**Formal languages**

- **Alphabet**: finite set of symbols  
  Ex: \( \Sigma = \{a,b,c\} \)

- **String**: finite symbol sequence  
  Ex: \( w = abcaabbcc \)

- **Length**: # of symbols  
  Ex: \( |bca| = 3 \)

- **String concatenation**: \( w_1 w_2 \) or \( w_1 \cdot w_2 \)  
  Ex: \( w_1 = aaa \quad w_2 = bb \)  
  \( w_1w_2 = aaabb \)
• **String exponentiation:**  
\[ w^k = \text{www...w} \ (k \text{ times}) \]

• **String reversal:**  
\[ w^R \]
Ex: \((abbaabbb)^R = bbbaaabba\)

• **Empty string:**  
\[ \varepsilon \text{ or } ^\wedge \]
Ex: \(\forall w \ w\varepsilon = \varepsilon w = w\)
\[ |\varepsilon| = 0 \]

• **Language:** set of strings
Ex: \(\{abc, abab, baaaa\}\)

• **Infinite language:** \(|L|>k \ \forall k \in \mathbb{Z}\)
\(\text{i.e. } \forall k \in \mathbb{Z} \ \exists w \in L \ \exists |w|>k\)
Ex:  
\[ \{a^n \ | \ n \geq 1\}\]
\[ = \{a, aa, aaa, ...\}\]
• **Language Concatenation:**

\[ L_1 L_2 = \{ w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2 \} \]

\[ LL = L^2 \]

\[ L^k = LL^{k-1} \quad L^0 = \{ \varepsilon \} \]

Ex: \( \{a, b\} \{1, 2, 3\} \)

\[ = \{a1, a2, a3, b1, b2, b3\} \]

• **Kleene closure:**

\[ L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots \]

\[ L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots \]

\[ L^+ = LL^* \]
• **Trivial language:**  \{\varepsilon\}

• **Empty language:**  \emptyset

• **All finite strings:**  \Sigma^*

\((L^*)^* = L^*\)

\(L \subseteq \Sigma^* \; \forall L\)

• **\(\Sigma^*\) is countable:**  \(|\Sigma^*| = |\mathbb{Z}|\)

a, b, ..., aa, ab, ..., aaa, aab, ...

• **2\(\Sigma^*\) is uncountable (why?)**
An algorithm is a “string”

Ex: “main() {int k; k=1;}”

$\Rightarrow |\{\text{algorithms}\}|$ is countable

A problem is a “language”

Ex: $\{2\$4, 3\$9, 4\$16, 5\$25, \ldots\}$

$\Rightarrow |\{\text{problems}\}|$ is uncountable

$\Rightarrow \exists$ more problems than algorithms!

[Alan Turing, 1936]

A description is a “string”

$\Rightarrow \exists$ undescendable languages!
Q: Prove that $2^\Sigma^*$ is uncountable.
## Machine Models

“The Chomsky hierarchy”

<table>
<thead>
<tr>
<th>Language type</th>
<th>Automata type</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>finite</td>
</tr>
<tr>
<td>context-free</td>
<td>pushdown</td>
</tr>
<tr>
<td>context-sensitive</td>
<td>linear-bounded</td>
</tr>
<tr>
<td>unrestricted</td>
<td>Turing</td>
</tr>
</tbody>
</table>
Finite Automata

Idea: “machine” changes states while processing symbols, one at a time.

1) Finite set of states \( Q \)
   \[ Q = \{ q_0, q_1, q_3, ..., q_k \} \]

2) Transition function \( \delta \)
   \[ \delta: Q \times \Sigma \rightarrow Q \]

3) Initial state: \( q_0 \in Q \)

4) Final states: \( F \subseteq Q \)

Def: FA \( M=(Q, \Sigma, \delta, q_0, F) \)
Pictorial Representation

- **State:** [Diagram of State]
- **Initial state:** [Diagram of Initial State]
- **Final State:** [Diagram of Final State]
- **Transition:** [Diagram of Transition]

Ex: FA accepting all odd-length strings over $\Sigma = \{a\}$:
String Recognition

Automaton consume string $w \in \Sigma^*$, one symbol at a time, & change states.

- **Acceptance**: end in a final state

- **Rejection**: anything else (including hang-up)

**Ex:**

![Diagram of a simple automaton](image-url)
Language Recognition

Extend $\delta$ to strings:

$$\delta: Q \times \Sigma^* \rightarrow Q$$

$$\delta(q_0, wx) = \delta(\delta(q_0, w), x)$$

$$\delta(q, \varepsilon) = q$$

$L(M) =$ set of strings accepted by $M$

$$= \{ w \in \Sigma^* | \delta(q_0, w) \in F \}$$

"regular" $\equiv$ accepted by some FA
Ex: All strings over $\Sigma = \{0, 1\}$ with odd # of 0’s and odd # of 1’s:

Q: All strings over $\Sigma = \{0, 1\}$ with even # of 0’s and 1’s.

Q: All strings in $\{0,1,2\}^*$ with even #’s of 0’s, 1’s, and 2’s.
\[ \Sigma = \{a, b\} \]

Ex: \( L_1 = \{w | w \text{ has 3 consecutive } b's\} \)

Ex: \( L_2 = \{w | w \text{ has no 3 consecutive } b's\} \)

Note: \( L_2 = \Sigma^* - L_1 \)
Thm: If \( L \) is regular, so is \( L' \).

