
THEORY OF COMPUTATION

Chapter 3 REGULAR EXPRESSIONS

Complete Study Notes for Engineering Students

Theory · Solved Examples · Diagrams · 15 MCQs with Answer Key
· 10 Broad Questions

Topics Covered

- 3.1 Regular expression and regular set · Identities · Arden's theorem
- 3.2 RE and finite automata · Λ -moves · NFA to DFA · FA from RE · Equivalence of FAs and REs · Pumping lemma · Closure properties · Regular grammar and transition systems

Contents

3.1 Regular Expressions and Regular Sets	3
3.1.1 Basic Terminology (Quick Recap)	3
3.1.2 Definition of a Regular Expression	4
3.1.3 Regular Set	4
3.1.4 Writing Regular Expressions – Examples	4
3.1.5 Identities of Regular Expressions	5
3.1.6 Arden's Theorem	6
3.2 Regular Expressions and Finite Automata	8
3.2.1 Relation between Regular Expression and Finite Automata	8
3.2.2 Transition System Containing Λ -moves	8
3.2.3 Conversion of Non-deterministic System to Deterministic System	10
3.2.4 Construction of Finite Automata Equivalent to a Regular Expression	12
3.2.5 Equivalence of Two Finite Automata	14
3.2.6 Equivalence of Two Regular Expressions	15
3.2.7 Pumping Lemma for Regular Sets	16
3.2.8 Closure Properties of Regular Sets	17
3.2.9 Regular Grammar, DFA and Transition System	19
Chapter Summary – Conversion Chart	21
Multiple Choice Questions (15)	22
Answer Key	23
Broad Questions (10)	24

Regular Expressions

A **regular expression** (RE) is a compact algebraic way of describing a pattern of strings. Regular expressions are the bridge between the *language* view of computation (sets of strings) and the *machine* view (finite automata). In this chapter we learn what an RE is, how to manipulate it with algebraic identities, how to convert it to and from a finite automaton, how to test whether a language is regular, and how regular languages relate to grammars.

After studying this chapter you should be able to:

- write a regular expression for a given language and state the regular set it denotes;
- simplify and prove equality of regular expressions using identities;
- find the RE of a finite automaton using **Arden's theorem**;
- remove Λ -moves, convert an NFA to a DFA, and build an FA from an RE;
- check equivalence of two FAs / two REs;
- apply the **pumping lemma** to show that a language is not regular;
- use closure properties and convert between **regular grammars** and finite automata.

3.1 Regular Expressions and Regular Sets

3.1.1 Basic Terminology (Quick Recap)

Term	Meaning	Example
Alphabet (Σ)	A finite, non-empty set of symbols.	$\Sigma = \{0, 1\}$, $\Sigma = \{a, b\}$
String	A finite sequence of symbols from Σ .	0110, abba
Length $ w $	Number of symbols in the string w .	$ 0110 = 4$
Empty string (Λ)	The string with no symbols; $ \Lambda = 0$. It is the identity for concatenation.	$w\Lambda = \Lambda w = w$
Σ^*	Set of all strings over Σ including Λ .	$\{\Lambda, 0, 1, 00, 01, \dots\}$
Σ^+	Set of all non-empty strings: $\Sigma^* - \{\Lambda\}$.	$\{0, 1, 00, 01, \dots\}$
Language	Any subset of Σ^* .	$L = \{w \mid w \text{ ends with } 1\}$

Table 3.1: Basic terms

Languages are sets, so we can combine them. Three operations are the foundation of regular expressions:

Operation	Definition	Example ($L_1 = \{a\}$, $L_2 = \{b, c\}$)
Union	$L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$	$\{a, b, c\}$
Concatenation	$L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$	$\{ab, ac\}$
Kleene closure	$L^* = L^0 \cup L^1 \cup L^2 \cup \dots$ ($L^0 = \{\Lambda\}$)	$L_1^* = \{\Lambda, a, aa, aaa, \dots\}$
Positive closure	$L^+ = L^1 \cup L^2 \cup \dots = L L^*$	$L_1^+ = \{a, aa, aaa, \dots\}$

Table 3.2: Operations on languages

3.1.2 Definition of a Regular Expression

Let Σ be an alphabet. Regular expressions over Σ , and the sets they denote, are defined **recursively**:

1. $\varnothing$ (the empty set) is a regular expression.
2. Λ (the empty string) is a regular expression.
3. Every symbol $a \in \Sigma$ is a regular expression.
4. If R_1 and R_2 are regular expressions, then so are $R_1 + R_2$ (union), $R_1 R_2$ (concatenation), R_1^* (closure) and (R_1) .
5. Nothing else is a regular expression, i.e. only those obtained by finitely many applications of rules 1-4.

Precedence of operators (highest to lowest): closure $*$, then concatenation, then union $+$. Hence ab^*+c means $(a(b^*)) + c$. Brackets override this order.

3.1.3 Regular Set

A **regular set** is the set of strings denoted by a regular expression. Equivalently, the class of regular sets over Σ is the smallest class containing $\varnothing$, $\{\Lambda\}$ and $\{a\}$ for each $a \in \Sigma$ that is closed under union, concatenation and closure. Every **finite** language is regular.

Regular expression R	Regular set L(R)
$\varnothing$	$\{ \}$ — the empty set (no strings)
Λ	$\{\Lambda\}$ — the set containing only the empty string
a ($a \in \Sigma$)	$\{a\}$
$R_1 + R_2$	$L(R_1) \cup L(R_2)$
$R_1 R_2$	$L(R_1) L(R_2)$
R^*	$(L(R))^* = \{\Lambda\} \cup L(R) \cup L(R)L(R) \cup \dots$

Table 3.3: Regular expressions and the sets they represent

Remember: $\varnothing^* = \{\Lambda\}$, whereas $\varnothing$ alone is the empty set. Also $\varnothing \neq \Lambda$: the set $\{\Lambda\}$ has one string, the set $\varnothing$ has none.

3.1.4 Writing Regular Expressions - Examples

Let $\Sigma = \{a, b\}$ unless stated otherwise.

