> The problems in this assignment are *tagged*:
>
> Core are essential problems that you *must solve* to pass the assignment.
>
> Challenge are *non-mandatory* problems that could be *skipped* for a passing grade.
>
> Bonus are *optional* problems for *memorable experience*. They won't be graded.

Problem 1: Language Zoo

Core

A *formal language* over alphabet Σ is any set $L \subseteq \Sigma^*$.

1. For each language below, list *three words in L* and *three words not in L*:
   (a) $L_1 = \{w \in \{0,1\}^* \mid w \text{ contains } \texttt{010} \text{ as a substring}\}$
   (b) $L_2 = \{a^n b^m \mid n \geq 1, m \geq 0\}$ over $\Sigma = \{a, b\}$
   (c) $L_3 = \{w \in \{a,b\}^* \mid |w| \text{ is prime}\}$
   (d) $L_4 = \{w\#w \mid w \in \{0,1\}^*\}$ over $\Sigma = \{0, 1, \#\}$
2. Let $L = \{\texttt{01}, \texttt{10}\}$ over $\Sigma = \{0, 1\}$. Explicitly describe or list each of the following languages:
   (a) $L^2 = L \cdot L$
   (b) L^3
   (c) L^* and L^+
   (d) $\overline{L} = \Sigma^* \setminus L$
   (e) $L \cdot \{0, 1\}$
3. Classify each language by the *lowest* level of the Chomsky hierarchy it belongs to.
   (a) $\{a^n \mid n \geq 0\}$
   (b) $\{a^n b^n \mid n \geq 0\}$
   (c) $\{a^n b^n c^n \mid n \geq 0\}$
   (d) $\{\langle M, w\rangle \mid \text{TM } M \text{ halts on input } w\}$

Problem 2: Regular Expressions

Core

Regex quick reference:

- `ab` = *concatenation.*
- `a|b` = *alternation.*
- `(...)` = *grouping.*
- `a*` = *zero or more.*
- `a+` = *one or more.*
- `a?` = *optional (0 or 1).*
- `a{n}` = *exactly n times.*
- `[abc]` = *any of the listed characters.*
- `[^abc]` = *any character* except *those listed.*

1. For each regular expression over $\Sigma = \{a, b, c, d\}$, describe the formal language in plain natural language, and count the *exact number of words of length ≤ 5* it accepts.
   (a) `ab*`
   (b) `a+b?c`
   (c) `[abc]*[cd]*`
   (d) `[^a]+[ab]`
   (e) `d(a|bc)*`
   (f) `((a|ab)[cd])+`
2. Write a regular expression for each of the following languages over $\Sigma = \{0, 1\}$:
   (a) All strings that *start and end with the same symbol*.
   (b) All strings *with no two consecutive* `1`s.
   (c) All strings of *even length*.
   (d) All strings whose *length is divisible by 3*.
   (e) All strings *containing* `110` as a substring.
   (f) All strings where *every* `1` is *immediately followed by* a `0`.

3. **Regex Crosswords.**[1] Fill each cell with a *single ASCII character* (uppercase letter, digit, punctuation, or space). Every row and column must match the corresponding regex pattern (read from left to right and top to bottom, respectively).

> - `[^ABC]` matches any character *except* those listed.
> - "`.`" matches *any* single character (unless escaped!).
> - `\1` is a PCRE *backreference* to the first capturing group — not a classical regex feature, but treat it concretely: if group 1 matched `X`, then `\1` must also match `X`.

The Beatles

	`[^SPEAK]+`	`EP|IP|EF`
`HE|LL|O+`		
`[PLEASE]+`		

Royal Dinner

	`(FI|A)+`	`(YE|OT)K`	`(.)|IF]+`	`[NODE]+`	`(FY|F|RG)+`
`(Y|F)(.)\2[DAF]\1`					
`(U|O|I)*T[FRO]+`					
`[KANE]*[GIN]*`					

Technology

	`[^NRU](NO|ON)`	`(D|FU|UF)+`	`(FO|A|R)*`	`(N|A)*`
`[RUNT]*`				
`O.*[HAT]`				
`(.)*DO\1`				

GMC Vandura

	`[^MCI]+`	`.A`	`(TM|BF)`	
`(CAT|A-T)+`				`[^KI\sP]+`
`[MA\-\sE]+`				`(M|APS|EA)*`
	`[AI][E\s]`	`[A\-Z]+`	`[\sT\-M]+`	

Big Mac

	`(.)\1(.)\2`	`[C\sOU]+`	`[^PU\sH]+`	
`.[LUH]+`				`.*L+`
`(P|K)[^U]+`				`[PUF\s]*`
`.*C+[TIF]`				`[TIC]*`
`(NO|ONE|ION)*`				`[NOI\sE]+`
	`[PIF]+`	`.*[OWE]*`	`(TN|LF|TF)*`	