Proof: Complement the final state set: i.e., given \( M=(Q,\Sigma,\delta,q_0,F) \), construct \( FA (Q,\Sigma,\delta,q_0,Q-F) \) accepting \( \Sigma^*-L(M) \).

Ex:
Nondeterminism

- Generalization of determinism
- Many “next-moves”: \( \delta: Q \times \Sigma \rightarrow 2^Q \)
- Computation is a “tree”
- Acceptance: \( \exists \) path to accepting leaf

Ex: 6\(^{th}\) symbol from end is an “a”: 

![Diagram](image-url)
Thm: DFAs can simulate NFAs.

Proof: (powerset construction)

Every subset of Q becomes a single state in our new machine!

i.e., given NFA $M=(Q, \Sigma, \delta, \{q_0\}, F)$, construct DFA $M'=(Q, \Sigma, \delta', \{q_0\}, F)$ with

$\delta'([q_1, q_2, \ldots, q_i], a) = [p_1, p_2, \ldots, p_j]$ iff $\delta(\{q_1, q_2, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}$.

Note: size of new DFA can be exponential in size of old NFA!
Thm: \( \cap \) preserves regularity.

Proof: (cross product construction)

Given \( M_1 = (Q_1, \Sigma, \delta_1, q', F_1) \)

and \( M_2 = (Q_2, \Sigma, \delta_2, q'', F_2) \),

construct \( M = (Q, \Sigma, \delta, q, F) \)

\[
Q = Q_1 \times Q_2
\]

\[
q = (q', q'')
\]

\[
F = F_1 \times F_2
\]

\[
\delta : Q \times \Sigma \rightarrow Q
\]

\[
\delta((q_i, q_j), x) = (\delta_1(q_i, x), \delta_2(q_j, x))
\]

i.e., “parallel” simulation!
Thm: $\cup$ preserves regularity.

Proof: (by DeMorgan’s law)

$$L_1 \cup L_2 = \overline{L_1} \cap \overline{L_2}$$

Thm: $-$ preserves regularity.

Proof: $L_1 - L_2 = L_1 \cap \overline{L_2}$
Thm: $\varepsilon$-transitions do not increase recognition power of FAs.
Regular Expressions

- $\emptyset$
- $\{\varepsilon\}$
- $\{x\}$ $\forall x \in \Sigma$
- If $R, S$ regular expressions, then so are:
  - $(R+S)$
  - $RS$
  - $R^*$

Note: “+” denotes union

Ex: $a^*b^*$ $a^*+b^*$ $a(a+b)^*$
    $(a+b)^*a(a+b)^*$ $aaaaaa^*$
Thm: Any regular expression is accepted by some FA.

Proof: (by construction)

\[ q_0 \Rightarrow \emptyset \]
\[ q_0 \Rightarrow \{ \varepsilon \} \]
\[ q_0 \xrightarrow{x} q_1 \Rightarrow \{ x \} \quad \forall x \in \Sigma \]

Assume \( M_1 \) accepts \( L(R) \), \( M_2 \) accepts \( L(S) \)

\[ \Rightarrow (R+S) \]
Thm: Each FA language can be denoted by a regular expression.

Proof idea: construct regular expression for union of all paths from $q_0$ to $F$.

$\Rightarrow$ FAs $\equiv$ regular expressions
Regular Expression Identities

- $RS \neq SR$
- $R+S = S+R$
- $R(ST) = (RS)T$
- $R(S+T) = RS+RT$
- $(R+S)T = RT+ST$
- $\emptyset^* = \varepsilon$
- $(R^*)^* = R^*$
- $(\varepsilon + R)^* = R^*$
- $(R^*S^*)^* = (R+S)^*$
2-Way Finite Automata

- Can move backwards on input
  
  i.e., \( \delta : Q \times \Sigma \rightarrow Q \times \{ \text{left, right} \} \)

Thm: 2-Way capability does not increase the power of FAs.

Proof idea: “crossing sequences”
Pumping

Thm: Almost all strings in a regular language contain a pumpable substring.

\[ \exists \ N \ \in \ \mathbb{N} \ \forall \ z \in L, \ |z| \geq N \ \exists \ u,v,w \in \Sigma^* \ \in z=uvw, \ |uv| \leq N, \ |v| \geq 1, \ uv^i w \in L \ \forall \ i \geq 0. \]

**Note:** not every long string in \( L \) is of the form \( uv^i w \)!

Ex: Show \( L = \{ a^i b^i \mid i \geq 1 \} \) not regular.

Assume \( L \) regular.

\[ \exists \ k \geq N \ \in \ a^k b^k = uvw \in L, \ |v| \geq 1. \]

\[ \Rightarrow uv^i w \in L \ \forall \ i \geq 0. \]

But if \( v \in a^+ \) or \( v \in b^+ \) then \( uw \notin L \), else if \( v \in a^+ b^+ \) then \( uv^2 w \notin L \), a contradiction.
Decidable FA Questions

Def: A problem is **decidable** iff there exists an algorithm which can determine (within finite time) the correct answer for any instance.

