c$

“Most strings are fairly incompressible”

Proof: (Number of binary strings of length $n$) = $2^n$

(Number of descriptions of length < $n-c$) \leq (Number of binary strings of length < $n-c$) = $2^{n-c} - 1$.

So the probability that a random $x$ has $K(x) < n-c$ is at most $(2^{n-c} - 1)/2^n < 1/2^c$. 
DETERMINING COMPRESSIBILITY

COMPRESS = \{(x,n) \mid K(x) \leq n\}

Theorem: COMPRESS is undecidable!

Proof:

M = “On input \( x \in \{0,1\}^* \), let \( x' = 1x \)
Interpret \( x' \) as integer \( n \). (\(|x'| \leq \log n\))
Find first \( y \in \{0,1\}^* \) in lexicographical order, s.t. \((y,n) \notin \text{COMPRESS}\), then print \( y \) and halt.”

\( M(x) \) prints the first string \( y^* \) with \( K(y^*) > n \).
Thus \(<M,x>\) describes \( y^* \), and \(|<M,x>| \leq c + \log n\)
So \( n < K(y^*) \leq c + \log n \). CONTRADICTION!
Theorem: K is not computable

Proof:
M = “On input $x \in \{0,1\}^*$, let $x' = 1x$
   Interpret $x'$ as integer $n$. ($|x'| \leq \log n$)
   Find first $y \in \{0,1\}^*$ in lexicographical order,
   s. t. $K(y) > n$, then print $y$ and halt.”

$M(x)$ prints the first string $y^*$ with $K(y^*) > n$.
Thus $<M,x>$ describes $y^*$, and $|<M,x>| \leq c + \log n$
So $n < K(y^*) \leq c + \log n$. CONTRADICTION!
TIME COMPLEXITY AND POLYNOMIAL TIME;
NON DETERMINISTIC TURING MACHINES AND NP

THURSDAY Mar 20
COMPLEXITY THEORY

Studies what can and can’t be computed under limited resources such as time, space, etc

Today: Time complexity
MEASURING TIME COMPLEXITY

We measure time complexity by counting the elementary steps required for a machine to halt.

Consider the language $A = \{ 0^k1^k \mid k \geq 0 \}$

On input of length $n$:

1. Scan across the tape and reject if the string is not of the form $0^i1^j$

2. Repeat the following if both 0s and 1s remain on the tape:
   - Scan across the tape, crossing off a single 0 and a single 1

3. If 0s remain after all 1s have been crossed off, or vice-versa, reject. Otherwise accept.
Definition:
Suppose $M$ is a TM that halts on all inputs.
The running time or time-complexity of $M$ is the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$. 
ASYMPTOTIC ANALYSIS

\[ 5n^3 + 2n^2 + 22n + 6 = O(n^3) \]
Let \( f \) and \( g \) be two functions \( f, g : \mathbb{N} \rightarrow \mathbb{R}^+ \). We say that \( f(n) = O(g(n)) \) if there exist positive integers \( c \) and \( n_0 \) so that for every integer \( n \geq n_0 \)

\[
f(n) \leq cg(n)
\]

When \( f(n) = O(g(n)) \), we say that \( g(n) \) is an asymptotic upper bound for \( f(n) \)

\( f \) asymptotically NO MORE THAN \( g \)

\[
5n^3 + 2n^2 + 22n + 6 = O(n^3)
\]

If \( c = 6 \) and \( n_0 = 10 \), then \( 5n^3 + 2n^2 + 22n + 6 \leq cn^3 \)
\[ 2n^{4.1} + 200283n^4 + 2 = O(n^{4.1}) \]

\[ 3n\log_2 n + 5n \log_2 \log_2 n = O(n\log_2 n) \]

\[ n\log_{10} n^{78} = O(n\log_{10} n) \]

\[ \log_{10} n = \frac{\log_2 n}{\log_2 10} \]

\[ O(n\log_{10} n) = O(n\log_2 n) = O(n\log n) \]
Definition: \( \text{TIME}(t(n)) = \{ L \mid L \text{ is a language decided by a } O(t(n)) \text{ time Turing Machine} \} \)

\[
A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n^2)
\]
A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n \log n)

Cross off every other 0 and every other 1. If the # of 0s and 1s left on the tape is odd, reject
We can prove that a TM cannot decide A in less time than $O(n \log n)$

*7.49 Extra Credit. Let $f(n) = o(n \log n)$. Then $\text{Time}(f(n))$ contains only regular languages.

where $f(n) = o(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

ie, for all $c > 0$, $\exists n_0$ such that $f(n) < cg(n)$ for all $n \geq n_0$

$f$ asymptotically LESS THAN $g$
Can $A = \{ 0^k1^k \mid k \geq 0 \}$ be decided in time $O(n)$ with a two-tape TM?