Time Walker

	`.*E.*`	`[^PII(IT|ME)`	`(EM|FE)(IT|IP)`	`(TS|PE|KE)*`
`(EP|ST)*`				
`T[A-Z]*`				
`.M.T`				
`.*P.[S-X]+`				

[1] Puzzles adapted from https://regexcrossword.com. Visit for hundreds of crosswords at all difficulty levels!

Problem 3: DFA Construction

Core

1. For each language over $\Sigma = \{0, 1\}$, design a DFA (draw a state diagram *and* write the transition table). Indicate the start state and all accepting states.

   (a) $L_A = \{w \mid w \text{ ends with } \texttt{101}\}$

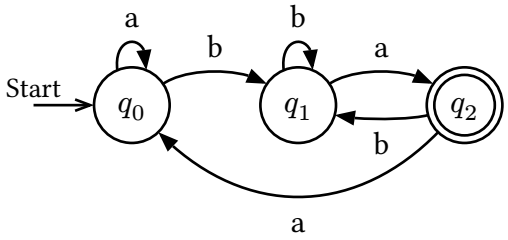
   (b) $L_B = \{w \mid w \text{ has an odd number of } \texttt{1}s \text{ and an even number of } \texttt{0}s\}$
   (Hint: track parities of both counts — 4 states suffice.)

   (c) $L_C = \{w \mid w \text{ interpreted as a binary number is divisible by } 3\}$
   (Hint: track the remainder modulo 3 — 3 states suffice.)

   (d) $L_D = \{w \mid w \text{ contains neither } \texttt{00} \text{ nor } \texttt{11} \text{ as a substring}\}$
   (Hint: the "last symbol seen" determines the state.)

2. The DFA $\mathcal{A}$ below has states $Q = \{q_0, q_1, q_2\}$ over $\Sigma = \{\mathrm{a}, \mathrm{b}\}$.

	a	b
q0	q0	q1
q1	q2	q1
q2	q0	q1



   (a) What language does $\mathcal{A}$ recognize? Give a concise description *and* a regular expression.

   (b) Trace the computation on each input: ε, `ba`, `aba`, `bbba`. Does $\mathcal{A}$ accept or reject each?

   (c) Is $\mathcal{L}(\mathcal{A})$ finite or infinite? How can you determine this from the DFA alone?

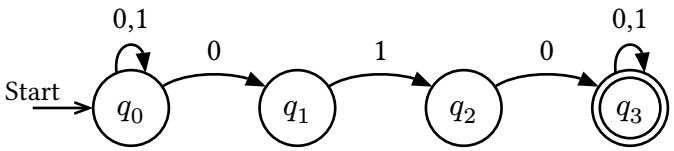
3. **Prove** that the DFA you constructed for L_C is correct. Specifically, show by induction on $|w|$ that after reading prefix w, the automaton is in state r_i if and only if $w \equiv i \pmod 3$ (treating w as a binary number). *(Remark: the empty string ε represents the value 0, so the machine starts in r_0.)*

Problem 4: Powerset Construction

Core

Consider the NFA $\mathcal{N}$ over $\Sigma = \{0, 1\}$ with states $Q = \{q_0, q_1, q_2, q_3\}$, start state q_0, accepting state q_3, and the transition table below.

State	$\delta(-, 0)$	$\delta(-, 1)$	Accepting?
q_0	$\{q_0, q_1\}$	$\{q_0\}$	
q_1	$\emptyset$	$\{q_2\}$	
q_2	$\{q_3\}$	$\emptyset$	
q_3	$\{q_3\}$	$\{q_3\}$	✓



(a) What language does $\mathcal{N}$ recognize? Argue informally (what pattern must w satisfy to be accepted?) and give a regular expression.