Language	Regular expression
All strings (including Λ)	$(a + b)^*$
Strings ending with b	$(a + b)^*b$
Strings ending with abb	$(a + b)^*abb$
Strings starting with a and ending with b	$a(a + b)^*b$
Strings containing the substring aa	$(a + b)^*aa(a + b)^*$
Strings with at least one a	$(a + b)^*a(a + b)^*$ or $b^*a(a + b)^*$
Strings with exactly two a's	$b^*ab^*ab^*$
Strings of even length	$((a + b)(a + b))^*$
Strings of a's of even length (a^{2n})	$(aa)^*$
Zero or more a's followed by exactly one b	a^*b
Over $\{0,1\}$: strings with no two consecutive 0's	$(1 + 01)^*(\Lambda + 0)$
Over $\{0,1\}$: binary strings with an even number of 1's	$0^*(10^*10^*)^*$

Table 3.4: Common regular expressions

3.1.5 Identities of Regular Expressions

Two regular expressions P and Q are **equivalent** ($P = Q$) if they denote the same regular set. The following identities hold for any regular expressions P, Q, R. They are used to simplify REs and to prove equivalences.

No.	Identity	Remark
I_1	$\varphi + R = R + \varphi = R$	φ is identity for union
I_2	$\varphi R = R\varphi = \varphi$	φ is zero for concatenation
I_3	$\Lambda R = R\Lambda = R$	Λ is identity for concatenation
I_4	$\Lambda^* = \Lambda$ and $\varphi^* = \Lambda$	
I_5	$R + R = R$	Idempotent law
I_6	$R^*R^* = R^*$	
I_7	$RR^* = R^*R$	Both equal R^+
I_8	$(R^*)^* = R^*$	
I_9	$\Lambda + RR^* = \Lambda + R^*R = R^*$	Very frequently used
I_{10}	$(PQ)^*P = P(QP)^*$	Shifting law
I_{11}	$(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$	Union inside a closure
I_{12}	$(P + Q)R = PR + QR$ and $R(P + Q) = RP + RQ$	Distributive laws
I_{13}	$P + Q = Q + P$	Commutative for +, but $PQ \neq QP$ in general

Table 3.5: Identities of regular expressions

Example 1: Prove that $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) = 0^*1(0 + 10^*1)^*$

Solution. Let $R = (1 + 00^*1)$ and $S = (0 + 10^*1)$. Then

- $LHS = R + R S^* S = R(\Lambda + S^*S)$ [distributive law, I_{12}, I_3]
- $= R S^* [I_9: \Lambda + S^*S = S^*]$
- $= (1 + 00^*1)(0 + 10^*1)^*$
- $= (\Lambda + 00^*)1(0 + 10^*1)^*$ [take 1 common]
- $= 0^*1(0 + 10^*1)^* = RHS [I_9: \Lambda + 00^* = 0^*]$ ■

3.1.6 Arden's Theorem

Arden's theorem is the main algebraic tool for solving the "state equations" of a finite automaton and finding its regular expression.

Statement. Let P and Q be regular expressions over Σ . If **P does not contain Λ** , then the equation **$R = Q + RP$** has a **unique solution** given by **$R = QP^*$** .

Proof (outline).

- *QP^* is a solution:* $R = Q + RP = Q + (QP^*)P = Q(\Lambda + P^*P) = QP^*$ (by I_9).
- *Uniqueness:* substitute $R = Q + RP$ repeatedly: $R = Q + QP + QP^2 + \dots + QP^n + RP^{n+1}$. Take any string $w \in R$ with $|w| = n$. Since P does not contain Λ , every string of RP^{n+1} has length at least $n + 1$, so w cannot lie in RP^{n+1} ; hence $w \in Q(\Lambda + P + \dots + P^n) \subseteq QP^*$. Conversely every string of QP^* is in R . ■

Why P must not contain Λ : if $\Lambda \in P$, the equation $R = Q + RP$ has more than one solution (for example, any $R \supseteq QP^*$ works), so the solution is no longer unique.

Equation	Solution (by Arden)
$R = a + Rb$	$R = ab^*$
$R = \Lambda + Ra$	$R = \Lambda \cdot a^* = a^*$
$R = 0 + R(1 + 01)$	$R = 0(1 + 01)^*$
$R = Q + RP$ with $P = \Lambda + a$	Not applicable (P contains Λ)

Table 3.6: Quick applications of Arden's theorem

Application: finding the regular expression of an FA

Conditions: the transition system must have *no Λ -moves* and exactly *one initial state*. Write one equation per state, in terms of the transitions that *enter* that state.

1. Give each state a variable (e.g. A, B, C). For every transition from X to Y on symbol a , add the term $X \cdot a$ to the equation of Y .
2. Add Λ to the equation of the initial state.
3. Eliminate variables one by one by substitution; whenever an equation has the form $R = Q + RP$, apply Arden's theorem to get $R = QP^*$.
4. The required RE is the **sum (union) of the expressions of all final states**.

Example 2: Find the RE accepted by the DFA below (start state A, final state C).

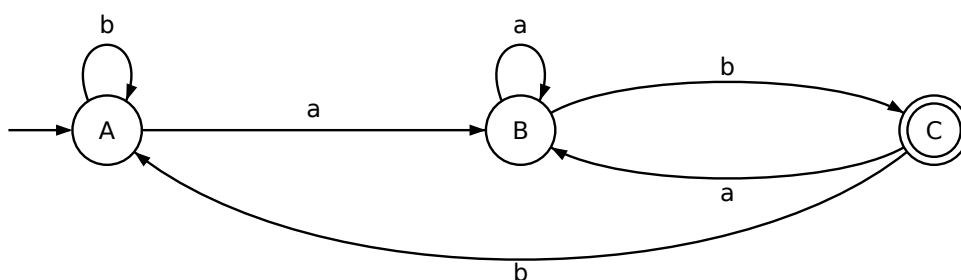


Fig. 3.1: DFA used in Example 2 (strings over $\{a, b\}$ ending in ab)

Step 1 - state equations (incoming transitions):

- $A = \Lambda + Ab + Cb$
- $B = Aa + Ba + Ca$
- $C = Bb$

Step 2 - substitute $C = Bb$: $A = \Lambda + Ab + Bbb$ and $B = Aa + Ba + Bba = Aa + B(a + ba)$.

Step 3 - Arden on B: $B = Aa + B(a + ba) \Rightarrow B = Aa(a + ba)^*$.