Scan all 0s and copy them to the second tape. Scan all 1s, crossing off a 0 from the second tape for each 1.
Different models of computation yield different running times for the same language!
**Theorem:** Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$-time multi-tape TM has an equivalent $O(t(n)^2)$ single tape TM.

**Claim:** Simulating each step in the multi-tape machine uses at most $O(t(n))$ steps on a single-tape machine. Hence total time of simulation is $O(t(n)^2)$.
\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k \]
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine

\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k \]
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Analysis: (Note, $k$, the # of tapes, is fixed.)

Let $S$ be simulator

- Put $S$’s tape in proper format: $O(n)$ steps
- Two scans to simulate one step,
  1. to obtain info for next move $O(t(n))$ steps, why?
  2. to simulate it (may need to shift everything over to right possibly $k$ times): $O(t(n))$ steps, why?
\[ P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k) \]
NON-DETERMINISTIC TURING MACHINES AND NP
Definition: A Non-Deterministic TM is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L,R\})}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$
NON-DETERMINISTIC TMs

...are just like standard TMs, except:

1. The machine may proceed according to several possibilities

2. The machine accepts a string if there exists a path from start configuration to an accepting configuration
Deterministic Computation
accept or reject

Non-Deterministic Computation
accept
reject
Definition: Let $M$ be a NTM that is a decider (i.e., all branches halt on all inputs). The **running time** or **time-complexity** of $M$ is the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any branch of its computation on any input of length $n$. 
Theorem: Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$-time nondeterministic single-tape TM has an equivalent $2^{O(t(n))}$ deterministic single tape TM.
Definition: $\text{NTIME}(t(n)) = \{ L \mid L \text{ is decided by a } O(t(n))-\text{time non-deterministic Turing machine} \}$

$\text{TIME}(t(n)) \subseteq \text{NTIME}(t(n))$
A satisfying assignment is a setting of the variables that makes the formula true.

\[ \phi = (\neg x \land y) \lor z \]

\(x = 1, y = 1, z = 1\) is a satisfying assignment for \(\phi\).
A Boolean formula is **satisfiable** if there exists a satisfying assignment for it.

**YES**  \[ a \land b \land c \land \neg d \]

**NO**  \[ \neg (x \lor y) \land x \]

\[ \text{SAT} = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \} \]
A 3cnf-formula is of the form:

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_4 \lor x_2 \lor x_5) \land (x_3 \lor \neg x_2 \lor \neg x_1)\]

**YES**

\[(x_1 \lor \neg x_2 \lor x_1)\]

**NO**

\[(x_3 \lor x_1) \land (x_3 \lor \neg x_2 \lor \neg x_1)\]

**NO**

\[(x_1 \lor x_2 \lor x_3) \land (\neg x_4 \lor x_2 \lor x_1) \lor (x_3 \lor x_1 \lor \neg x_1)\]

**NO**

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_3 \land \neg x_2 \land \neg x_1)\]

**3SAT = \{ \phi | \phi \text{ is a satisfiable 3cnf-formula} \}**
3SAT = \{ \phi \mid \phi \text{ is a satisfiable 3cnf-formula} \}

**Theorem:** 3SAT \( \in \text{NTIME}(n^2) \)

On input \( \phi \):

1. Check if the formula is in 3cnf
2. For each variable, non-deterministically substitute it with 0 or 1

\[ (x \lor \neg y \lor x) \]

\[ (0 \lor \neg y \lor 0) \quad (1 \lor \neg y \lor 1) \]

\[ (0 \lor \neg 0 \lor 0) \quad (0 \lor \neg 1 \lor 0) \]

3. Test if the assignment satisfies \( \phi \)
\[ NP = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k) \]
Theorem: $L \in \text{NP} \iff$ if there exists a poly-time Turing machine $V(\text{erifier})$ with

$L = \{ x \mid \exists y (\text{witness}) \mid y \mid = \text{poly}(\mid x \mid) \text{ and } V(x, y) \text{ accepts} \}$

Proof:

(1) If $L = \{ x \mid \exists y \mid y \mid = \text{poly}(\mid x \mid) \text{ and } V(x, y) \text{ accepts} \}$ then $L \in \text{NP}$

