NEC Registration Examination

Computer Engineering (ACtE)

Chapter 6: Theory of Computation and Computer Graphics (ACtE06)

Computer Engineering (ACtE)
Electronics and Communication
Telecommunication
Software Engineering
Information Technology

DFA/NFA Comparison • CFG Normal Forms • Turing Machine Variants • Transformation Matrices • Projection Tables • Diagrams • Practice MCQs

COMPLETE NOTES WITH SOLVED MCQ AND PYQS

EASY EXPLANATION



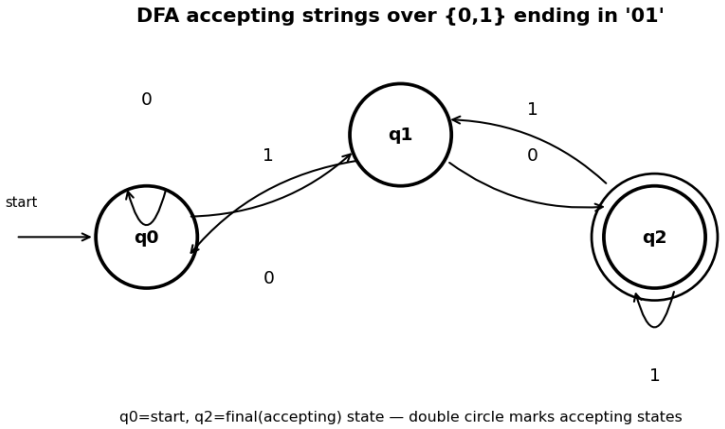
6.1 Introduction to Finite Automata

Formal Definition of a Finite Automaton

FA = (Q, Σ, δ, q_o, F)

Symbol	Meaning
Q	Finite set of STATES
Σ	Input ALPHABET (finite set of symbols)
δ	TRANSITION function: Q × Σ → Q
q_o	START state (q_o ∈ Q)
F	Set of FINAL/accepting states (F ⊆ Q)

DFA vs NFA — Detailed Comparison

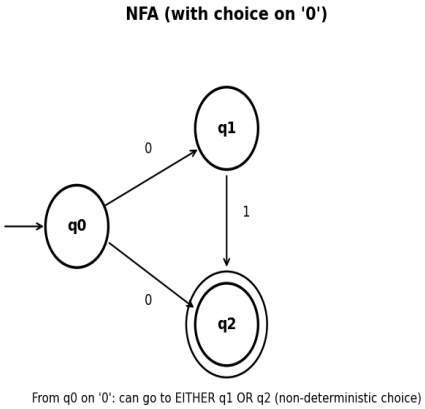


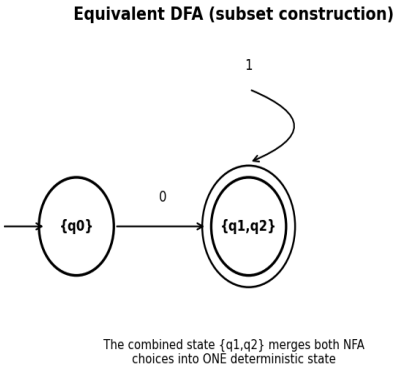
Feature	DFA	NFA
Full form	Deterministic Finite Automaton	Non-deterministic Finite Automaton
Transitions per (state,symbol)	EXACTLY ONE	Zero, one, or MANY
ε (epsilon) moves	NOT allowed	ALLOWED
Backtracking needed?	No	Conceptually yes (or parallel simulation)
Ease of construction	Harder to design directly	Easier — more intuitive
Memory needed to simulate	O(1) — track a single state	O(n) — must track a SET of states
Expressive power	Same as NFA	Same as DFA (equivalent)
Speed of a single run	Faster	Conceptually slower (or needs subset simulation)

Easy Explanation

Common trap: NFA and DFA are equally POWERFUL (they accept exactly the regular languages) — NFA is only more CONVENIENT to design, never more capable. Every NFA can be converted to an equivalent DFA.

Subset (Powerset) Construction — NFA to DFA





Step	Action
1	Start DFA state = the SET containing the NFA's start state (plus ε-closure if any)
2	For each DFA state (a set of NFA states) and each input symbol, compute the UNION of all NFA transitions
3	That union becomes a new DFA state (create it if not already present)
4	Repeat until no new DFA states are generated
5	A DFA state is FINAL if it contains at least one NFA final state

Worst case: an NFA with n states can require up to 2^n DFA states after subset construction (though often far fewer).

DFA Minimization

- Goal: find the **smallest equivalent DFA** (fewest states) that accepts the same language.