(b) Apply the *Rabin–Scott powerset construction* to convert $\mathcal{N}$ to a DFA $\mathcal{D}$. Enumerate all *reachable* DFA states (as subsets of Q) and compute the transition table. Mark which states are accepting.

(c) Draw $\mathcal{D}$. How many reachable states does it have? How does this compare to the $2^4 = 16$ states that the powerset construction could in principle produce?

Problem 5: Kleene's Theorem in Action

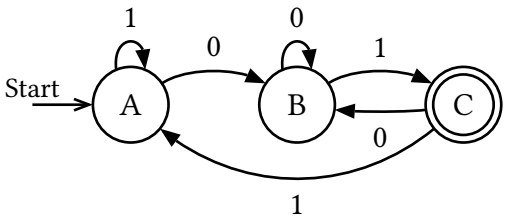
Core

> **Kleene's Theorem:** A language is *regular* (described by a regex) if and only if it is recognized by a *finite automaton* (DFA/NFA/ε-NFA). The proof is constructive in both directions:
> - *Regex $\to$ ε-NFA*: Thompson's construction (induction on the grammar).
> - *DFA $\to$ Regex*: Kleene's algorithm (dynamic programming, analogous to Floyd–Warshall).

1. **Thompson's construction.** Consider the regular expression `(0|01)*1` over $\Sigma = \{0, 1\}$.
   (a) Construct an ε-NFA for this language using Thompson's construction.
   (b) Compute the ε-closure of each state.
   (c) Eliminate the ε-transitions to obtain a plain NFA. Test your NFA on the inputs `1`, `01`, `001`, `011`.

2. **Kleene's algorithm.** Consider the DFA $\mathcal{D}$ with states $\{A, B, C\}$ over $\Sigma = \{0, 1\}$:

	0	1
A	B	A
B	B	C
C	B	A



   (a) What language does $\mathcal{D}$ accept? Give an informal description.
   (b) Define the base cases R^0_{ij} for all pairs of states (A, B, C) numbered $1, 2, 3$.
   (c) Compute R^1_{ij}, R^2_{ij}, and finally R^3_{ij} for the pair $(i, j) = (1, 3)$, *i.e.* extract the regex for words that take A to C. Show all intermediate steps.

3. **Conversion cycle.** Let $L = \{w \in \{0,1\}^* \mid |w| \equiv 0 \pmod 2\}$ (strings of *even length*).
   (a) Write a regular expression R for L.
   (b) Convert R to an ε-NFA using Thompson's construction.
   (c) Convert the ε-NFA to a DFA using the powerset construction.
   (d) Convert the DFA back to a regex using Kleene's algorithm.
   (e) Is the resulting regex syntactically identical to R? Do the two regexes describe the same language? Explain briefly.

Problem 6: Pumping Lemma

Core

> ***Full Pumping Lemma for Regular Languages.*** *Let L be a regular language. Then there exists $n \in \mathbb{N}$, $n > 0$, such that for every $w \in L$ with $|w| \geq n$, there exist strings x, y, z satisfying:*
> - $w = xyz$,
> - $y \neq \varepsilon$,
> - $|xy| \leq n$,
> - $xy^iz \in L$ *for every* $i \in \mathbb{N}$.

1. For each language, determine whether it is *regular* or not. For regular languages, exhibit a DFA or regular expression. For non-regular languages, prove non-regularity using the pumping lemma.
   - Fix an arbitrary pumping length $n \geq 1$.
   - Exhibit a specific $w \in L$ with $|w| \geq n$, expressed as a function of n.
   - Show that every valid split $w = xyz$ with $y \neq \varepsilon$ and $|xy| \leq n$ leads to a contradiction: produce an explicit $i \geq 0$ such that $xy^iz \notin L$.

(a) $\{0^n 1^{2n} \mid n \geq 0\}$
(b) $\{w \in \{0,1\}^* \mid w = w^R\}$
(c) $\{ww \mid w \in \{0,1\}^*\}$
(d) $\left\{1^{n^2} \mid n \geq 0\right\}$

2. **Weak vs. full pumping lemma.** Consider $L = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$ (strings with *equal* numbers of 0s and 1s).