Q1: Is $L(M) = \emptyset$ ?
Hint: look at $w \in |w| \leq N$

Q2: Is $L(M)$ infinite ?
Hint: look at $w \in N \leq |w| \leq 2N$

Q3: Is $L(M_1) = L(M_2)$ ?
Hint: look at $(L_1- L_2) \cup (L_2- L_1)$
Context-Free Grammars

- Finite set of variables $V$
- Finite set of terminals $T$
- Finite set of productions $P$
- Start symbol $S$

Productions: $A \rightarrow \alpha$

$A \in V \quad \alpha \in (V \cup T)^*$

Applying $A \rightarrow \alpha$ to $vAw$

ields $v\alpha w$

i.e., does not depend on "context"

$G = (V, T, P, S)$
Grammars as Generators

$L(G) = \{ w \in T^* | S \xrightarrow{*} w \}$

$L$ is CF $\iff \exists$ CFG $G \ni L = L(G)$

Ex: $G: S \rightarrow 0S1$

$S \rightarrow \varepsilon$

A derivation in $G$:

$S \rightarrow 0S1 \rightarrow 00S11 \rightarrow 000S111 \rightarrow \ldots \rightarrow 0^kS1^k \rightarrow 0^k\varepsilon 1^k = 0^k1^k$

$\Rightarrow L(G) = \{ a^i b^i | i \geq 1 \}$ is CF

Q: If $L$ is CF, is $\overline{L}$ necessarily CF?
Pushdown Automata

PDA $\equiv$ FA with a stack:

$L(M) =$ language accepted by PDA $M$
Pushdown Automata (PDA)

1) Finite set of states $Q$

2) Transition function $\delta$

3) A stack

- $Q = \{q_0, q_1, q_3, ..., q_k\}$
- Input alphabet $\Sigma$
- Stack alphabet $\Gamma$
- $\delta: Q \times (\Sigma+\varepsilon+\emptyset) \times (\Gamma+\varepsilon+\$) \rightarrow Q \times \Gamma^*$
- Initial state: $q_0 \in Q$
- Initial stack: $\$
- Final states: $F \subseteq Q$

PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \$, $F)$
Ex: $\Sigma = \{0, 1\}$  $\Gamma = \{x\}$

$$L(M) = \{ a^i b^i | i \geq 1 \}$$

$M$ Fully Specified:
Nondeterministic PDAs

Many “next-moves”:

\[ \delta: Q \times (\Sigma + \varepsilon + \varnothing) \times (\Gamma + \varepsilon + \$) \rightarrow 2^{Q \times \Gamma^*} \]

M is deterministic if:

- \[ \forall q \in Q, z \in \Gamma \]
  \[ \delta(q, \varepsilon, z) \neq \emptyset \implies \delta(q, a, z) = \emptyset \quad \forall a \in \Sigma \]
  i.e., \varepsilon\text{-}moves are not allowed along with input-dependent moves

- \[ \forall q \in Q, z \in \Gamma, a \in \Sigma \cup \{\varepsilon\} \quad |\delta(q, a, z)| \leq 1 \]
  i.e., no choices allowed

Otherwise, M is nondeterministic.
Ex: \( L_1 = \{ \text{ww}^R \mid w \in \{a,b\}^* \} \)

\[ \begin{array}{c}
q_0 \xrightarrow{a/\varepsilon/a} q_1 \\
q_0 \xrightarrow{b/\varepsilon/b} q_1 \\
q_1 \xrightarrow{\varepsilon/\varepsilon/\varepsilon} q_2 \\
q_1 \xrightarrow{\varepsilon/\varepsilon/\varepsilon} q_2
\end{array} \]

vs. \( L_2 = \{ \text{w#w}^R \mid w \in \{a,b,\#\}^* \} \)

\[ \begin{array}{c}
q_0 \xrightarrow{a/\varepsilon/a} q_1 \\
q_0 \xrightarrow{b/\varepsilon/b} q_1 \\
q_0 \xrightarrow{\#/\varepsilon/\varepsilon} q_1 \\
q_1 \xrightarrow{\varepsilon/\varepsilon/\varepsilon} q_2 \\
q_1 \xrightarrow{\varepsilon/\varepsilon/\varepsilon} q_2
\end{array} \]

Thm: \( \{\text{ww}^R \mid w \in \{a,b\}^* \} \) can not be accepted by any deterministic PDA.
Ambiguous CFGs

CFG $G$ is **ambiguous** if some word in $L(G)$ has $>1$ non-isomorphic derivations.

**EX:** $S \rightarrow SS \mid \varepsilon \mid a \mid b$

$S \rightarrow SS \rightarrow aa$

$S \rightarrow SS \rightarrow SSS \rightarrow aa$

CFL $L$ is **inherently ambiguous** if every CFG for $L$ is ambiguous.

**Ex:** the following is an ambiguous CFL:

$$\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$
Normal Forms

Chomsky normal form (CNF):

All rules of form: \( A \rightarrow BC \)
\( A \rightarrow a \)

Greibach normal form (GNF):

All rules of form: \( A \rightarrow a\alpha \)
where \( \alpha \in V^*, a \in T \)

Thm: Every CFL may be represented in CNF or GNF.
2-Way PDAs

- Can move backwards on input i.e., $\delta: Q \times (\Sigma + \varepsilon + \emptyset) \times (\Gamma + \varepsilon + \$$) \rightarrow Q \times \Gamma^* \times \{L, R\}$

Thm: a 2-Way "input-tape" can increase the power of PDAs.
Pumping for CFLs

Thm: Almost all strings in a context-free language contain a pumpable substring.