Because we can guess $y$ and then run $V$

(2) If $L \in \text{NP}$ then

$L = \{ x \mid \exists y \mid y \mid = \text{poly}(\mid x \mid) \text{ and } V(x, y) \text{ accepts} \}$

Let $N$ be a non-deterministic poly-time TM that decides $L$ and define $V(x, y)$ to accept if $y$ is an accepting computation history of $N$ on $x$
$3\text{SAT} = \{ \phi \mid \exists y \text{ such that } y \text{ is a satisfying assignment to } \phi \text{ and } \phi \text{ is in 3cnf} \}$

$\text{SAT} = \{ \phi \mid \exists y \text{ such that } y \text{ is a satisfying assignment to } \phi \}$
A language is in NP if and only if there exist polynomial-length certificates* for membership to the language.

SAT is in NP because a satisfying assignment is a polynomial-length certificate that a formula is satisfiable.

* that can be verified in poly-time
HAMiltonian Paths
HAMPATH = \{ (G,s,t) \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t \} \\

**Theorem:** HAMPATH \(\in\) NP \\

The Hamilton path itself is a certificate
CLIQUE = \{ (G,k) \mid G \text{ is an undirected graph with a } k\text{-clique} \}

**Theorem:** CLIQUE ∈ NP

The k-clique itself is a certificate
NP = all the problems for which once you have the answer it is easy (i.e. efficient) to verify
P = NP?
POLY-TIME REDUCIBILITY

\( f : \Sigma^* \rightarrow \Sigma^* \) is a polynomial time computable function if some poly-time Turing machine \( M \), on every input \( w \), halts with just \( f(w) \) on its tape.

Language \( A \) is polynomial time reducible to language \( B \), written \( \leq_p B \), if there is a poly-time computable function \( f : \Sigma^* \rightarrow \Sigma^* \) such that:

\[
    w \in A \iff f(w) \in B
\]

\( f \) is called a polynomial time reduction of \( A \) to \( B \).
Theorem: If \( A \leq_p B \) and \( B \in P \), then \( A \in P \)

Proof: Let \( M_B \) be a poly-time (deterministic) TM that decides \( B \) and let \( f \) be a poly-time reduction from \( A \) to \( B \)

We build a machine \( M_A \) that decides \( A \) as follows:

On input \( w \):

1. Compute \( f(w) \)
2. Run \( M_B \) on \( f(w) \)
Definition: A language B is NP-complete if:

1. $B \in \text{NP}$
2. Every A in NP is poly-time reducible to B (i.e. $B$ is NP-hard)
Suppose B is NP-Complete

So, if B is NP-Complete and $B \in P$ then $NP = P$. Why?
Theorem (Cook-Levin): SAT is NP-complete

Corollary: SAT ∈ P if and only if P = NP
WWW.FLAC.WS

Read Chapter 7.3 of the book for next time
NP-COMPLETENESS: 
THE COOK-LEVIN THEOREM
Theorem (Cook-Levin.'71): SAT is NP-complete

Corollary: SAT ∈ P if and only if P = NP
Theorem (Cook-Levin): SAT is NP-complete

Proof:

(1) SAT ∈ NP

(2) Every language $A$ in NP is polynomial time reducible to SAT

We build a poly-time reduction from $A$ to SAT

The reduction turns a string $w$ into a 3-cnf formula $\phi$ such that $w \in A$ iff $\phi \in 3$-SAT.

$\phi$ will simulate the NP machine $N$ for $A$ on $w$.

Let $N$ be a non-deterministic TM that decides $A$ in time $n^k$. How do we know $N$ exists?
So proof will also show:

3-SAT is NP-Complete
The reduction \( f \) turns a string \( w \) into a 3-cnf formula \( \phi \) such that: \( w \in A \iff \phi \in 3\text{SAT} \).

\( \phi \) will “simulate” the NP machine \( N \) for \( A \) on \( w \).
Deterministic Computation

Non-Deterministic Computation

accept or reject

accept

reject

\[ \exp(n^k) \]
Suppose $A \in \text{NTIME}(n^k)$ and let $N$ be an NP machine for $A$. A **tableau** for $N$ on $w$ is an $n^k \times n^k$ table whose rows are the configurations of some possible computation of $N$ on input $w$.