Step 4 - put B in A: $A = \Lambda + Ab + Aa(a + ba)^*bb = \Lambda + A[b + a(a + ba)^*bb] \Rightarrow A = [b + a(a + ba)^*bb]^*$.

Step 5 - final state C: $C = Bb = Aa(a + ba)^*b$, so

$$\text{RE} = [b + a(a + ba)^*bb]^* a (a + ba)^* b$$

This is equivalent to the simpler expression $(a + b)^*ab$, which we can see directly from the diagram: the DFA accepts exactly the strings ending in ab .

3.2 Regular Expressions and Finite Automata

3.2.1 Relation between Regular Expression and Finite Automata

The fundamental result connecting the two views is **Kleene's theorem**:

A language is regular $\Leftrightarrow$ it is described by a regular expression $\Leftrightarrow$ it is accepted by some DFA $\Leftrightarrow$ it is accepted by some NFA (with or without Λ -moves) $\Leftrightarrow$ it is generated by a regular grammar.

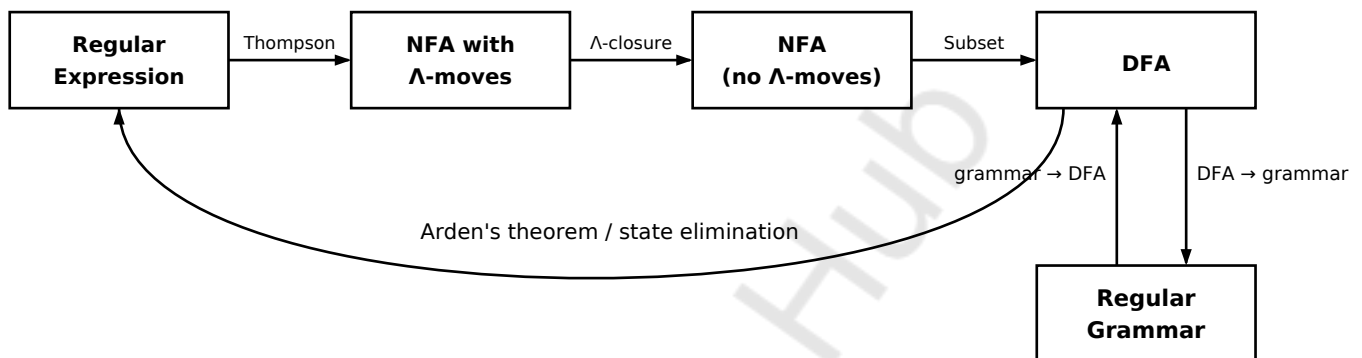


Fig. 3.2: Conversions among regular expressions, automata and grammars

Before studying the conversions, recall the three types of finite automata:

Feature	DFA	NFA	NFA with Λ -moves
Transition function	$\delta: Q \times \Sigma \rightarrow Q$	$\delta: Q \times \Sigma \rightarrow 2^Q$	$\delta: Q \times (\Sigma \cup \{\Lambda\}) \rightarrow 2^Q$
Moves on one symbol	Exactly one next state	Zero, one or many next states	Same, plus free Λ -moves
Moves without input	Not allowed	Not allowed	Allowed (Λ -transitions)
Power	Regular languages	Regular languages	Regular languages
Typical use	Implementation, lexers	Easy design	Easy construction from RE

Table 3.7: Comparison of DFA, NFA and NFA with Λ -moves

All are 5-tuples $\mathbf{M} = (Q, \Sigma, \delta, q_0, F)$ where Q is the finite set of states, Σ the input alphabet, δ the transition function, $q_0 \in Q$ the initial state and $F \subseteq Q$ the set of final states. A **transition system** is the same idea drawn as a labelled directed graph: vertices are states and edges are labelled by symbols (or Λ). A string is accepted if some path from the initial state to a final state spells it out.

3.2.2 Transition System Containing Λ -moves

A Λ -move lets the machine change state *without reading any input symbol*. Λ -moves do not add power, but they make constructions (especially from regular expressions) much easier. We can always remove them.

Λ -closure. The Λ -closure of a state q , written $\Lambda\text{-closure}(q)$, is the set of all states reachable from q using only Λ -transitions, *including q itself*. For a set of states, it is the union of the closures of its members.

Conversion of NFA with Λ -moves to NFA without Λ -moves.

1. Compute Λ -closure(q) for every state q .
2. For every state q and every $a \in \Sigma$ define $\delta'(q, a) = \Lambda\text{-closure}(\delta(\Lambda\text{-closure}(q), a))$.
3. The new final states are $F' = F \cup \{ q \mid \Lambda\text{-closure}(q) \text{ contains a state of } F \}$.
4. The initial state stays the same. Delete all Λ -edges (and unreachable states, if any).

Poly Notes Hub

Example 3: Remove the Λ -moves from the transition system of Fig. 3.3 (initial q_0 , final q_2).

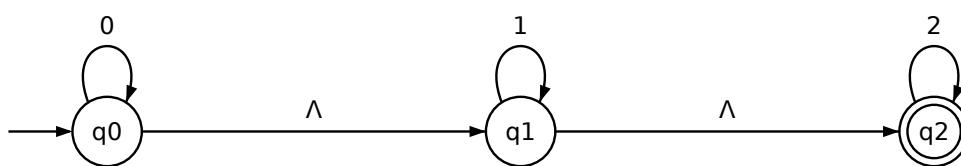


Fig. 3.3: NFA with Λ -moves (accepts $0^*1^*2^*$)

State q	Λ -closure(q)
q_0	$\{q_0, q_1, q_2\}$
q_1	$\{q_1, q_2\}$
q_2	$\{q_2\}$

Table 3.8: Λ -closures

State	Input 0	Input 1	Input 2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	$\varnothing$	$\{q_1, q_2\}$	$\{q_2\}$
q_2	$\varnothing$	$\varnothing$	$\{q_2\}$

Table 3.9: δ' after removing Λ -moves

For example, $\delta'(q_0, 0) = \Lambda\text{-closure}(\delta(\{q_0, q_1, q_2\}, 0)) = \Lambda\text{-closure}(\{q_0\}) = \{q_0, q_1, q_2\}$. Since $\Lambda\text{-closure}(q_0)$ and $\Lambda\text{-closure}(q_1)$ contain the final state q_2 , all three states become final.