L is CF $\Rightarrow \exists N \in \mathbb{N} \; \forall z \in L, \; |z| \geq N \; \exists$

$u,v,w,x,y \in \Sigma^* \; \exists \; z=uvwxy, \; |vwx| \leq N, \; |vx| \geq 1,$

$uv^iwx^iy \in L \; \forall \; i \geq 0.$

Proof idea: put grammar for L into CNF and find a repeated variable along some path in a derivation of a long string.

Ex: Can show $L = \{ a^ib^ic^i | i \geq 1 \}$ not CF.
Thm: CFLs are closed under $\cup$.

Thm: CFLs are closed under Kleene.

Thm: CFLs are closed under $\cap$ with regular sets.
Thm: CFLs are not closed under $\cap$.

Thm: CFLs are not closed under complementation.
Decidable PDA Questions

Q1: Is $L(M) = \emptyset$ ?

Q2: Is $L(M)$ finite ?

Q3: Is $L(M)$ infinite ?

Q4: Is $w \in L(M)$ ?
Turing Machines

TM ≡ FA with a tape:

L(M) = language accepted by TM M
Turing Machine Model

1) Finite set of states $Q$

2) Transition function $\delta$

3) A semi-infinite tape

- $Q = \{q_0, q_1, q_3, \ldots, q_k\}$
- Tape alphabet $\Gamma$
- Input alphabet $\Sigma \subseteq \Gamma$
- $\delta: Q \times (\Gamma + \beta) \rightarrow Q \times \Gamma \times \{L, R\}$
- Initial state: $q_0 \in Q$
- Blank symbol: $\beta$
- Final states: $F \subseteq Q$

$TM \ M = (Q, \Sigma, \Gamma, \delta, q_0, \beta, F)$
Nondeterministic TMs

Many “next-moves”:
\[ \delta: Q \times (\Gamma + \beta) \rightarrow 2^{Q \times \Gamma \times \{L, R\}} \]

M is deterministic if:
\[ \forall q \in Q, a \in \Gamma \ | \delta(q, a)| \leq 1 \]

i.e., no multiple choices allowed

Otherwise, M is nondeterministic.

Q: Does non-determinism increase the power of TMs?
Problem 1: design a TM to accept
\[ L_1 = \{ 0^n1^n2^n \mid n \geq 1 \} \]

Problem 2: design a TM to accept
\[ L_2 = \{ n$12^n \mid n \geq 1 \} \]

Problem 3: give generating grammars for \( L_1 \) and \( L_2 \).
**TM “Enhancements”**

- Larger alphabet
- Doubly-infinite tape
- Multiple tapes
- Multiple heads
- Multi-dimensional tape
- Non-determinism
- Combinations

**Thm:** These "enhancements" do not increase the power of TMs.

- "Oracles"?
Church's Thesis

Computable $\equiv$ TM computable

- Recognition vs. enumeration
- Recursively Enumerable:
  $\equiv$ TM-generatable
  $\equiv$ recongizable by TM that stops on all "yes" instances
- Recursive:
  $\equiv$ TM-generatable in order
  $\equiv$ recongizable by TM that stops on all instances
Decidability

Def: L is decidable if \( \exists \) an always-halting TM \( M \) that accepts \( L \).

Ex: Finite languages are decidable.
Ex: Regular languages are decidable.
Ex: CF languages are decidable.
Ex: Halting problem is not decidable.

Def: L is (Turing-)recognizable if \( \exists \) an TM \( M \) that accepts \( L \).

Ex: Halting problem is recognizable.
Ex: Its complement is not recognizable.

Theorem: L is decidable iff \( L \) and its complement are Turing-recognizable.
Reducibilities

Def: language A is reducible to a lang. B if \( \exists \) computable function \( f: \Sigma^* \rightarrow \Sigma^* \) where \( \forall w, w \in A \Leftrightarrow f(w) \in B \)

\( f \) is called a "reduction" of A to B

Denotation: \( A \leq B \)

Theorem: If \( A \leq B \) and B is decidable, then A is decidable.

Corrolary: If \( A \leq B \) and A is undecidable, then B is undecidable.
Examples of Reductions:

$H_\varepsilon$: Given TM M, does M halt on $\varepsilon$?

Reduction from the Halting Problem $H$:

Given an arbitrary TM M & string w, construct a new TM M' that for input x:

1) Completely erases the input tape;
2) Writes w onto the input tape;
3) Simulates M on the input w;
4) Accepts $\iff$ M accepts w.

$\Rightarrow$ M’ halts on $\varepsilon$ $\iff$ M halts on w

An algorithm for $H_\varepsilon$ can thus be used to solve the general Halting Problem $H$!

$\Rightarrow$ $H_\varepsilon$ is not decidable
L₀: Given TM M, is L(M)=Ø?

Reduction from the Halting Problem H:

Given an arbitrary TM M & string w, construct a new TM M' that for input x:

1) Completely erases input tape;
2) Writes w onto input tape;
3) Simulates M on the input w;
4) Accepts ⇔ M accepts w.

⇒ L(M')=∑* if M halts on w;
    L(M')=Ø if M does not halt on w

An algorithm for L₀ can thus be used to solve the general Halting Problem H!

⇒ L₀ is not decidable
L\textsubscript{reg}: Given TM M, is L(M) regular?