   (a) Show that L *passes the weak pumping lemma*: for any n and any $w = 0^n 1^n$, the adversary can always find a split $w = xyz$ (without the constraint $|xy| \leq n$) such that all pumpings stay in L. *(Hint: let y span the boundary between 0s and 1s.)*
   (b) Now apply the *full* pumping lemma (with the constraint $|xy| \leq n$). Show that the adversary is forced to choose y entirely from the leading 0s, and derive a contradiction.
   (c) Conclude that L is not regular. Explain in one sentence why the full version catches what the weak version misses.

Problem 7: Myhill–Nerode Theorem

Core

Myhill–Nerode Theorem. *A language $L \subseteq \Sigma^*$ is regular if and only if the relation $\equiv_L$ defined by*

$$x \equiv_L y \iff \forall z \in \Sigma^*.\, (xz \in L \iff yz \in L)$$

has finite index (finitely many equivalence classes). Moreover, the number of classes equals the number of states in the minimal DFA for L.

1. **Non-regularity via distinguishable strings.** Recall: strings $x_1, x_2, \ldots$ are *pairwise L-distinguishable* if for every $i \neq j$ there exists z such that exactly one of $x_i z$, $x_j z$ is in L. An infinite set of pairwise distinguishable strings implies L is not regular.

   For each language, exhibit an *infinite* set of pairwise L-distinguishable strings:

   (a) $L = \{0^n 1^n \mid n \geq 0\}$. *(Use the strings $0^0, 0^1, 0^2, \ldots$ and find the right z for each pair.)*
   (b) $L = \{ww \mid w \in \{0,1\}^*\}$. *(Use the strings $0^0 1, 0^1 1, 0^2 1, \ldots$)*
   (c) $L = \{w \in \{0,1\}^* \mid w \text{ is a palindrome}\}$. *(Find an appropriate infinite family.)*

2. **Minimal DFA via equivalence classes.** Identify the Myhill–Nerode equivalence classes of each language over $\Sigma = \{0,1\}$, and draw the corresponding minimal DFA. Verify that the number of classes equals the number of DFA states.

   (a) $L = \{w \mid w \text{ ends with } \texttt{01}\}$. *(Claim: exactly 3 classes.)*
   (b) $L = \{w \mid w \text{ has an odd number of } \texttt{1}s\}$. *(Claim: exactly 2 classes.)*
   (c) $L = \{w \mid w \text{ has length divisible by } 3\}$. *(Claim: exactly 3 classes.)*

3. **Conceptual question.** Explain in your own words:

   (a) Why does the number of Myhill–Nerode classes equal the number of states in the *minimal* DFA? What is the role of the transition function $\delta([x], a) = [xa]$?
   (b) What goes wrong (specifically, what does the DFA fail to do) if we try to *merge* two states that belong to different equivalence classes?

Problem 8: Closure Properties

Core

1. **True / False / Depends.** For each statement, decide whether it is *always true*, *always false*, or *depends on the choice of languages*. For each answer, give a brief *proof* (if always true/false) or a *counterexample* (if depends).

   (a) If L is regular, then $L^R = \{w^R \mid w \in L\}$ is regular.
   (b) If L_1 and L_2 are both non-regular, then $L_1 \cup L_2$ is non-regular.
   (c) If $L_1 \cup L_2$ is regular and L_1 is regular, then L_2 is regular.
   (d) If $L_1 \cdot L_2$ is regular, then both L_1 and L_2 are regular.
   (e) If L is regular, then $\{w \mid ww \in L\}$ is regular.

2. **Product construction.** Let $L_1 = \{w \in \{0,1\}^* \mid w \text{ contains } \texttt{00}\}$ and $L_2 = \{w \in \{0,1\}^* \mid w \text{ contains } \texttt{11}\}$. Both are regular.

   (a) Construct a DFA $\mathcal{A}_1$ for L_1 (3 states) and a DFA $\mathcal{A}_2$ for L_2 (3 states). Draw both automata.
   (b) Construct the *product automaton* $\mathcal{A}_1 \times \mathcal{A}_2$ for $L_1 \cap L_2$. The product state space is $Q_1 \times Q_2$; mark the initial and accepting states.
   (c) Which states of the product automaton are accepting for $L_1 \cup L_2$?
   (d) Draw the resulting product DFA (without unreachable states).