<table>
<thead>
<tr>
<th></th>
<th>$q_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$\ldots$</th>
<th>$w_n$</th>
<th>$\square$</th>
<th>$\square$</th>
<th>$#$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$#$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$#$</td>
</tr>
<tr>
<td>$#$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$#$</td>
</tr>
<tr>
<td>$#$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$#$</td>
</tr>
<tr>
<td>$#$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$#$</td>
</tr>
</tbody>
</table>
A tableau is **accepting** if any row of the tableau is an accepting configuration.

Determining whether $N$ accepts $w$ is equivalent to determining whether there is an accepting tableau for $N$ on $w$.

Given $w$, our 3cnf-formula $\phi$ will describe a *generic* tableau for $N$ on $w$ (in fact, essentially *generic* for $N$ on any string $w$ of length $n$).

The 3cnf formula $\phi$ will be satisfiable *if and only if* there is an accepting tableau for $N$ on $w$. 
VARIABLES of $\phi$

Let $C = \mathbb{Q} \cup \Gamma \cup \{ \# \}$

Each of the $(n^k)^2$ entries of a tableau is a cell

cell[i,j] = the cell at row i and column j

For each $i$ and $j$ ($1 \leq i, j \leq n^k$) and for each $s \in C$
we have a variable $x_{i,j,s}$

# variables = $|C|n^{2k}$, ie $O(n^{2k})$, since $|C|$ only depends on $N$

These are the variables of $\phi$ and represent the contents of the cells

We will have: $x_{i,j,s} = 1 \iff$ cell[i,j] = s
$$X_{i,j,s} = 1$$

means

cell[ i, j ] = s
We now design $\phi$ so that a satisfying assignment to the variables $x_{i,j,s}$ corresponds to an accepting tableau for $N$ on $w$.

The formula $\phi$ will be the AND of four parts:

$$
\phi = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{move}}
$$

$\phi_{\text{cell}}$ ensures that for each $i,j$, exactly one $x_{i,j,s} = 1$.

$\phi_{\text{start}}$ ensures that the first row of the table is the starting (initial) configuration of $N$ on $w$.

$\phi_{\text{accept}}$ ensures* that an accepting configuration occurs somewhere in the table.

$\phi_{\text{move}}$ ensures* that every row is a configuration that legally follows from the previous config.

*if the other components of $\phi$ hold.
\( \phi_{\text{cell}} \) ensures that for each \( i,j \), exactly one \( x_{i,j,s} = 1 \)

\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i,j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right)^\wedge \left( \bigwedge_{s,t \in C, s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right]
\]

at least one variable is turned on

at most one variable is turned on
$$\phi_{\text{start}} = x_{1,1,#} \land x_{1,2,q_0} \land$$

$$x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land$$

$$x_{1,n+3,\square} \land \ldots \land x_{1,n^{k-1},\square} \land x_{1,n^k,#}$$
\( \phi_{\text{accept}} \) ensures that an accepting configuration occurs somewhere in the table

\[
\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}}
\]
\( \phi \) _{\text{move}} \) ensures that every row is a configuration that legally follows from the previous. It works by ensuring that each 2 × 3 “window” of cells is legal (does not violate N’s rules).
If $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

Which of the following windows are legal:
If $\delta(q_1,a) = \{(q_1,b,R)\}$ and $\delta(q_1,b) = \{(q_2,c,L), (q_2,a,R)\}$

Which of the following windows are legal:
CLAIM:
If
• the top row of the tableau is the start configuration, and
• and every window is legal,
Then
each row of the tableau is a configuration that legally follows the preceding one.
CLAIM:
If
• the top row of the tableau is the start configuration, and
• and every window is legal,

Then
each row of the tableau is a configuration that legally follows the preceding one.

Proof:
In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.
CLAIM:
If
• the top row of the tableau is the start configuration, and
• and every window is legal,

Then
each row of the tableau is a configuration that legally follows the preceding one.

Proof:
In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.

Case 1. center cell of window is a non-state symbol and not adjacent to a state symbol
CLAIM:
If
• the top row of the tableau is the start configuration, and
• and every window is legal,
Then
each row of the tableau is a configuration that legally follows the preceding one.

Proof:
In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.