Reduction from the Halting Problem H:

Given an arbitrary TM M & string w, construct a new TM M' that on input x:

1) Accepts if x\in\{0^n1^n\}; else:
2) Completely erases input tape;
3) Writes w onto input tape;
4) Simulate M on the input w;
5) Accepts \iff M accepts w.

\Rightarrow L(M')=\sum^* \text{ if } M \text{ halts on } w
\Rightarrow L(M')=\{0^n1^n\} \text{ otherwise}

An algorithm for L\textsubscript{reg} can thus be used to solve the general Halting Problem H!

\Rightarrow L\textsubscript{reg} is not decidable
Rice's Theorem

Def: *property* $P \equiv$ a set of languages

Ex: $P_1 = \{L \mid L \text{ is a regular language}\}$
    $P_2 = \{L \mid L \text{ is a finite language}\}$
    $P_3 = \{L \mid L \text{ contains the string } 011\}$

$L$ is said to have the property $P$ if $L \in P$

Ex: $a^* b^*$ has property $P_1$
    $a^* b^*$ does not have property $P_2$

Def: A property $P$ is trivial if $P = \emptyset$ or
     $P$ contains all R.E. languages
The trivial properties are decidable:

Ex: $P_{all} = \{L \mid L \text{ is an R.E. language}\}$

$P_{none} = \emptyset$

TM that determines property $P_{all}$:

$x \rightarrow M_{all} \rightarrow \text{yes}$

TM that determines property $P_{none}$:

$x \rightarrow M_{none} \rightarrow \text{no}$

Q: What other properties are decidable?

A: None!
Thm [Rice]: All non-trivial properties of the recursive languages are undecidable.

Proof: Let $P$ be a non-trivial property.

Select $L \in P$ (WLOG assume $L \neq \emptyset$, otherwise consider complement of $P$).

Assume $M_L$ decides $L$:

$x$ \[\rightarrow M_L \rightarrow \text{yes} \quad \text{no}\]

(note: input $x$ is a TM description)

Reduction strategy: “solve” the halting problem using $M_L$ as an “oracle”.
Given TM M & string w, construct M’:

Analyze language of M’:
Note: L(M’) is either L or Ø, and
\[
\begin{align*}
L(M') = L & \iff w \in L(M) \\
L(M') = \emptyset & \iff w \notin L(M)
\end{align*}
\]

Assume \( \exists \) TM \( M_P \) that decides P. Now use \( M_P \) to determine if M' has property P, i.e., if \( L(M') = L \in P \)

But this solves the halting problem for an arbitrary M(w)!
\( \Rightarrow \) Property P is not decidable!
Ex: Undecidable Properties:

- L is empty
- L is finite
- L is infinite
- L is regular
- L is context-free
- L is decidable
- L = ∅
- L = Σ*
- L contains an odd string
- L contains a palindrome
Algorithms

• Existance
• Efficiency

Analysis

• Correctness
• Time
• Space
• Other resources

Worst case analysis
(as function of input size |w|)

Asymptotic growth: $\Theta$ $\Omega$ $\Theta$ $o$
Upper Bounds

\[ f(n) = O(g(n)) \iff \exists c, k > 0 \quad \exists |f(n)| \leq c \cdot |g(n)| \quad \forall n > k \]

\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} \text{ exists} \]

“\( f(n) \) is big-O of \( g(n) \)”

Ex: \( n = O(n^2) \)

\[ 33n + 17 = O(n) \]

\[ n^8 - n^7 = O(n^{123}) \]

\[ n^{100} = O(2^n) \]

\[ 213 = O(1) \]
Lower Bounds

\[ f(n) = \Omega(g(n)) \iff g(n) = \Theta(f(n)) \]

\[ \lim_{n \to \infty} \frac{g(n)}{f(n)} \text{ exists} \]

"f(n) is Omega of g(n)"

Ex: \( 100n = \Omega(n) \)

\( 33n + 17 = \Omega(\log n) \)

\( n^8 - n^7 = \Omega(n^8) \)

\( 213 = \Omega(1/n) \)

\( 1 = \Omega(213) \)
Tight Bounds

\[ f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \land g(n) = O(f(n)) \]

“\( f(n) \) is Theta of \( g(n) \)”

Ex: \( 100n = \Theta(n) \)

\[ 33n + 17 + \log n = \Theta(n) \]

\[ n^8 - n^7 - n^{-13} = \Theta(n^8) \]

\[ 213 = \Theta(1) \]

\[ 3 + \cos(2^n) = \Theta(1) \]
Loose Bounds

\[ f(n) = o(g(n)) \iff f(n) = O(g(n)) \land f(n) \neq \Omega(g(n)) \]

\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \]

“\( f(n) \) is little-o of \( g(n) \)”

Ex: \( 100n = o(n \log n) \)

\( 33n + 17 + \log n = o(n^2) \)

\( n^8 - n^7 - n^{-13} = o(2^n) \)

\( 213 = o(\log n) \)

\( 3 + \cos(2^n) = o(\sqrt{n}) \)
Growth Laws

Let \( f_1(n) = O(g_1(n)) \) and \( f_2(n) = O(g_2(n)) \)

Thm: \( f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n))) \)

Thm: \( f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)) \)

Thm: \( n^k = O(c^n) \quad \forall \ c, k > 0 \)

Ex: \( n^{1000} = O(1.001^n) \)
Recurrences

\[ T(n) = a \cdot T(n/b) + f(n) \]

let \( c = \log_b a \)