Problem 9: Context-Free Grammars

Core

A *context-free grammar* $G = (V, \Sigma, R, S)$ has *variables* V, *terminals* Σ, *productions* R, and a *start variable* S. The language $\mathcal{L}(G)$ is the set of all terminal strings derivable from S.

1. **Reading grammar notation.** Each grammar below is written in *EBNF* (Extended Backus–Naur Form), a compact notation widely used in language specifications.

   > ***EBNF quick reference:***
   >
   > - `<name> ::= ...` *defines a non-terminal.*
   > - `"x"` *is a literal character.*
   > - `AB` *= concatenation.*
   > - `A | B` *= alternation.*
   > - `( ... )` *= grouping.*
   > - `A?` *= optional (0 or 1).*
   > - `A+` *= one or more.*
   > - `A*` *= zero or more.*

   For each EBNF grammar, (i) describe the language in *plain natural language*, (ii) give a *regular expression* defining the same language (if possible), and (iii) argue whether the language is *regular*.

   (a)
   ```
   <integer> ::= <sign>? <nonzero> <digit>*
   <sign>    ::= "+" | "-"
   <nonzero> ::= "1" | "2" | ... | "9"
   <digit>   ::= "0" | "1" | ... | "9"
   ```

   (Examples: `42`, `-7`, `+100` are in; `007`, `0`, `+-5` are not.)

   (b)
   ```
   <id>     ::= <letter> <rest>*
   <rest>   ::= <letter> | <digit> | "_"
   <letter> ::= "a" | ... | "z" | "A" | ... | "Z"
   <digit>  ::= "0" | ... | "9"
   ```

   (This is the simplified token grammar for identifiers in most programming languages.)

2. For each CFG, describe $\mathcal{L}(G)$ and show few example derivations to confirm your answer. Also exhibit *one string not in the language* and briefly argue why no derivation of it exists.

   (a) G_1: $S \to \mathtt{0}S\mathtt{1} \mid \varepsilon$
   (b) G_2: $S \to aSb \mid bSa \mid SS \mid \varepsilon$
   (c) G_3: $S \to AB, \quad A \to aA \mid a, \quad B \to bB \mid b$
   (d) G_4: $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

3. Design a CFG for each of the following languages:

   (a) $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$

   > **Hint:** Design G' for $\{a^n b^n c^k\}$ and G'' for $\{a^i b^n c^n\}$, then combine with $S \to S' \mid S''$.

   (b) $\{w \in \{a, b\}^* \mid \#_a(w) = 2 \cdot \#_b(w)\}$

   > **Hint:** Each b must be paired with two a's. Think of productions that introduce two a's alongside each b: for example, $S \to aaSbS \mid aSaSb \mid SaaSb \mid \ldots$

   (c) The set of all properly matched bracket strings over $\Sigma = \{(,)\}$.
   For example, $()$, $(())$, $()()$ are properly matched; $)($ and $(()$ are not.

4. **Chomsky Normal Form.** A CFG is in *Chomsky Normal Form* (CNF) if every production has the form $A \to BC$ (two variables) or $A \to a$ (one terminal).

   (a) Convert the grammar $S \to \mathtt{0}S\mathtt{1} \mid \varepsilon$ into CNF. Show each step of the conversion: add new start symbol, eliminate ε-productions, then binarize rules.
   (b) Why is CNF useful? Name one algorithm that requires the grammar to be in CNF.

Problem 10: Turing Machines

Core

> ***Turing Machine.*** *A TM $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ reads and writes on an infinite tape. At each step, based on the current state and tape symbol, it: transitions to a new state, writes a symbol, and moves the head left (L) or right (R). It accepts if it reaches q_{acc}, rejects if it reaches q_{rej}, and may loop forever. A language is decidable if some TM always halts; recognizable (RE) if some TM accepts every word in the language (but may loop on other inputs).*

1. **Tracing.** Consider a Turing machine $\mathcal{M}$ for $\{0^n 1^n \mid n \geq 0\}$ that works as follows: repeatedly find the leftmost uncrossed `0`, mark it, move right to find the leftmost uncrossed `1`, mark it, and return to the left; if it ever sees a `1` before an unmatched `0`, it rejects; if all symbols are matched, it accepts.