Step	Description
1	Remove UNREACHABLE states (not reachable from q_0)
2	Partition states into FINAL and NON-FINAL groups
3	Repeatedly SPLIT groups where two states in the same group go to DIFFERENT groups on some input symbol
4	Stop when no further splitting occurs — remaining groups are the MINIMIZED states

This is known as the **table-filling method** or based on the **Myhill-Nerode theorem** (two states are equivalent if they are indistinguishable for all possible input suffixes).

Regular Expressions

Operator	Symbol	Meaning	Example
Union	`+ or |`	Either alternative	`0+1 matches '0' OR '1'`
Concatenation	`. (often omitted)`	One followed by another	`ab matches 'ab'`
Kleene star	*	Zero or more repetitions	`a* matches ε, a, aa, aaa...`
Kleene plus	+	ONE or more repetitions	`a⁺ matches a, aa, aaa... (NOT ε)`
Optional	?	Zero or one occurrence	`a? matches ε or a`

Kleene's Theorem: Regular Expressions ≡ Finite Automata
(every RE has an equivalent FA, and every FA has an equivalent RE)

Regular expression	Language described
`(0+1)*`	All strings of 0s and 1s (including ε)
`1(0+1)*`	All strings starting with 1

(0+1)*00	All strings ENDING in 00
0*1*	Any number of 0s followed by any number of 1s
(ab)*	ε, ab, abab, ababab...

Pumping Lemma for Regular Languages

For regular language L, ∃ pumping length p such that ANY string w ∈ L with |w| ≥ p can be split as w = xyz where:

|xy| ≤ p, |y| ≥ 1, and $xy^iz \in L$ for ALL i ≥ 0

Use	Can it be used to...?
Prove a language is NOT regular	YES — find a contradiction (this is its only real use)
Prove a language IS regular	NO — satisfying the lemma is necessary but NOT sufficient

Classic proof pattern (showing L = {0^n1^n} is not regular):

Step	Action
1	Assume L IS regular, so the pumping lemma must apply with some length p
2	Choose the string w = 0^p1^p (which is in L and has length ≥ p)
3	Since |xy| ≤ p, y must consist ENTIRELY of 0s
4	Pump y (set i=2): the new string has MORE 0s than 1s, so it is NOT in L
5	This contradicts the pumping lemma — so L is NOT regular

Closure Properties of Regular Languages

Operation	Closed?
Union	YES
Concatenation	YES
Kleene star	YES
Intersection	YES
Complement	YES
Difference	YES
Reversal	YES

Easy Explanation

Regular languages are closed under EVERY common set operation — union, intersection, complement, concatenation, star, reversal. This is unlike context-free languages, which are NOT closed under intersection or complement (covered in the next section).

? Probable MCQs: Finite Automata

1. In a DFA, transitions for each (state, symbol) pair must be:

a) Zero or more
b) Exactly one
c) At least two
d) Undefined
✓Answer: b) Exactly one
Why: Determinism requires exactly one transition per state-symbol combination.

2. An NFA with n states, after subset construction, may require up to how many DFA states?

a) n
b) 2n
c) n^2
d) 2^n
✓Answer: d) 2^n
Why: The powerset of n states has 2^n possible subsets, the worst-case bound.

3. DFA minimization is based on which theorem?

a) Pumping lemma
b) Myhill-Nerode theorem
c) Kleene's theorem
d) Church-Turing thesis
✓Answer: b) Myhill-Nerode theorem
Why: Myhill-Nerode formalizes when two states are indistinguishable and can be merged.

4. Which regular expression describes all strings ending in '00'?

a) 00(0+1)*
b) (0+1)*00
c) 0*0*
d) (0+1)00*
✓Answer: b) (0+1)*00
Why: Any prefix followed by exactly '00' at the end.

5. The Pumping Lemma can be used to prove that a language is:

a) Regular
b) NOT regular
c) Context-free
d) Decidable
✓Answer: b) NOT regular
Why: It only provides a necessary condition, useful for proving non-regularity via contradiction.

6. Regular languages are NOT closed under which of these?

a) Union
b) Intersection
c) Complement
d) None — closed under all of these

✓**Answer: d) None — closed under all of these**

Why: Regular languages are closed under union, intersection, complement and more — trick question.

7. Kleene's theorem establishes equivalence between:

a) CFG and PDA

b) Regular expressions and finite automata

c) PDA and Turing machines

d) CFG and Turing machines

✓**Answer: b) Regular expressions and finite automata**

Why: REs and FAs describe exactly the same class: regular languages.