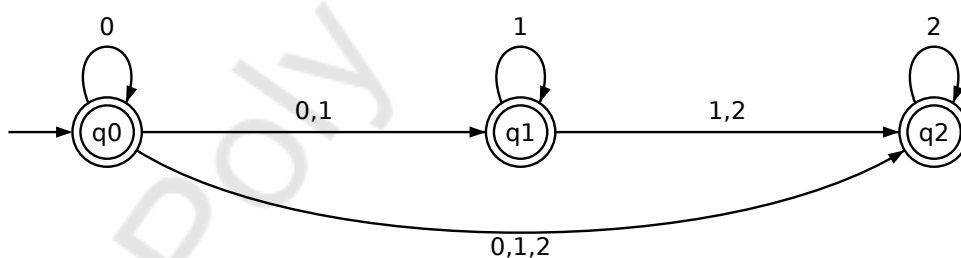


Fig. 3.4: Equivalent NFA without Λ -moves (all states final)

3.2.3 Conversion of Non-deterministic System to Deterministic System

Every NFA can be converted into an equivalent DFA by the **subset construction**: each DFA state is a *set* of NFA states, recording all states in which the NFA could be after reading the input so far.

1. The start state of the DFA is the set $\{q_0\}$.
2. For every DFA state S (a subset) and every $a \in \Sigma$ compute $\delta_D(S, a) = \cup \delta(q, a)$ over all $q \in S$.

3. Every newly obtained subset becomes a new DFA state. Repeat until no new subset appears.
4. A DFA state is final if it contains **at least one** final state of the NFA. The empty subset $\varnothing$ acts as the dead (trap) state.

Remarks: an NFA with n states can give up to 2^n DFA states in the worst case; constructing only the *reachable* subsets usually gives far fewer. If the NFA has Λ -moves, first remove them (3.2.2) or start with the Λ -closure of q_0 .

Poly Notes Hub

Example 4: Convert the NFA for $(a + b)^*abb$ into a DFA.

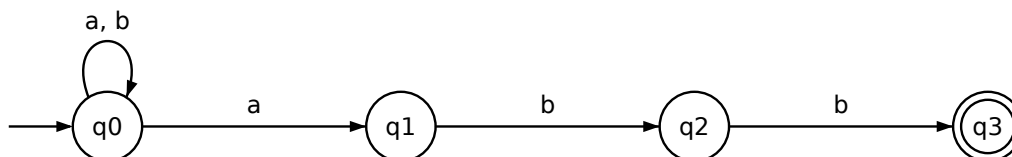


Fig. 3.5: NFA accepting strings ending in abb

DFA state	Subset of NFA states	On a	On b
A (start)	{q0}	{q0, q1} = B	{q0} = A
B	{q0, q1}	{q0, q1} = B	{q0, q2} = C
C	{q0, q2}	{q0, q1} = B	{q0, q3} = D
D (final)	{q0, q3}	{q0, q1} = B	{q0} = A

Table 3.10: Subset construction

Only four of the $2^4 = 16$ possible subsets are reachable, so the DFA has four states. D is final because it contains $q3$.

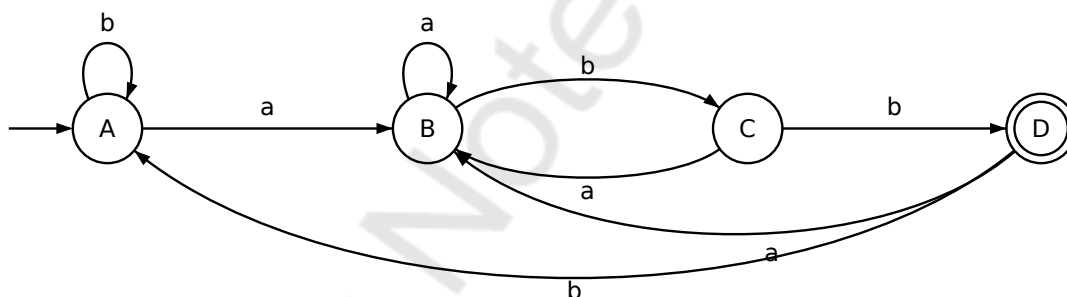


Fig. 3.6: Resulting DFA

3.2.4 Construction of Finite Automata Equivalent to a Regular Expression

Given a regular expression R we build an NFA with Λ -moves M_R that accepts $L(R)$ (Thompson's construction), with one initial and one final state, by induction on the structure of R .

Regular expression	Machine
$\varnothing$	Two states, initial and final, with no transition between them.
Λ	One state that is both initial and final (or initial $\rightarrow \Lambda \rightarrow$ final).
$a \in \Sigma$	Initial state $\rightarrow a \rightarrow$ final state.
$R_1 + R_2$	New initial state s with Λ -edges to the initial states of N_1 and N_2 ; Λ -edges from their final states to a new final state f .
$R_1 R_2$	Join the final state of N_1 to the initial state of N_2 (merge them or add a Λ -edge).
R^*	New s and f ; Λ -edges $s \rightarrow N$, $N \rightarrow f$, $f \rightarrow N$'s start (repeat) and $s \rightarrow f$ (zero repetitions).

Table 3.11: Building blocks of the construction

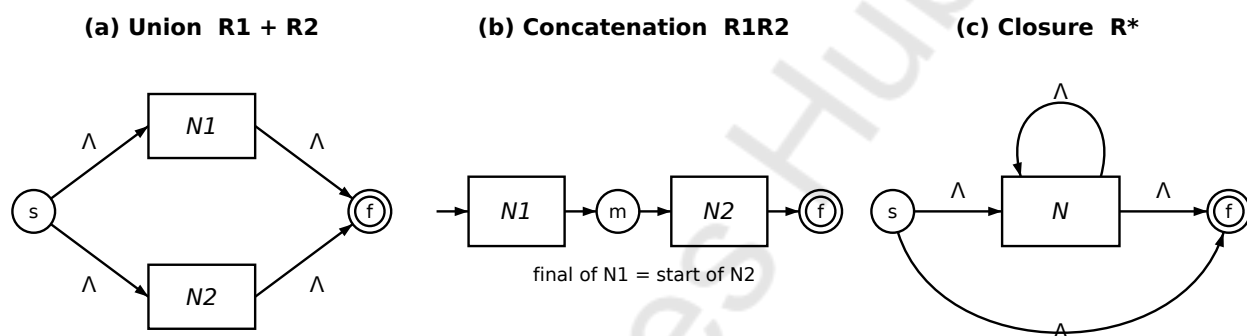


Fig. 3.7: Union, concatenation and closure constructions ($N, N1, N2$ are machines for sub-expressions)

Procedure: (1) parse R into its sub-expressions; (2) build machines for symbols; (3) combine them bottom-up using the three rules; (4) if a DFA is required, remove Λ -moves (3.2.2) and apply the subset construction (3.2.3); optionally minimise.