**Thm:**

\[ f(n) = O(n^{c-\varepsilon}) \implies T(n) = \Theta(n^c) \]
\[ f(n) = \Theta(n^c) \implies T(n) = \Theta(n^c \log n) \]
\[ f(n) = \Omega(n^{c+\varepsilon}) \land a \cdot f(n/b) \leq d \cdot f(n) \]
\[ \forall d < 1, \ n > n_0 \implies T(n) = \Theta(f(n)) \]

**Ex:** \( T(n) = 9T(n/3) + n \implies T(n) = \Theta(n^2) \)

\[ T(n) = T(2n/3) + 1 \implies T(n) = \Theta(\log n) \]
Stirling's Formula

\[ n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-2) \cdot (n-1) \cdot n \]

\[ n! \approx \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \cdot \left(1 + O\left(\frac{1}{n}\right)\right) \]

\[ n! \approx \left(\frac{n}{e}\right)^n \]

\[ \log(n!) = O(n \log n) \]

• Useful in analyses and bounds
Resource-Bounded Computations

Previously: can it be done?
Now: how efficiently can we do it?

Conserve computational resources:

   \[ \Rightarrow \text{Time, space, other resources?} \]

Def: \( L \) is \underline{decidable within time} \( O(t(n)) \)
if some TM \( M \) for \( L \) halts on all \( w \in \Sigma^* \) within \( O(t(|w|)) \) steps / time.

Def: \( L \) is \underline{decidable within space} \( O(s(n)) \)
if some TM \( M \) for \( L \) halts on all \( w \in \Sigma^* \) while never using more than \( O(s(|w|)) \) space / tape cells.
Complexity Classes

Def: $\text{DTIME}(t(n)) = \{L \mid L \text{ is decidable by a } O(t(n))-\text{time deterministic TM}\}$

Def: $\text{NTIME}(t(n)) = \{L \mid L \text{ is decidable by a } O(t(n))-\text{time non-deterministic TM}\}$

Def: $\text{DSPACE}(s(n)) = \{L \mid L \text{ decidable by a } O(s(n))-\text{space deterministic TM}\}$

Def: $\text{NSPACE}(s(n)) = \{L \mid L \text{ decidable by a } O(s(n))-\text{space non-deterministic TM}\}$

Note: Times depends on #tapes in TM.
Note: For multi-tape TM’s, input tape space does not count in $s(n)$.

Theorem: Space is \#tapes-independent.

Theorem: If $s(n) < s'(n) \forall n>1$ then:
\[
\text{DSPACE}(s(n)) \subseteq \text{DSPACE}(s'(n)) \\
\text{NSPACE}(s(n)) \subseteq \text{NSPACE}(s'(n))
\]

Theorem: If $t(n) < t'(n) \forall n>1$ then:
\[
\text{DTIME}(t(n)) \subseteq \text{DTIME}(t'(n)) \\
\text{NTIME}(t(n)) \subseteq \text{NTIME}(t'(n))
\]

Ex: $\text{NTIME}(n) \subseteq \text{NTIME}(n^2)$
Ex: $\text{DSPACE}(\log n) \subseteq \text{DSPACE}(n)$
Ex: $L_1 = \{0^n1^n | n > 0\}$:

For 1-tape TM’s:
- $L_1 \in \text{DTIME}(n^2)$
- $L_1 \in \text{DSPACE}(n)$
- $L_1 \in \text{DTIME}(n \log n)$

For 2-tape TM’s:
- $L_1 \in \text{DTIME}(n)$
- $L_1 \in \text{DSPACE}(\log n)$

Ex: $L_2 = \Sigma^*$
- $L_2 \in \text{DTIME}(n)$
- $L_2 \in \text{DSPACE}(1)$

Ex: $L_3 = \{w$w | w in $\Sigma^*\}$
- $L_3 \in \text{DTIME}(n^2)$
- $L_3 \in \text{DSPACE}(n)$
- $L_3 \in \text{DSPACE}(\log n)$
**Special Classes**

**Def:** \( P = \bigcup_{k} DTIME(n^k) \)

\( P \equiv \) deterministic polynomial time

**Def:** \( NP = \bigcup_{k} NTIME(n^k) \)

\( NP \equiv \) non-deterministic polynomial time

**Def:** \( PSPACE = \bigcup_{k} DSPACE(n^k) \)

\( = \bigcup_{k} NSPACE(n^k) \)

\( PSPACE \equiv \) polynomial space

**Def:** \( EXPTIME = \bigcup_{k} DTIME(2^{n^k}) \)

\( EXPTIME \equiv \) exponential time
Time Hierarchy

Theorem: for any \( t(n) > 0 \), there exists a decidable language \( L \not\in \text{DTIME}(t(n)) \).

(note: \( t(n) \) must computable & everywhere defined)

\( \Rightarrow \) No time complexity class contains all decidable languages!

\( \Rightarrow \) There are decidable languages that take arbitrarily long times to decide!

Proof: (diagonalization)

Consider lexicographic orders:
\( w_i \) for strings, \( M_i \) for TM’s, \( i > 0 \)

Define \( L = \{ w_i \mid M_i \text{ does not accept } w_i \text{ within } t(|w_i|) \text{ time/moves} \} \)
Claim: L is decidable (why?)

Q: is \( L \in \text{DTIME}(t(n)) \) ?