6.2 Introduction to Context-Free Language

Context-Free Grammar — Formal Definition

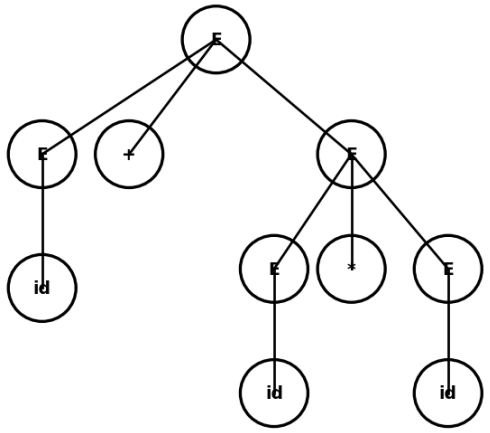
CFG = (V, T, P, S)

Symbol	Meaning
V	Set of VARIABLES (non-terminals)
T	Set of TERMINALS
P	Set of PRODUCTION rules, each of form A → α (A ∈ V, α ∈ (V∪T)*)
S	START symbol (S ∈ V)

Derivations and Parse Trees

Derivation type	Rule for choosing next non-terminal to expand
Leftmost derivation (LMD)	`Always expand the LEFTMOST non-terminal first`
Rightmost derivation (RMD)	`Always expand the RIGHTMOST non-terminal first`

Parse Tree for E → E+E | E*E | id (string: id+id*id)



- **Parse tree:** graphical representation of a derivation. **Leaves = terminals** (read left to right = the derived string); **internal nodes = variables**.
- **Ambiguous grammar:** a grammar where **some string has more than one parse tree** (equivalently, more than one leftmost derivation). Problematic for programming languages, since it allows multiple interpretations of the same code.

Classic ambiguity example: grammar `E → E+E | E*E | id` is ambiguous for `id+id*id` — it can be parsed as `(id+id)*id` or `id+(id*id)`, giving two different parse trees with two different meanings.

Normal Forms — Detailed

Normal Form	Production rule shape	Key property
CNF (Chomsky Normal Form)	`A → BC` or `A → a` (B, C non-terminals; a terminal)	Every derivation of a string of length n takes exactly 2n−1 steps
GNF (Greibach Normal Form)	`A → aα` (a = single terminal, α = 0 or more non-terminals)	Every production starts with exactly one terminal — useful for TOP-DOWN parsing
BNF (Backus-Naur Form)	`A ::= expression1 | expression2 | ...`	A NOTATION style (not a restriction) — the standard for specifying programming language syntax

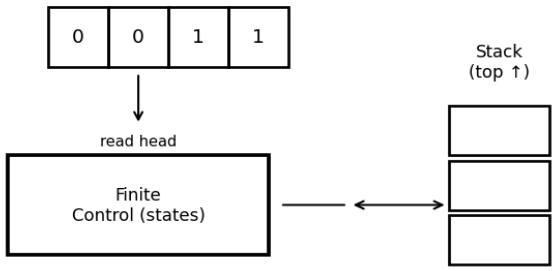
CNF conversion — the four standard steps

```
Example grammar converted step by step toward CNF:
S → ASA | aB          (not yet CNF — mixed terminals/non-terminals)
A → B | S
B → b | ε

After removing ε-productions, unit productions, and converting:
S → AB' | aB      where B' handles the remaining SA part, etc.
(Full conversion involves: (1) remove ε-productions, (2) remove unit
 productions A→B, (3) remove useless symbols, (4) convert remaining
 productions to the A→BC or A→a shape.)
```

Pushdown Automata (PDA)

Pushdown Automaton (PDA) — Structure



PDA = Finite Automaton + a Stack (unbounded memory) — power equal to Context-Free Grammars

PDA = (Q, Σ, Γ, δ, q_0, Z_0, F)

Symbol	Meaning
Γ	STACK alphabet
Z_0	Initial stack symbol
δ	Q × (Σ∪{ε}) × Γ → Q × Γ* (transition also depends on and updates the stack top)

Acceptance mode	Rule
Final state	PDA accepts if it reaches a final state (stack contents irrelevant)
Empty stack	PDA accepts if the stack becomes completely empty (any state)

Pushdown Automata ⇔Context-Free Grammars (EQUAL expressive power)

Worked example: PDA for $L = \{0^n1^n \mid n \geq 0\}$

Input symbol	Stack top	Action
0	Z_0 or 0	PUSH 0 onto the stack
1	0	POP one 0 from the stack
ε (end of input)	Z_0	ACCEPT (stack is back to just Z_0 — equal count of 0s and 1s)

Pumping Lemma for Context-Free Languages

For CFL L, ∃ pumping length p such that any string z ∈L, |z| ≥ p,
can be split as z = uvwxy where:
|vwx| ≤ p, |vx| ≥ 1 (v and x not both empty),
and $uv^nwx^ny \in L$ for ALL n ≥ 0