Example 5: Construct a finite automaton for $R = (a + b)^*b$.

Build $(a + b)$ by the union rule (states $q1$ - $q6$), apply closure (the Λ -edges $q6 \rightarrow q1$ for repetition and $q0 \rightarrow q7$ for zero repetitions), then concatenate b ($q7 \rightarrow b \rightarrow q8$).

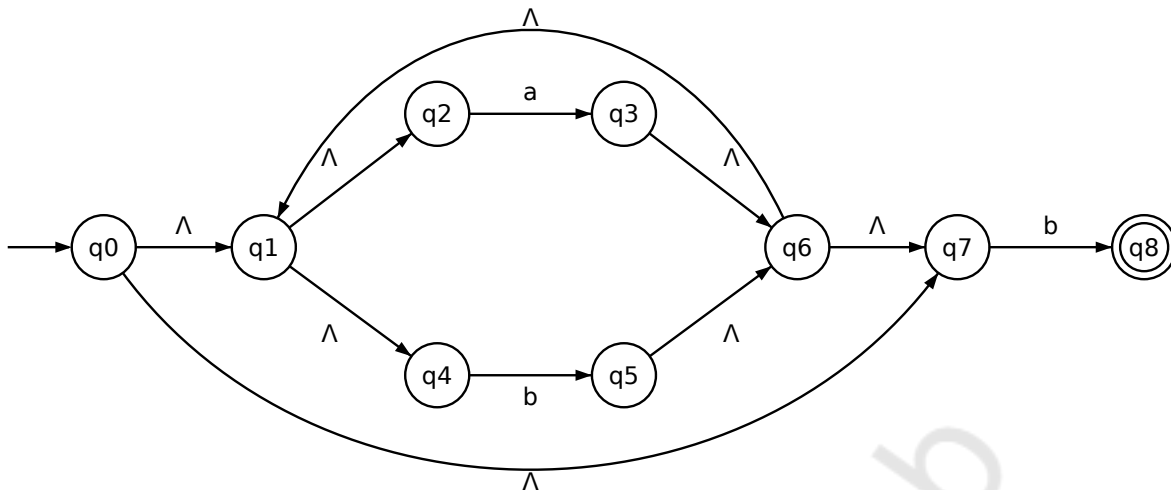


Fig. 3.8: NFA with Λ -moves for $(a + b)^*b$ (start q_0 , final q_8)

Check: the string ab : $q_0 \xrightarrow{\Lambda} q_1 \xrightarrow{\Lambda} q_2 \xrightarrow{a} q_3 \xrightarrow{\Lambda} q_6 \xrightarrow{\Lambda} q_7 \xrightarrow{b} q_8$ (accepted). The string Λ is rejected, since q_8 cannot be reached without reading b ; and every accepted string ends in b — so the machine accepts exactly the strings ending in b .

3.2.5 Equivalence of Two Finite Automata

Two finite automata M_1 and M_2 are **equivalent** if they accept the same language, $L(M_1) = L(M_2)$. Methods:

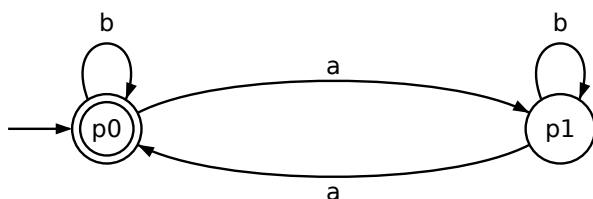
- **Comparison (pair) method** - explore pairs of states reached by the same input; described below.
- **Minimisation method** - minimise both DFAs; they are equivalent iff the minimal DFAs are identical up to renaming of states.
- **Product method** - build the DFA for the symmetric difference $L(M_1) \oplus L(M_2)$; the automata are equivalent iff this language is empty.

Comparison method (for DFAs M_1, M_2 over the same Σ):

1. Start with the pair (q_0, q_0') of initial states.
2. For each pair (p, q) in the table and each input a , write the pair $(\delta_1(p, a), \delta_2(q, a))$. Add any new pair as a new row.
3. If any pair contains **one final and one non-final** state, the automata are **not equivalent** - stop.
4. If no new pairs appear and every pair is “both final or both non-final”, the automata are **equivalent**.

Example 6: Check whether M1 and M2 are equivalent.

M1



M2

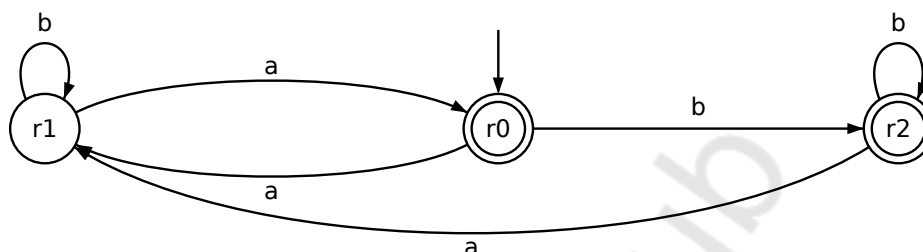


Fig. 3.9: M1 (start p0, final p0) and M2 (start r0, finals r0 and r2)

Pair (M1, M2)	On a	On b	Finality check
(p0, r0)	(p1, r1)	(p0, r2)	both final - OK
(p1, r1)	(p0, r0)	(p1, r1)	both non-final - OK
(p0, r2)	(p1, r1)	(p0, r2)	both final - OK

Table 3.12: Comparison table

No new pair appears and every pair is consistent, so **M1** $\equiv$ **M2**; both accept strings with an even number of a's. Had r2 been non-final, the pair (p0, r2) would have a mismatch and the automata would not be equivalent.