Case 1. center cell of window is a non-state symbol and not adjacent to a state symbol
Case 2. center cell of window is a state symbol
<table>
<thead>
<tr>
<th>#</th>
<th>$q_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>…</th>
<th>$w_n$</th>
<th>□</th>
<th>…</th>
<th>□</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>ok</td>
<td>ok</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td>$w_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td>a₁</td>
<td>q</td>
<td>a₂</td>
<td>a₃</td>
<td>a₄</td>
<td>a₅</td>
<td>...</td>
<td>aₙ</td>
<td>□</td>
<td>...</td>
<td>□</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>-----</td>
<td>----</td>
<td>---</td>
<td>-----</td>
<td>---</td>
</tr>
<tr>
<td>#</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
<td>a₃</td>
<td>a₄</td>
<td>a₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a_1</td>
<td>q</td>
<td>a_2</td>
<td>a_3</td>
<td>a_4</td>
<td>a_5</td>
<td>...</td>
<td>a_n</td>
<td>□</td>
<td>...</td>
<td>□</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>---</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>---</td>
<td>-----</td>
<td>---</td>
</tr>
<tr>
<td>#</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
<td>a_3</td>
<td>a_4</td>
<td>a_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The table shows a sequence of symbols and variables with some marked as 'ok' and others as '□'.
So the lower configuration follows from the upper!!!
### The (i,j) Window

<table>
<thead>
<tr>
<th></th>
<th>col. j-1</th>
<th>col. j</th>
<th>col. j+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>row i</td>
<td>(i,j-1)</td>
<td>(i,j) a₂</td>
<td>(i,j+1) a₃</td>
</tr>
<tr>
<td></td>
<td>a₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>row i+1</td>
<td>(i+1,j-1)</td>
<td>(i+1,j) a₅</td>
<td>(i+1,j+1) a₆</td>
</tr>
</tbody>
</table>

- The (i,j) window consists of the elements $a_1, a_2, a_3, a_4, a_5,$ and $a_6$. 
- The elements in the window are indexed by $(i,j)$ and $(i+1,j)$. 
- The window spans across columns $j-1$ to $j+1$. 

This diagram illustrates how the window moves across the matrix, capturing a sliding subset of elements.
\[ \phi_{\text{move}} = \bigwedge_{1 \leq i, j \leq n^k} ( \text{the (i, j) window is legal} ) \]

the (i, j) window is legal =

\[ \bigvee_{a_1, \ldots, a_6} \left( x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a} \land x_{i+1,j-1,a} \land x_{i+1,j,a} \land x_{i+1,j+1,a} \right) \]

is a legal window

This is a disjunct over all (\( \leq |C|^6 \)) legal sequences (\( a_1, \ldots, a_6 \)).
\[ \phi_{move} = \bigwedge \left( \text{the (i, j) window is legal} \right) \]

\[ 1 \leq i, j \leq n^k \]

\[ \text{the (i, j) window is legal} = \lor \left( x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a} \land x_{i+1,j-1,a} \land x_{i+1,j,a} \land x_{i+1,j+1,a} \right) \]

is a legal window

This is a disjunct over all (\( \leq |C|^6 \)) legal sequences \((a_1, \ldots, a_6)\).

This disjunct is satisfiable

\[ \iff \]

There is some assignment to the cells (ie variables) in the window (i,j) that makes the window legal
\[ \phi_{\text{move}} = \bigwedge ( \text{the (i, j) window is legal} ) \]

\[ 1 \leq i, j \leq n^k \]

the (i, j) window is legal =

\[ \bigvee ( x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a} \land x_{i+1,j-1,a} \land x_{i+1,j,a} \land x_{i+1,j+1,a} ) \]

is a legal window

This is a disjunct over all \((\leq |C|^6)\) legal sequences \((a_1, \ldots, a_6)\).

So \( \phi_{\text{move}} \) is satisfiable

\[ \iff \]

There is some assignment to each of the variables that makes every window legal.
\( \phi_{\text{move}} = \bigwedge ( \text{the (i, j) window is legal}) \)

1 \leq i, j \leq n^k

the (i, j) window is legal =

\[ \bigvee_{a_1, \ldots, a_6} (x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a} \land x_{i+1,j-1,a} \land x_{i+1,j,a} \land x_{i+1,j+1,a}) \]

is a legal window

This is a disjunct over all \((\leq |C|^6)\) legal sequences \((a_1, \ldots, a_6)\).