Assume yes, i.e., \( L = L(M_{i_0}) \) for some \( t(n) \)-time-bounded TM \( M_{i_0} \)

Consider whether \( w_{i_0} \in L \):

\[ w_{i_0} \in L \Rightarrow M_{i_0}(w_{i_0}) \text{ halts in } t(|w_{i_0}|) \text{ time} \]
\[ \Rightarrow w_{i_0} \notin L \text{ by } L \text{'s definition.} \]

\[ w_{i_0} \notin L \Rightarrow M_{i_0}(w_{i_0}) \text{ does not halts within } t(|w_{i_0}|) \text{ time} \Rightarrow w_{i_0} \in L \text{ by } L \text{'s definition.} \]

So, \((w_{i_0} \in L) \iff (w_{i_0} \notin L)\), a contradiction!

\[ \Rightarrow L \neq L(M_{i_0}) \text{ for any } t(n) \text{-time TM } M_{i_0} \]
\[ \Rightarrow L \notin \text{DTIME}(t(n)) \]
Space Hierarchy

Theorem: for any \( s(n) > 0 \), there exists a decidable language \( L \not\in \text{DSPACE}(s(n)) \).

(note: \( s(n) \) must computable & everywhere defined)

\[ \Rightarrow \] No space complexity class contains all decidable languages!

\[ \Rightarrow \] There are decidable languages that take arbitrarily large space to decide!

Proof: (diagonalization)

Consider lexicographic orders:
\[ w_i \text{ for strings, } M_i \text{ for TM’s, } i > 0 \]

Define \( L = \{ w_i \mid M_i \text{ does not accept } w_i \text{ within } s(|w_i|) \text{ space} \} \)
Claim: $L$ is decidable

(Q: how can we detect space-bounded $\infty$ loops?)

Q: is $L \in \text{DSAPCE}(s(n))$?

Assume yes, i.e., $L = L(M_{i_0})$ for some $s(n)$-space-bounded TM $M_{i_0}$

Consider whether $w_{i_0} \in L$:

$w_{i_0} \in L \Rightarrow M_{i_0}(w_{i_0})$ halts in $t(|w_{i_0}|)$ space
$\Rightarrow w_{i_0} \notin L$ by $L$’s definition.

$w_{i_0} \notin L \Rightarrow M_{i_0}(w_{i_0})$ does not halt within $t(|w_{i_0}|)$ space
$\Rightarrow w_{i_0} \in L$ by definition.

So, $(w_{i_0} \in L) \iff (w_{i_0} \notin L)$, a contradiction!

$\Rightarrow L \neq L(M_{i_0})$ for any $s(n)$-space TM $M_{i_0}$
$\Rightarrow L \notin \text{DSPACE}(s(n))$
Dense Space Hierarchy

Q: How much additional space does it take to recognize more languages?
A: Very little more!

Theorem: Given bounds $s_1$ and $s_2$ such that $\lim s_1(n) / s_2(n) = 0$ as $n \to \infty$, i.e., $s_1(n) = o(s_2(n))$, $\exists$ a decidable language $L$ such that $L \in \text{DSPACE}(s_2(n))$ but $L \not\in \text{DSPACE}(s_1(n))$.

(note: $s_2(n)$ must computable within $s_2(n)$ space)

$\Rightarrow \text{DSPACE}(o(s(n))) \not\subset \text{DSPACE}(s(n))$

Ex: $\text{DSPACE}(n) \not\subset \text{DSPACE}(n \log n)$
$\text{DSPACE}(n^2) \not\subset \text{DSPACE}(n^{2.001})$
$\text{DSPACE}(n^x) \not\subset \text{DSPACE}(n^y) \forall x < y$
Proof: (diagonalization)

Consider an encoding $M_i$ of TM’s
(note: this represents each R.E. language $\infty$ times)

Construct new TM $M'$ that for each $w$:
1) Mark $s_2(n)$ cells on tape; if the rest of this Simulation tries to exit marked calls, reject;
2) If input $w$ is not a valid TM encoding, reject;
3) If $w=M_i$, simulate TM $M_i$ on $w$, subject to:
   If $M_i$ accepts, reject;
   If $M_i$ rejects, accept;
   If $M_i$ runs for longer than $2^{s_2(|w|)}$, reject;
   (since $M_i$ is then in an $\infty$ loop!)

$\Rightarrow M'$ always halts within $s_2(n)$ space.
$\Rightarrow L \in \text{DSPACE}(s_2(n))$

Claim: $L(M')$ is not in $\text{DSPACE}(s_1(n))$
**Assume**: \( L(M') \) is in \( \text{DSPACE}(s_1(n)) \), where \( s_1(n) = o(s_2(n)) \)

\[ \Rightarrow \exists \ s_1(n) \text{-space } M'' \text{ that decides } L(M') \]

i.e., \( L(M'') = L(M') \)

Note: \( s_1(n) < s_2(n) \ \forall n > n_0 \)

Consider what happens when you run \( M' \) on \( w=M''0^n \), i.e., analyze the case of \( M' \) running on “padded” \( M''0^n \).

\[ \Rightarrow \] the simulation of \( M' \) on \( M''0^n \) in step (3) will run to completion

But \( M' \) will do the opposite of \( M'' \), so \( L(M') \) can not be the same as \( L(M'') \), a contradiction!

\[ \Rightarrow \ L(M') \notin \text{DSPACE}(s_1(n)) \]
Dense Time Hierarchy

Q: How much additional time does it take to recognize more languages?
A: At most a logarithmic factor more!