3.2.6 Equivalence of Two Regular Expressions

Two regular expressions R_1 and R_2 are equivalent if $L(R_1) = L(R_2)$. There are three standard approaches:

Method	Idea	Best used when
Algebraic	Transform one side into the other using identities I_1 - I_{13} and Arden's theorem.	Expressions are structurally close
Automata	Build DFAs for both, minimise, and compare (3.2.5).	Algebra does not work easily
Set-based	Show $L(R_1) \subseteq L(R_2)$ and $L(R_2) \subseteq L(R_1)$ by reasoning on strings.	Small or simple languages

Table 3.13: Methods for equivalence of regular expressions

Example 7: Show that $(a + b)^* = (a^*b^*)^*$.

By identity I_{11} , $(P + Q)^* = (P^*Q^*)^*$. Taking $P = a$ and $Q = b$ gives the result immediately.

■

Example 8: Prove that $\Lambda + 1^*(011)^*(1^*(011)^*)^* = (1 + 011)^*$.

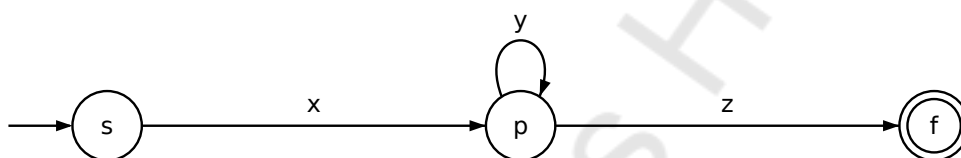
Let $R = 1^*(011)^*$. Then $\text{LHS} = \Lambda + R R^*$, which equals R^* by I_9 . So $\text{LHS} = (1^*(011)^*)^*$. By I_{11} with $P = 1$ and $Q = 011$ we have $(P^*Q^*)^* = (P + Q)^*$, therefore $\text{LHS} = (1 + 011)^* = \text{RHS}$. ■

3.2.7 Pumping Lemma for Regular Sets

The pumping lemma gives a **necessary** property of every regular language. It is used mainly to **prove that a language is not regular**.

Statement. Let L be an infinite regular language. Then there is a constant n (the pumping length, at most the number of states of a DFA for L) such that every string $w \in L$ with $|w| \geq n$ can be written as $w = xyz$ satisfying:

1. $|y| \geq 1$ (the part y is non-empty);
2. $|xy| \leq n$ (the loop occurs within the first n symbols);
3. $xy^iz \in L$ for every $i \geq 0$ (y can be repeated or deleted).



$w = xyz$, xy^iz ($i = 0, 1, 2, \dots$) is also accepted

Fig. 3.10: Idea of the lemma - a long string must revisit a state, creating a loop labelled y

Reason: a DFA with n states that reads a string of length $\geq n$ must repeat some state (pigeonhole principle). The portion of the path between the two visits is a loop y ; it can be traversed any number of times (or skipped) and the string is still accepted.

Proving a language is not regular - the “game”:

1. Assume L is regular. Let n be its pumping length (an unknown constant).
2. Choose a string $w \in L$ with $|w| \geq n$ - the choice should depend on n .
3. For every split $w = xyz$ with $|xy| \leq n$ and $|y| \geq 1$, find an i such that $xy^iz \notin L$.
4. This contradicts the lemma, so L is not regular.

Caution: the pumping lemma can *only* show non-regularity. A language that satisfies the lemma may still be non-regular, so it can never be used to prove a language is regular.

Example 9: Show that $L = \{a^m b^m \mid m \geq 0\}$ is not regular.

- Assume L is regular with pumping length n . Choose $w = a^n b^n \in L$, $|w| = 2n \geq n$.
- Since $|xy| \leq n$, both x and y lie in the block of a 's: $y = a^k$ with $k \geq 1$.
- Take $i = 2$: $xy^2z = a^{n+k} b^n$, which has more a 's than b 's, so it is not in L .
- Contradiction. Hence L is **not regular**. ■

Example 10: Show that $L = \{a^p \mid p \text{ is a prime number}\}$ is not regular.

- Assume L is regular with pumping length n . Choose a prime $p \geq n$ and $w = a^p$.
- Write $w = xyz$ with $y = a^k$, $1 \leq k \leq n$. Then $|xz| = p - k$.
- Pump with $i = p + 1$: $|xy^{p+1}z| = (p - k) + k(p + 1) = p + kp = p(1 + k)$.
- Since $p \geq 2$ and $1 + k \geq 2$, this number is composite, so $xy^{p+1}z \notin L$. Contradiction, hence L is **not regular**. ■

Language	Regular?	Reason
a^*b^*	Yes	RE exists
$\{a^m b^m \mid m \geq 0\}$	No	Needs unbounded counting
$\{ww \mid w \in \{a, b\}^*\}$	No	Must remember the whole first half
$\{w \mid \#a(w) = \#b(w)\}$	No	Pump $w = a^n b^n$
$\{a^{m^2} \mid m \geq 0\}$	No	Gaps between squares keep growing
$\{w \mid w \text{ has an even number of } a\text{'s}\}$	Yes	Two-state DFA
Any finite language	Yes	Union of finitely many strings

Table 3.14: Examples of regular and non-regular languages

3.2.8 Closure Properties of Regular Sets

A class of languages is **closed** under an operation if applying the operation to members of the class always gives a member of the class. Regular sets are closed under all the common operations:

Operation	Closed?	Construction / reason
Union $L_1 \cup L_2$	Yes	RE $R_1 + R_2$; or NFA with a new start state and Λ -edges to both machines.
Concatenation $L_1 L_2$	Yes	RE $R_1 R_2$; Λ -edges from finals of M_1 to the start of M_2 .
Kleene closure L^*	Yes	RE R^* ; add Λ -edges from finals back to the start and a new start/final state.
Complement $\Sigma^* - L$	Yes	Take a <i>complete</i> DFA for L and swap final and non-final states.
Intersection $L_1 \cap L_2$	Yes	$L_1 \cap L_2 = (L_1' \cup L_2)'$ (De Morgan), or the product automaton.
Difference $L_1 - L_2$	Yes	$L_1 \cap L_2'$.
Reversal L^R	Yes	Reverse all edges, swap initial and final states (use a new start with Λ -edges if several finals).
Homomorphism $h(L)$	Yes	Replace each symbol a in the RE by the string $h(a)$.
Inverse homomorphism	Yes	Run the DFA on $h(a)$ for each input symbol a .
Infinite union	No	Union of the regular sets $\{a^m b^m\}$ for $m = 0, 1, 2, \dots$ is not regular.

