Regular Languages

Definition, Examples, Closure, and Limits

Sara Sorahi

Introduction to the Theory of Computation

From Machines to Language Classes

Until now, we studied:

- DFA and NFA
- how they process input
- how nondeterminism is removed

Now we shift focus: **What kinds of languages can be recognized with finite memory?**

What Is a Language?

A **language** is a set of strings over an alphabet Σ:

$$L \subseteq \Sigma^*$$

Examples:

- $\{a, ab, abb\}$
- $\{w \mid w \text{ ends with } 01\}$
- $\emptyset,\ \{\varepsilon\},\ \Sigma^*$

Definition of Regular Language

A language L is **regular** if there exists a DFA M such that:

$$L = L(M)$$

Because every NFA can be converted into a DFA:

- DFA and NFA define the same class of languages

The Meaning of Finite Memory

A finite automaton:

- has finitely many states
- has no stack or counters
- summarizes the past using its current state

A language is regular if all relevant information about the past can be stored in a **finite number of states**.

Worked Example: Ends with 01

$$L = \{w \in \{0,1\}^* \mid w \text{ ends with } 01\}$$

To decide membership, we only need to remember:

- the last symbol
- the second-to-last symbol

Accepted vs Rejected Strings

Accepted:

- 01
- 101

Rejected:

- 100
- 0

The decision depends on a fixed-size suffix.

Counting Modulo Patterns

Finite automata can count modulo a fixed number.

Examples:

- even number of 1s
- length divisible by 3
- number of as $\equiv 2 \pmod 4$

Counting modulo k requires only k states.

Regular Expressions

Regular expressions describe regular languages:

- without machines
- using symbolic patterns

Regular expressions and finite automata describe the **same class of languages**.

Basic Symbols

- a : $\{a\}$
- ε : $\{\varepsilon\}$
- $\emptyset$: empty language

ε is a string. $\emptyset$ is a set.

Reading a Regular Expression

$$(a|b)^*abb$$

Accepted: `abb`, `aabb`, `abababb`
Rejected: `abba`

Closure of Regular Languages

Regular languages are closed under:

- union
- intersection
- complement
- concatenation
- Kleene star

Closure means: applying these operations keeps us inside the class.

Closure Under Union

Idea:

- simulate two DFAs in parallel
- accept if either accepts

This is called the **product construction**.

Closure Under Complement

Given a DFA:

- keep transitions the same
- swap accepting and non-accepting states

This works because DFAs have exactly one computation path.

Why a Limitation Theorem?

Some languages require unbounded memory.

Examples:

- $\{a^n b^n\}$
- palindromes
- equal number of 0s and 1s

Pumping Lemma (Statement)

If L is regular, then there exists p such that every $s \in L$ with $|s| \geq p$ can be written as $s = xyz$ satisfying:

1. $|y| \geq 1$
2. $|xy| \leq p$
3. $xy^i z \in L$ for all $i \geq 0$

Non-Regular Example: $\{a^n b^n\}$

Choose:

$$s = a^p b^p$$

Pumping changes the number of a's but not b's.
Contradiction.

Closure Under Intersection

Theorem. If L_1 and L_2 are regular languages, then

$$L_1 \cap L_2$$

is also regular.

Proof idea: Product construction

- Let M_1 and M_2 be DFAs for L_1 and L_2
- Construct a new DFA whose states are pairs (q_1, q_2)
- The machine simulates both DFAs in parallel
- Accept *only if both components are accepting*

Why the Pumping Lemma Is True

Idea behind the proof

- Suppose L is regular
- Then there exists a DFA M with p states
- Consider any string $s \in L$ with $|s| \geq p$
- While reading the first p symbols, M visits $p + 1$ states

By the pigeonhole principle, at least one state must repeat.

Pumping Lemma: Loop Argument

- Repeating a state creates a **loop** in the DFA
- The input segment that causes this loop is called y
- Traversing the loop multiple times does not change acceptance

This is why the string can be written as:

$$s = xyz$$

and why y can be repeated or removed.

Example: A Regular Language (Even Number of 1s)

Language:

$$L_{\text{even-1}} = \{\, w \in \{0,1\}^* \mid w \text{ has an even number of 1s} \,\}$$

Claim: $L_{\text{even-1}}$ is regular.

Reason (finite memory idea): To decide whether the number of 1s is even, we only need to remember *one bit of information*: parity (even/odd).

So a DFA with two states is enough:

- state E: seen an even number of 1s so far (start, accepting)
- state O: seen an odd number of 1s so far

Proof via DFA (Parity Automaton)

Transitions:

- On input 0: stay in the same state (parity unchanged)
- On input 1: toggle $E \leftrightarrow O$

Accepting state: E (even parity).

Why this proves regularity: We have explicitly described a DFA that accepts exactly $L_{\text{even-1}}$. Therefore $L_{\text{even-1}}$ is regular.

Example: A Non-Regular Language ($a^n b^n$)

Language:

$$L = \{ a^n b^n \mid n \geq 0 \}$$

(strings of a's followed by the same number of b's)

Claim: L is **not** regular.

Idea: A finite automaton has no unbounded memory, but L requires remembering how many a's were seen to match the same number of b's.

Proof (Pumping Lemma)

Assume, for contradiction, that L is regular. Then the pumping lemma gives a pumping length p.

Choose the string:

$$s = a^p b^p \in L$$

with $|s| \geq p$.

By the pumping lemma, s can be written as $s = xyz$ such that:

- $|y| \geq 1$
- $|xy| \leq p$
- $xy^i z \in L$ for all $i \geq 0$

Proof (Key Step: Where y Must Be)

Because $|xy| \leq p$, the substring xy lies entirely within the first p symbols of s.

But the first p symbols of s are all a's.

Therefore:

$$y = a^k \quad \text{for some } k \geq 1$$

So pumping changes the number of a's, but leaves the b's unchanged.

Proof (Pump Down and Contradict)

Pump down with $i = 0$:

$$xy^0z = xz$$

Since $y = a^k$ with $k \geq 1$, we remove at least one a:

$$xz = a^{p-k}b^p$$

This string has fewer a's than b's, so:

$$xz \notin L$$

But the pumping lemma requires $xz \in L$. Contradiction.

Therefore, L is not regular.