   For each input, show the computation as a sequence of *configurations* (using the notation $\langle \alpha; q; \beta \rangle$ where q is the current state, α is the tape to the left of the head, β is the tape at and to the right of the head). State whether $\mathcal{M}$ *accepts*, *rejects*, or *loops*.

   (a) Input 0011
   (b) Input 001
   (c) Input ε (empty input)
   (d) Input 10

2. **Design.** Describe — in clear informal pseudocode or numbered steps, *not* a full state table — a Turing machine for each language. For each, argue briefly why the machine *always halts*.

   (a) $L = \{0^{2^n} \mid n \geq 0\}$, strings whose length is a *power of 2*.

   > **Hint:** Repeatedly halve the tape: cross off every other 0. Accept if exactly 1 unmarked 0 remains; reject if an odd number > 1 remains.

   (b) $L = \{w\#w \mid w \in \{0,1\}^*\}$, the *equality language*.

   > **Hint:** Match the k-th symbol on the left of $\#$ with the k-th symbol on the right. Use crossing-off and left-right zigzagging.

3. **Decidability classification.** For each language, classify it as: **(R)** decidable, **(RE \ R)** recognizable but undecidable, or **(co-RE)** / **(neither)** as applicable. Justify briefly (cite Rice's theorem, known reductions, or direct arguments).

   (a) $\{\langle M \rangle \mid M \text{ is a TM with exactly 5 states}\}$
   (b) $\{\langle M, w \rangle \mid M \text{ accepts } w \text{ within 1000 steps}\}$
   (c) $\{\langle M \rangle \mid M \text{ accepts the empty string } \varepsilon\}$
   (d) $\{\langle M \rangle \mid \mathcal{L}(M) \text{ is infinite}\}$
   (e) $\{\langle M \rangle \mid \mathcal{L}(M) = \Sigma^*\}$ (the TM accepts *everything*)

Problem 11: Exact Counting vs Modular Counting

Challenge

1. **Closure instead of pumping.** Consider $L = \{0^i 1^j \mid i \neq j\}$.

   (a) Prove that L is *not regular* using *closure properties*, not the pumping lemma.
   (b) Explain briefly why closure is the natural tool here. What structure of the language does the closure argument expose?

2. **Regularity from similarity.** The language $L_{a=b} = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$, where $\#_a(w)$ and $\#_b(w)$ are the number of a's and b's in w respectively, is not regular (it requires unbounded counting). Now consider $L' = \{w \in \{a,b\}^* \mid \#_a(w) \equiv \#_b(w) \pmod 2\}$.

   (a) Is L' regular? Prove your answer (build DFA/regex if yes, apply pumping lemma if no).
   (b) Explain intuitively why modular counting is "easier" for a DFA than exact counting.

3. **Minimal automata for modular counting.** Fix an integer $m \geq 2$ and define the language

$$L_m = \{w \in \{a,b\}^* \mid \#_a(w) \equiv \#_b(w) \pmod m\}$$

   (a) Construct a DFA $\mathcal{A}_m$ for L_m with m states. Prove that your DFA recognizes exactly L_m.
   (b) Describe the meaning of each state and the transition rule on input a and b. State a precise invariant of the form "after reading w, the automaton is in state q_k iff ..." and prove it.
   (c) Prove that no DFA with fewer than m states can recognize L_m.

   > **Hint:** Show that the strings $\varepsilon, a, a^2, \ldots, a^{m-1}$ are pairwise distinguishable w.r.t. L_m.

Problem 12: Non-Regular Languages and the Closure Game

Challenge

1. **Closure traps.** For each claim, decide: is it **always** true, or can it **fail**? If it always holds, prove it. If it can fail, give an explicit counterexample (a concrete non-regular L, L_1, or L_2).

   (a) L non-regular $\Rightarrow$ $L^2 = L \cdot L$ non-regular.
   (b) L non-regular $\Rightarrow$ L^* non-regular.
   (c) L_1, L_2 non-regular $\Rightarrow$ $L_1 \cap L_2$ non-regular.
   (d) L_1 regular, L_2 non-regular $\Rightarrow$ $L_1 \setminus L_2$ non-regular.

2. **Mutual constraints.** Suppose we are given that:
   - $L_1 \cup L_2 = \Sigma^*$ (*regular*),
   - $L_1 \cap L_2 = \{a^n b^n \mid n \geq 0\}$ (*non-regular*).