Theorem: Given $t_1$ and $t_2$ such that $t_1(n) \cdot \log(t_1(n)) = o(t_2(n))$, $\exists$ decidable language $L$ such that $L \in \text{DTIME}(t_2(n))$ but $L \notin \text{DTIME}(t_1(n))$.

(note: $t_2(n)$ must computable within $t_2(n)$ time)

$\Rightarrow \text{DTIME}(o(t(n) / \log t(n))) \nsubseteq \text{DTIME}(t(n))$

Ex: $\text{DTIME}(n) \nsubseteq \text{DTIME} (n \log^2 n)$
$\text{DTIME}(2^n) \nsubseteq \text{DTIME}(n^{2^2} 2^n)$
$\text{DTIME}(n^x) \nsubseteq \text{DTIME}(n^y)$ $\forall x < y$
Time/Space Relationships

Thm: $\text{DTIME}(f(n)) \subseteq \text{DSPACE}(f(n))$

Thm: $\text{DSPACE}(f(n)) \subseteq \text{DTIME}(c^{f(n)})$ for some $c$ depending on the language.

Thm: $\text{NTIME}(f(n)) \subseteq \text{DTIME}(c^{f(n)})$ for some $c$ depending on the language.

Thm: $\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f^2(n))$

Thm: $\text{NSPACE}(n^r) \subseteq \text{DSPACE}(n^{r+\varepsilon})$ for all $r>0$, $\varepsilon>0$. 
NP-Completeness

• Tractability
• Polynomial time
• Computation vs. verification
• Non-determinism
• Encodings
• Transformation & reducibilities
• P vs. NP
• "completeness"
A problem $L$ is NP-hard if:
1) all problems in NP reduce to $L$ in polynomial time.

A problem $L$ is NP-complete if:
1) $L$ is NP-hard; and
2) $L$ is in NP.

- One NPC problem is in $P \implies P=NP$

Open question: is $P=NP$?
**Satisfiability**

**SAT**: is a given n-variable boolean formula (in CNF) satisfiable?

**CNF** (Conjunctive Normal Form): i.e., product-of-sums

"satisfiable" ⇒ can be made "true"

**Ex:** \((x+y)(\overline{x} +z)\) is satisfiable

\((x+z)(\overline{x} )(\overline{z} )\) is not satisfiable

**3-SAT**: is a given n-var boolean formula (in 3-CNF) satisfiable?

**3-CNF**: three literals per clause

**Ex:** \((x_1+x_5+x_7)(x_3+\overline{x} _4+\overline{x} _5)\)
Cook's Theorem

**Thm**: SAT is NP-complete [Cook 1971]

**Pf idea**: given a non-deterministic polynomial-time TM $M$ and input $w$, construct a CNF formula that is satisfiable iff $M$ accepts $w$.

Use variables:

- $q[i,k] \Rightarrow$ at step $i$, $M$ is in state $k$
- $h[i,k] \Rightarrow$ at step $i$, read-write head scans tape cell $k$
- $s[i,j,k] \Rightarrow$ at step $i$, tape cell $j$ contains symbol $\Sigma_k$

$M$ always halts in polynomial time \( \Rightarrow \) # of variables is polynomial
Clauses for necessary restrictions:
- At each time i:
  - M is in exactly 1 state
  - r/w head scans exactly 1 cell
  - all cells contain exactly 1 symb
- Time 0 ⇒ initial state
- Time P(n) ⇒ final state
- Transitions from time i to time i+1 obey M's transition function

Resulting formula is satisfiable iff M accepts w.

Thm: 3-SAT is NP-complete
Pf idea: convert each long clause to an equivalent set of short ones:

\[(x+y+z+u+v+w)\]

⇒\[(x+y+a)(\overline{a} +z+b)(\overline{b} +u+c)(\overline{c} +v+w)\]
Q: is 1-SAT NP-complete?

Q: is 2-SAT NP-complete?
COLORABILITY: given a graph $G$ and integer $k$, is $G$ $k$-colorable? (different colors for adjacent nodes)

Ex:

Thm: 3-COLORABILITY is NPC

Proof: reduction from 3-SAT

\[(x + y + z) \Rightarrow \]

\[\forall x\]

gadget is 3-colorable $\iff x + y + z$ is true
Ex: \((x+y+z)(\overline{x} + \overline{y} + z)(\overline{x} + y + z)\)
Ex (cont.): a 3-coloring:

Solution $\Rightarrow x=$true, $y=$false, $z=$false
Thm: 3-COLORABILITY is NPC for graphs with max degree 4.

Pf: degree-reduction "gadget":

a) max degree 4
b) 3-colorable but not 2-colorable
c) all corners get same color

"Super"-gadgets:

Use these "fanout" components to reduce node degrees to 4 or less
Ex:

\[ G \text{ is 3-colorable } \iff G' \text{ is 3-colorable} \]
Q: is 3-COLORABILITY NPC for graphs with max degree 3?
Thm: 3-COLORABILITY is NPC for planar graphs.

Pf: planarity-preserving "gadget":

- a) planar and 3-colorable
- b) Opposite Corners get same color
- c) "independence" of pairs of OC's

Use gadget to avoid edge crossings:
Ex:

G:

G':

G is 3-colorable $\iff$ G' is 3-colorable