Table 3.15: Closure properties

Uses of closure properties: to prove a language regular, express it from known regular languages using these operations; to prove a language non-regular, assume it is regular

and derive a known non-regular language (e.g. $\{a^m b^m\}$) using closure operations, giving a contradiction.

Poly Notes Hub

Decision properties of regular languages:

Question	Method
Is w in L ? (membership)	Run the DFA on w .
Is L empty?	Check whether a final state is reachable from the start state.
Is L finite?	L is infinite iff a cycle lies on some path from the start to a final state.
Is $L_1 = L_2$?	Equivalence of automata (3.2.5).

Table 3.16: Decision properties

3.2.9 Regular Grammar, DFA and Transition System

A **regular grammar** (Type-3 grammar) is a grammar $G = (V, T, P, S)$ where V is the set of variables, T the terminals, P the productions and S the start symbol. Two forms exist:

Type	Allowed productions ($A, B \in V, a \in T$)
Right-linear	$A \rightarrow aB, A \rightarrow a, A \rightarrow \Lambda$
Left-linear	$A \rightarrow Ba, A \rightarrow a, A \rightarrow \Lambda$

Table 3.17: Forms of regular grammars

Both generate exactly the regular languages (we use right-linear grammars, mixing the two forms in one grammar is not allowed).

(a) Regular grammar for a given DFA

1. Take the states of the DFA as the variables; the initial state is the start symbol S .
2. For every transition $\delta(A, a) = B$ write the production $A \rightarrow aB$.
3. If B is a final state, also write $B \rightarrow \Lambda$ (or add $A \rightarrow a$).

Example 11: Write a regular grammar for the DFA of Fig. 3.1 (strings ending in ab).

Transition	Production
$\delta(A, a) = B, \delta(A, b) = A$	$A \rightarrow aB \mid bA$
$\delta(B, a) = B, \delta(B, b) = C$	$B \rightarrow aB \mid bC$
$\delta(C, a) = B, \delta(C, b) = A; C \text{ is final}$	$C \rightarrow aB \mid bA \mid \Lambda$

Table 3.18: Productions from the DFA

Grammar: $G = (\{A, B, C\}, \{a, b\}, P, A)$. **Derivation of aab:** $A \Rightarrow aB \Rightarrow aaB \Rightarrow aabC \Rightarrow aab$ (using $C \rightarrow \Lambda$).

(b) Transition system for a given regular grammar

1. Introduce one state for each variable (the start symbol becomes the initial state) and one extra **final state F**.
2. For $A \rightarrow aB$ add an edge $A \xrightarrow{a} B$.
3. For $A \rightarrow a$ add an edge $A \xrightarrow{a} F$.
4. For $A \rightarrow \Lambda$ make A a final state (or add $A \xrightarrow{\Lambda} F$). The result is generally an NFA.

Example 12: Construct a transition system for $G: S \rightarrow aS \mid bA \mid b, A \rightarrow aA \mid bS \mid a$.

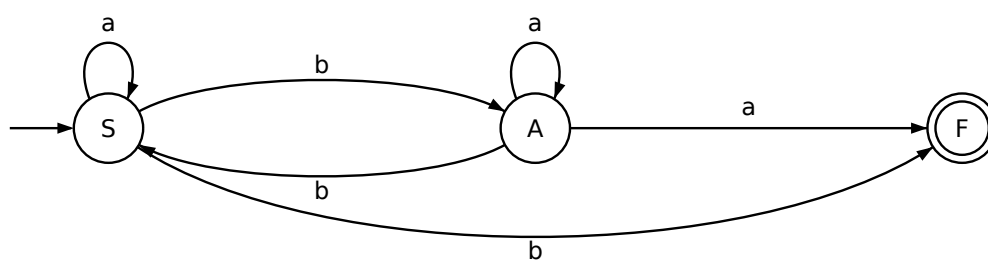


Fig. 3.11: Transition system for G (initial S , final F)

Reading the diagram: S and A record whether the number of b 's so far is even or odd, and the machine moves to F on the last symbol only when the total number of b 's is odd. Hence $L(G) = \{w \in \{a, b\}^* \mid w \text{ has an odd number of } b\text{'s}\}$.

Left-linear grammars: reverse the productions ($A \rightarrow Ba$ becomes $A \rightarrow aB$), obtain the automaton for the reversed language, then reverse the automaton (swap start and final states, reverse edges).

Chapter Summary - Conversion Chart

To convert	Use
FA $\rightarrow$ RE	Arden's theorem (state equations) or state elimination
RE $\rightarrow$ FA	Thompson's construction $\rightarrow$ NFA with Λ -moves
NFA with Λ -moves $\rightarrow$ NFA	Λ -closure: $\delta'(q, a) = \Lambda\text{-closure}(\delta(\Lambda\text{-closure}(q), a))$
NFA $\rightarrow$ DFA	Subset construction (up to 2^n states)
DFA $\rightarrow$ regular grammar	$\delta(A, a) = B \Rightarrow A \rightarrow aB$; final state $B = B \rightarrow \Lambda$
Regular grammar $\rightarrow$ FA	Variables $\rightarrow$ states, add final state F ; $A \rightarrow aB \Rightarrow A \xrightarrow{a} B$; $A \rightarrow a \Rightarrow A \xrightarrow{a} F$
Is a language non-regular?	Pumping lemma (by contradiction) / closure properties

Table 3.19: Conversions at a glance

Key points to remember

- $\varphi^* = \Lambda$, $\Lambda + RR^* = R^*$, $(P + Q)^* = (P^*Q^*)^*$.
- Arden: $R = Q + RP$, $\Lambda \notin P \Rightarrow R = QP^*$.
- Λ -closure of a state always contains the state itself.
- In the subset construction, a subset is final if it contains *any* NFA final state.
- Pumping lemma: $|y| \geq 1$, $|xy| \leq n$, $xy^iz \in L$ for all $i \geq 0$ - used only to disprove regularity.
- Regular sets are closed under union, concatenation, closure, complement, intersection, difference and reversal.

Multiple Choice Questions (15)

Choose the correct option for each question. Answers are given in the answer key that follows.