   What can you conclude about L_1 and L_2? Can *both* be regular? Can *exactly one* be regular? Can *both* be non-regular? Prove each case that is possible, and disprove each case that is not.

3. **Half-language.** Define $\text{HALF}(L) = \{x \mid \exists y \in \Sigma^*.\ |y| = |x| \text{ and } xy \in L\}$.

   Prove: if L is *regular*, then $\text{HALF}(L)$ is also *regular*.

   > **Hint:** Start from a DFA for L. Think about what information about a prefix x is enough to decide whether there exists a suffix y with $|y| = |x|$ and $xy \in L$.

4. **Homomorphisms.** A *string homomorphism* $h : \Sigma_1^* \to \Sigma_2^*$ replaces each symbol $a \in \Sigma_1$ by a fixed string $h(a) \in \Sigma_2^*$. Define $h : \{a, b\}^* \to \{0, 1\}^*$ by $h(a) = \texttt{01}$ and $h(b) = \texttt{10}$.

   (a) Let $L = \{w \in \{0, 1\}^* \mid w \text{ starts with } \texttt{01}\}$. Describe the inverse image $h^{-1}(L) = \{x \in \{a, b\}^* \mid h(x) \in L\}$, and determine whether it is regular.

   (b) Let $P = \{ww^R \mid w \in \{a, b\}^*\}$ be the language of even-length palindromes over $\{a, b\}$. Is the image $h(P)$ regular? Justify your answer carefully.

   (c) State and prove the theorem: if $L \subseteq \Sigma_2^*$ is regular, then $h^{-1}(L)$ is regular. Your proof should explain how to build a finite automaton for $h^{-1}(L)$ from a finite automaton for L.

Problem 13: Undecidability from First Principles

Challenge

> ***Rice's Theorem.*** *Any non-trivial property of the language recognized by a Turing machine is* undecidable.
>
> *Let P be a property of languages such that:*
> - *P is* semantic*: it refers only to the language, not to the specific TM that recognizes it;*
> - *P is* non-trivial*: there exist TMs M_1, M_2 with $\mathcal{L}(M_1) \in P$ and $\mathcal{L}(M_2) \notin P$.*
>
> *Then $\{\langle M \rangle \mid \mathcal{L}(M) \in P\}$ is* undecidable.

1. **Rice's theorem.** For each property P, determine whether $\{\langle M \rangle \mid \mathcal{L}(M) \in P\}$ is decidable. If undecidable, cite Rice's theorem and verify that P is non-trivial. If decidable, explain why.

   (a) $\mathcal{L}(M)$ contains at least one string.
   (b) $\mathcal{L}(M)$ is empty.
   (c) $\mathcal{L}(M)$ is a regular language.
   (d) M has fewer than 100 states. *(Beware!)*
   (e) M halts on all inputs.
   (f) $\mathcal{L}(M)$ contains at least one palindrome.

2. **Diagonalization.** A student proposes the following "algorithm" H for the halting problem:

> *Simulate M on w step by step. After each step, check if M has halted. If M halts, output* YES*. If M has not halted after k steps, output* NO*.*

(a) What is wrong with this algorithm? For which inputs does it give the wrong answer?
(b) Even if we let $k \to \infty$ ("simulate forever"), why doesn't this solve the halting problem?
(c) The actual proof of the halting problem undecidability constructs a TM D that, on input $\langle M \rangle$, loops iff M halts on $\langle M \rangle$, and halts iff M does not halt on $\langle M \rangle$. What happens when we run D on $\langle D \rangle$? Describe the contradiction that arises.

3. **Reduction.** Recall:

$$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ accepts } w\},$$
$$\text{HALT} = \{\langle M, w \rangle \mid M \text{ halts on } w\}.$$

Using the undecidability of HALT, prove that A_{TM} is also undecidable. Construct a many-one reduction $\text{HALT} \leq_m A_{\text{TM}}$: a computable function f such that for every input $\langle M, w \rangle$:

$$\langle M, w \rangle \in \text{HALT} \iff f(\langle M, w \rangle) \in A_{\text{TM}}.$$

Optional Bonus Problems

> The following problems are *"optional"*. They require programming and/or deeper theory. They will *not* count toward your grade but may earn bonus points and — more importantly — genuine understanding.

