THEORY OF AUTOMATA & FORMAL LANGUAGES

Language Building — Solved Examples

From verbal conditions to precise mathematical languages

Understand → Construct → Test → Refine

A systematic construction method

Throughout this lecture: $\Sigma = \{a, b\}$; A denotes (a | b).

Five steps for every problem

1. Identify the alphabet.
2. Identify the required condition.
3. Describe allowed strings as a set.
4. Construct an expression if possible.
5. Test positive and negative cases.

Example: strings ending in b

Any prefix is permitted.

The last symbol must be b.

$L = \{xb : x \in \Sigma^*\}$

Regular expression: A*b

Accept b, ab, abb; reject ε, ba.

Always test the shortest strings, especially ε.

Example 1 — Length constraints

Construct the language of all strings of length exactly two.

Build the language

Each position permits a or b.

First position: a | b

Second position: a | b

$L = \{w \in \Sigma^* : |w| = 2\}$

Expression: AA

Enumerate and verify

a followed by a → aa

a followed by b → ab

b followed by a → ba

b followed by b → bb

$L = \{aa, ab, ba, bb\}$

Exactly two symbols: reject ε, a and aaa. At most two: ε | A | AA.

Example 2 — Starting and ending symbols

Construct all strings starting with a and ending with b.

Build the expression

First symbol must be a.

Middle may be any string $x \in \Sigma^*$.

Last symbol must be b.

$L = \{axb : x \in \Sigma^*\}$

Expression: aA*b

Test the boundary cases

$x = \varepsilon \rightarrow ab$

$x = a \rightarrow aab$

$x = ba \rightarrow abab$

Reject a: no final b.

Reject ba: wrong first symbol.

The minimum length is two because the required endpoints differ.

Example 3 — Required substring

Construct all strings containing the contiguous substring aba.

Build the expression

Before aba: any string x.

Required contiguous block: aba.

After aba: any string y.

$L = \{xabay : x,y \in \Sigma^*\}$

Expression: A*abaA*

Test membership

aba = ε · aba · ε

babab = b · aba · b

aaba = a · aba · ε

Reject ab: too short.

Reject aabb: no contiguous aba.

A substring is contiguous; intervening symbols are not permitted.

Example 4 — Exactly two b symbols

Construct all strings with exactly two occurrences of b.

Divide into blocks

Before first b: only a symbols.

Between the two b symbols: a only.

After second b: a only.

$L = \{a^i b a^j b a^k : i,j,k \geq 0\}$

Expression: a*ba*ba*

Test membership

i=j=k=0 → bb

i=1, j=2, k=0 → abaab

i=0, j=0, k=2 → bbaa

Reject ab: only one b.

Reject bbb: three b symbols.

Each a* can be empty; no additional b is permitted.

Example 5 — Even number of b symbols

Construct all strings with 0, 2, 4, ... occurrences of b.

Build a repeatable pair

Start with any number of a symbols.

One pair of b symbols: ba*ba*

Repeat the entire pair zero or more times.

$L = \{w \in \Sigma^* : |w|_\beta \text{ is even}\}$

Expression: a*(ba*ba*)*

Test membership

Zero pairs: ε, a, aaa

One pair: bb, abab, baab

Two pairs: bbbb, babbab

Reject b: one b.

Reject abbab: three b symbols.

Repeat pairs, not individual b symbols; zero is even.

Example 6 — Equal ordered blocks

Construct all strings with n a symbols followed by n b symbols.

Describe the language

Choose an integer $n \geq 0$.

Write n copies of a.

Write n copies of b.

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

$n = 0,1,2,3 \rightarrow \varepsilon$, ab, aabb, aaabbb

Test and distinguish

aabb: accepted (n = 2).

abab: rejected (wrong order).

aaabb: rejected (unequal counts).

a*b* allows independent counts.

(ab)* repeats alternating blocks.

This language is not regular; no classical regular expression describes it. Proof comes later.

Practice — Construct and verify

Use the five-step method; check the boundary cases.

Problems

1. Strings of length at least two.

2. Strings starting and ending in a.

3. Strings containing at least one b.

4. Strings with no b symbols.

5. Strings with equal ordered blocks, n ≥ 1.

Answers and reasoning

1. AAA* : two required symbols.

2. a | aA*a : include single a.

3. A*bA* : one required b.

4. a* : includes ε.

5. $\{a^n b^n : n \geq 1\}$; excludes ε.

Next: construct finite automata to recognize regular languages.