Can re-write as equivalent conjunct:

\[ \equiv \bigwedge (x_{i,j-1,a} \lor x_{i,j,a} \lor x_{i,j+1,a} \lor x_{i+1,j-1,a} \lor x_{i+1,j,a} \lor x_{i+1,j+1,a}) a_1, \ldots, a_6 \]

ISN’T a legal window
\[ \phi = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{move}} \]

\( \phi \) is satisfiable (ie, there is some assignment to each of the variables s.t. \( \phi \) evaluates to 1)

\( \iff \)

there is some assignment to each of the variables s.t. \( \phi_{\text{cell}} \) and \( \phi_{\text{start}} \) and \( \phi_{\text{accept}} \) and \( \phi_{\text{move}} \) each evaluates to 1

\( \iff \)

There is some assignment of symbols to cells in the tableau such that:

- The first row of the tableau is a start configuration and
- Every row of the tableau is a configuration that follows from the preceding by the rules of \( N \) and
- One row is an accepting configuration

\( \iff \)

There is some accepting computation for \( N \) with input \( w \)
3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals
3-SAT?

How do we convert the whole thing into a 3-cnff formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals

If a clause has less than three variables:

\[ a \equiv (a \lor a \lor a), \quad (a \lor b) \equiv (a \lor b \lor b) \]
3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals

If a clause has less than three variables:

\[
a \equiv (a \lor a \lor a), \quad (a \lor b) \equiv (a \lor b \lor b)
\]

If a clause has more than three variables:

\[
(a \lor b \lor c \lor d) \equiv (a \lor b \lor z) \land (\neg z \lor c \lor d)
\]

\[
(a_1 \lor a_2 \lor \ldots \lor a_t) \equiv \left( a_1 \lor a_2 \lor z_1 \right) \land \left( \neg z_1 \lor a_3 \lor z_2 \right) \land \left( \neg z_2 \lor a_4 \lor z_3 \right) \ldots
\]
WHAT’S THE LENGTH OF $\phi$?
\[ \phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left( \bigvee_{s \in C} x_{i,j,s} \right)^\wedge \left( \bigwedge_{s,t \in C, s \neq t} (\neg x_{i,j,s} \lor \neg x_{i,j,t}) \right) \]

If a clause has less than three variables:

\[ (a \lor b) = (a \lor b \lor b) \]
\[ \phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left( \bigvee_{s \in C} x_{i,j,s} \right)^\wedge \left( \bigwedge_{s,t \in C \quad s \neq t} (\neg x_{i,j,s} \lor \neg x_{i,j,t}) \right) \]

\[ \text{O}(n^{2k}) \text{ clauses} \]

\[ \text{Length}(\phi_{\text{cell}}) = \text{O}(n^{2k} \log (n)) = \text{O}(n^{2k} \log n) \]

\[ \text{length(indices)} \]
\[ \phi_{\text{start}} = x_{1,1,#} \land x_{1,2,q_0} \land x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land x_{1,n+3,\square} \land \ldots \land x_{1,n^{k-1},\square} \land x_{1,n^k,#} \]

\[ = (x_{1,1,#} \lor x_{1,1,#} \lor x_{1,1,#}) \land (x_{1,2,q_0} \lor x_{1,2,q_0} \lor x_{1,2,q_0}) \land \ldots \land (x_{1,n^k,#} \lor x_{1,n^k,#} \lor x_{1,n^k,#}) \]
\[ \phi_{\text{start}} = x_{1,1,#} \land x_{1,2,q} \land x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land x_{1,n+3,\square} \land \ldots \land x_{1,n^k-1,\square} \land x_{1,n^k,#} \]

\[ O(n^k) \]
$$\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}}$$

$$(a_1 \lor a_2 \lor \ldots \lor a_t) = (a_1 \lor a_2 \lor z_1) \land (\neg z_1 \lor a_3 \lor z_2) \land (\neg z_2 \lor a_4 \lor z_3) \ldots$$
\( \phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}} \)

\( O(n^{2k}) \)
\[ \phi_{\text{move}} = \bigwedge \ (\text{the } (i, j) \text{ window is legal}) \]

\[ 1 \leq i, j \leq n^k \]

the (i, j) window is legal =
\[ \bigwedge_{a_1, \ldots, a_6} \left( \overline{x_{i,j-1,a_1}} \lor \overline{x_{i,j,a_2}} \lor \overline{x_{i,j+1,a_3}} \lor \overline{x_{i+1,j-1,a_4}} \lor \overline{x_{i+1,j,a_5}} \lor \overline{x_{i+1,j+1,a_6}} \right) \]

ISN’T a legal window

This is a conjunct over all (\leq |C|^6) illegal sequences \((a_1, \ldots, a_6)\).

\[ O(n^{2k}) \]
Theorem (Cook-Levin): 3-SAT is NP-complete

Corollary: 3-SAT \( \in \) P if and only if P = NP