Regular Expressions

Describing regular languages with algebraic notation

Union, concatenation, Kleene star and language construction

Lecture focus: syntax • semantics • worked examples • practice

Learning outcomes

01 Define

Recognize the base regular expressions and their languages.

02 Interpret

Apply union, concatenation and Kleene star.

03 Construct

Write expressions for specified string patterns.

04 Verify

Test accepted and rejected strings and explain why.

From automata to expressions

Finite automata

A DFA or NFA recognizes a language by processing input symbols.

Regular expressions

An algebraic description of the same class of languages.

Kleene’s theorem

A language is regular exactly when some finite automaton recognizes it; equivalently, some regular expression denotes it.

Base regular expressions

Expression	Language denoted	Meaning
$\emptyset$	$\emptyset$	No strings
ε	$\{\varepsilon\}$	Empty string only
a $(a \in \Sigma)$	$\{a\}$	One symbol a

ε is a string of length zero; ∅ is a language with no elements.

Recursive definition

Union: R | S

$L(R \mid S) = L(R) \cup L(S)$. Choose either language.

Concatenation: RS

$L(RS) = \{uv : u \in L(R), v \in L(S)\}$.

Kleene star: R*

$L(R^*) = \bigcup_{n \geq 0} L(R)^n$, where $L(R)^0 = \{\varepsilon\}$.

Only expressions formed by the base cases and these rules are regular expressions.

Operator examples

Expression	Sample language / strings	Interpretation
a | b	{a, b}	Either symbol
ab	{ab}	a followed by b
a*	{ε, a, aa, aaa, …}	Zero or more a’s
(ab)*	{ε, ab, abab, …}	Repeat the block ab

Notation: | means union; some textbooks use + for union.

Precedence and grouping

Highest precedence: star

ab* means a(b*), not (ab)*.

Next: concatenation

ab | c means (ab) | c.

Lowest: union

Use parentheses to override precedence: a(b | c).

All binary strings

Problem

$\Sigma = \{0, 1\}$. Describe every finite binary string.

Construction

At each position choose 0 or 1; repeat zero or more times.

Regular expression

$(0 \mid 1)^*$ Accepts ε, 0, 1, 01, 110, …

Binary strings ending in 01

Step 1 — arbitrary prefix

The prefix may contain any number of binary symbols: (0 | 1)*.

Step 2 — required suffix

Append the literal block 01.

Result

(0 | 1)*01 Accepts 01, 101, 0001; rejects ε, 10, 010.

Trace the suffix expression

Input	Prefix + suffix	Result
01	ε · 01	Accept
1101	11 · 01	Accept
010	No final 01	Reject
ε	Too short	Reject

A valid decomposition must consume the entire input string.

Binary strings containing 01

Step 1 — allow any prefix

(0 | 1)* permits any binary string before the required block.

Step 2 — allow any suffix

(0 | 1)* permits any binary string after the required block.

Result

(0 | 1)*01(0 | 1)* Accepts 010, 1011; rejects 111000.

Binary strings with exactly two 1s

Step 1 — isolate the required symbols

Two literal 1s must occur in order.

Step 2 — fill the gaps

Only zeros may occur before, between or after those two 1s.

Result

0*10*10* Accepts 11, 101, 0010100; rejects 1, 111.

Binary strings with an even number of 1s

Step 1 — allow leading zeros

Use 0*; this also handles strings with zero 1s.

Step 2 — pair the ones

A block 10*10* contributes exactly two 1s.

Result

0*(10*10*)* Accepts ε, 00, 101, 1111; rejects 1, 1011.

Useful identities and cautions

Identity	Meaning
$R \mid \emptyset = R$	Union with empty language
$R\varepsilon = \varepsilon R = R$	Empty string is concatenation identity
$R\emptyset = \emptyset R = \emptyset$	Empty language annihilates concatenation
$\emptyset^* = \varepsilon^* = \varepsilon$	Both stars denote $\{\varepsilon\}$

Equality here means equality of the languages denoted.

Construct regular expressions

Q1

All binary strings beginning with 1.

Q2

All binary strings containing no 1s.

Q3

All binary strings with at least two 1s.

Q4

All binary strings of even length.

Solutions and recap

Q1: 1(0 | 1)*

Q2: 0*

Q3: (0 | 1)*1(0 | 1)*1(0 | 1)*

Q4: ((0 | 1)(0 | 1))*

Key takeaway

Build complex languages from union, concatenation and star. Next: regular expressions to finite automata.