Problem A: Regex Engine

Bonus

Implement a simplified regex-to-DFA compiler in a language of your choice (Python recommended).

1. **Parsing.** Implement a *recursive-descent parser* for the grammar below (no lookahead, no backreferences):

```
<regex>  ::= <term> ( '|' <term> )*
<term>   ::= <factor>+
<factor> ::= <atom> ( '*' | '+' | '?' )?
<atom>   ::= char | '(' <regex> ')'
```

   Represent the result as an AST (abstract syntax tree).

2. **Thompson's NFA.** Implement Thompson's construction to convert the AST to an ε-NFA. Represent states as integers and transitions as adjacency lists.

3. **Powerset (subset) construction.** Convert the ε-NFA to a DFA. Implement ε-closure and transition computation on sets of states.

4. **Testing.** Test your engine on the following and verify against Python's built-in `re` module:
   - regex `(a|b)*abb` on all strings over $\{a, b\}$ of length ≤ 8
   - regex `0*(10*10*)*` on all binary strings of length ≤ 10

5. **Count.** For regex `ab*c|d` over $\Sigma = \{a, b, c, d\}$, use your engine to enumerate and count all accepted words of length ≤ 6.

Problem B: DFA Minimization

Bonus

Implement the *table-filling* algorithm for DFA minimization.

1. **Table-filling algorithm.** Given a complete DFA $(Q, \Sigma, \delta, q_0, F)$:
   (a) Mark all pairs (p, q) where exactly one of p, q is accepting.
   (b) Repeat: for each unmarked pair (p, q), if there exists $a \in \Sigma$ such that $(\delta(p, a), \delta(q, a))$ is already marked, mark (p, q) as well. Stop when no new pair is marked in a full pass.
   (c) The *unmarked* pairs are pairs of equivalent (indistinguishable) states. Merge them into single states.

2. Test your implementation on at least three nontrivial DFAs. At least one of them should come from applying the powerset construction to an NFA of your choice. Report the number of states before and after minimization in each case.

3. **Correctness.** Prove that the table-filling algorithm marks (p, q) if and only if p and q are distinguishable (belong to different Myhill–Nerode equivalence classes).

4. **Benchmark.** Implement *Hopcroft's algorithm* (time $O(n \log n)$).
   - Randomly generate DFAs over $\Sigma = \{0, 1\}$ with $n \in \{10, 50, 100, 500, 1000\}$ states.
   - Compare runtimes of table-filling ($O(n^2|\Sigma|)$) vs. Hopcroft.
   - Include a plot of "runtime vs. n".

Problem C: CYK Membership Testing

Bonus

Implement the *Cocke–Younger–Kasami* (CYK) algorithm for context-free grammar membership.

1. **CNF conversion.** Implement a general CFG to Chomsky Normal Form converter.
   - Add a new start symbol S' with $S' \to S$.
   - Eliminate ε-productions (nullable variables).
   - Eliminate *unit productions* ($A \to B$).
   - *Binarize* all remaining productions of length ≥ 3.

   Verify your converter on the grammar $S \to \texttt{0}S\texttt{1} \mid \varepsilon$.

2. **CYK algorithm.** Implement CYK: given a CFG G in CNF and a string $w = a_1 \ldots a_n$, compute the $n \times n$ table $T[i, j] = \{A \in V \mid A \Rightarrow^* a_i \ldots a_j\}$. Time complexity: $O(n^3|G|)$.

3. **Parse tree reconstruction.** Extend your CYK implementation to *reconstruct a parse tree* (not just answer yes/no).

4. **Testing.** Test on:

   (a) The grammar $S \to \texttt{0}S\texttt{1} \mid \varepsilon$: verify all 0^n1^n for $n = 1, \ldots, 8$ are accepted; verify 0^n1^m for $n \neq m$ are rejected.

   (b) A small arithmetic expression grammar: $E \to E + T \mid T, T \to T * F \mid F, F \to (E) \mid \text{num}$. Test on `num + num * num` and `(num + num) * num`.

5. **Benchmark.** Measure parsing time vs. string length n for $n = 10, 20, 50, 100, 200$. Verify the $O(n^3)$ scaling empirically (plot time vs. n^3; the line should be approximately linear).