Q1. The regular expression $\varnothing^*$ denotes

- (a) $\varnothing$
- (b) $\{\Lambda\}$
- (c) Σ^*
- (d) undefined

Q2. Which of the following is **not** a valid identity of regular expressions?

- (a) $R + R = R$
- (b) $(R^*)^* = R^*$
- (c) $\varnothing R = R$
- (d) $\Lambda R = R$

Q3. According to Arden's theorem, if $\Lambda \notin P$, the unique solution of $R = Q + RP$ is

- (a) $R = QP^*$
- (b) $R = P^*Q$
- (c) $R = Q^*P$
- (d) $R = PQ^*$

Q4. $(P + Q)^*$ is equal to

- (a) $P^* + Q^*$
- (b) P^*Q^*
- (c) PQ^*
- (d) $(P^*Q^*)^*$

Q5. The Λ -closure of a state q is

- (a) the states reachable from q by reading exactly one symbol
- (b) the states reachable from q using only Λ -moves, including q itself
- (c) all states reachable from q by any path
- (d) the set of final states

Q6. In the worst case, the DFA obtained from an NFA with n states has

- (a) n^2 states
- (b) $2n$ states
- (c) 2^n states
- (d) $n!$ states

Q7. Thompson's construction converts a regular expression into

- (a) an NFA with Λ -moves
- (b) a minimal DFA
- (c) a pushdown automaton
- (d) a Turing machine

Q8. Which of the following languages is **not** regular?

- (a) a^*b^*
- (b) $(ab)^*$
- (c) strings over $\{a, b\}$ with an even number of a 's
- (d) $\{a^n b^n \mid n \geq 0\}$

Q9. For the pumping lemma, $w = xyz$ must satisfy

- (a) $|y| \geq 1$, $|xy| \leq n$ and $xy^i z \in L$ for all $i \geq 0$
- (b) $|y| = 0$ and $|xy| \geq n$
- (c) $|xy| > n$ and $xy^i z \in L$ only for $i = 1$
- (d) $|x| = |z|$ and $xy^i z \notin L$

Q10. Regular languages are closed under

- (a) union only
- (b) union, intersection and complement
- (c) union and concatenation but not star
- (d) none of the operations

Q11. If $\delta(A, a) = B$ in a DFA, the corresponding right-linear production is

- (a) $a \rightarrow AB$ (b) $A \rightarrow Ba$
(c) $B \rightarrow aA$ (d) $A \rightarrow aB$

Q12. In the comparison method for equivalence of two DFAs, every pair (q_i, q_j) in the table must be

- (a) both final (b) both non-final
(c) both final or both non-final (d) exactly one final

Q13. Which RE denotes the set of all strings over $\{a, b\}$ containing at least one a ?

- (a) a^* (b) b^*a^*
(c) $(a + b)^*a(a + b)^*$ (d) $(a + b)^*$

Q14. The minimum number of states in a DFA for $(a + b)^*abb$ is

- (a) 3 (b) 4
(c) 5 (d) 8

Q15. $\Lambda + RR^*$ is equal to

- (a) R^* (b) R
(c) φ (d) RR

Answer Key

Q.	Ans.	Q.	Ans.	Q.	Ans.
1	(b)	6	(c)	11	(d)
2	(c)	7	(a)	12	(c)
3	(a)	8	(d)	13	(c)
4	(d)	9	(a)	14	(b)
5	(b)	10	(b)	15	(a)

Table 3.20: Answer key for MCQs

Broad Questions (10)

- Q1.** Define a regular expression and a regular set. Write the regular expressions for (i) strings over $\{a, b\}$ ending with ab , (ii) strings over $\{0, 1\}$ with exactly two 1's, and (iii) strings over $\{a, b\}$ of odd length, and state the sets they represent.
- Q2.** State and prove Arden's theorem. Explain why the condition $\Lambda \notin P$ is necessary.
- Q3.** List the identities of regular expressions. Prove that $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) = 0^*1(0 + 10^*1)^*$ using identities.
- Q4.** Using Arden's theorem, find the regular expression for the DFA with states q_1 (initial), q_2 , q_3 (final) and transitions: $\delta(q_1, 0) = q_2$, $\delta(q_1, 1) = q_1$, $\delta(q_2, 0) = q_3$, $\delta(q_2, 1) = q_1$, $\delta(q_3, 0) = q_3$, $\delta(q_3, 1) = q_3$.
- Q5.** What is a transition system with Λ -moves? Define Λ -closure. Convert the NFA with Λ -moves having states q_0 , q_1 , q_2 (initial q_0 , final q_2) and transitions $\delta(q_0, a) = \{q_0\}$, $\delta(q_0, \Lambda) = \{q_1\}$, $\delta(q_1, b) = \{q_1\}$, $\delta(q_1, \Lambda) = \{q_2\}$, $\delta(q_2, c) = \{q_2\}$ into an NFA without Λ -moves.
- Q6.** Explain the subset construction. Convert the NFA with states $\{q_0, q_1, q_2\}$ (initial q_0 , final q_2) and transitions $\delta(q_0, 0) = \{q_0, q_1\}$, $\delta(q_0, 1) = \{q_0\}$, $\delta(q_1, 1) = \{q_2\}$, $\delta(q_2, 0) = \varnothing$, $\delta(q_2, 1) = \varnothing$ (others $\varnothing$) into an equivalent DFA.
- Q7.** Describe Thompson's construction of a finite automaton from a regular expression, with diagrams for union, concatenation and closure. Construct an NFA with Λ -moves for the regular expression $10 + (0 + 11)0^*1$.
- Q8.** Explain with an example the comparison method for checking equivalence of two finite automata. Also prove that $(a + b)^* = (a^*b^*)^*$ and $\Lambda + 1^*(011)^*(1^*(011)^*)^* = (1 + 011)^*$.
- Q9.** State the pumping lemma for regular sets. Use it to prove that (i) $L = \{a^m b^m \mid m \geq 0\}$ and (ii) $L = \{a^p \mid p \text{ is prime}\}$ are not regular.
- Q10.** List the closure properties of regular sets and prove closure under complement and intersection. Also construct a regular grammar for a given DFA and a transition system for the grammar $S \rightarrow aS \mid bA \mid b$, $A \rightarrow aA \mid bS \mid a$, stating the language generated.
-

— End of Chapter 3 —