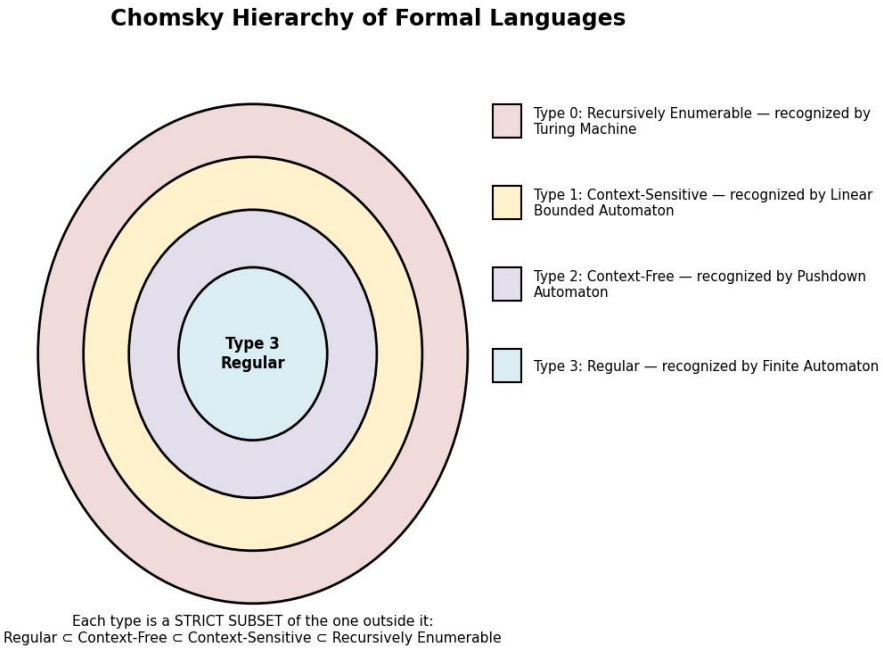
Used the same way as the regular pumping lemma: **only to prove a language is NOT context-free**

Classic example: L = {$a^nb^nc^n \mid n \geq 0$} is **NOT context-free** (proved via this pumping lemma — pumping v and x together cannot keep the counts of a, b, c all equal).

Properties of Context-Free Languages

Operation	Closed?
Union	YES
Concatenation	YES
Kleene star	YES
Intersection	NO (in general)
Complement	NO (in general)
Intersection with a REGULAR language	YES

Chomsky Hierarchy — The Complete Picture



Type	Language class	Grammar restriction	Recognizing automaton
Type 0	Recursively Enumerable	No restriction (α → β, α≠ε)	`Turing Machine`
Type 1	Context-Sensitive	|α| ≤ |β| (non-contracting)	`Linear Bounded Automaton`
Type 2	Context-Free	A → β (single non-terminal on left)	`Pushdown Automaton`
Type 3	Regular	A → a or A → aB (or A→ε)	`Finite Automaton`

Easy Explanation

The hierarchy is STRICTLY nested: Regular ⊂Context-Free ⊂Context-Sensitive ⊂Recursively Enumerable. Every regular language is context-free, but not every context-free language is regular (e.g. 0^n1^n). Automaton power matches exactly: FA < PDA < LBA < TM.

Probable MCQs: Context-Free Languages

1. A grammar is ambiguous if:

a) It has no productions

b) Some string has more than one parse tree

c) It uses only terminals

d) It cannot generate any string

✓Answer: b) Some string has more than one parse tree

Why: Ambiguity means at least one derivable string has multiple distinct parse trees.

2. In Chomsky Normal Form, a production must be of the form:

a) A→BCD

b) A→BC or A→a

c) A→aB only

d) A→ε always

✓Answer: b) A→BC or A→a

Why: CNF restricts productions to exactly two non-terminals or a single terminal.

3. In Greibach Normal Form, every production must start with:

a) A non-terminal

b) Exactly one terminal

c) The start symbol
d) Nothing specific
✓Answer: b) Exactly one terminal
Why: GNF productions have the shape A → aα, always beginning with one terminal.

4. A PDA differs from an FA by having an additional:
a) Extra tape
b) A stack
c) Multiple heads
d) A random generator
✓Answer: b) A stack
Why: The stack gives the PDA extra memory, enabling it to recognize context-free languages.

5. The language $\{a^n b^n c^n\}$ is:
a) Regular
b) Context-free but not regular
c) NOT context-free
d) Recognized by a DFA
✓Answer: c) NOT context-free
Why: This language requires matching THREE counts simultaneously, which a PDA's single stack cannot enforce.

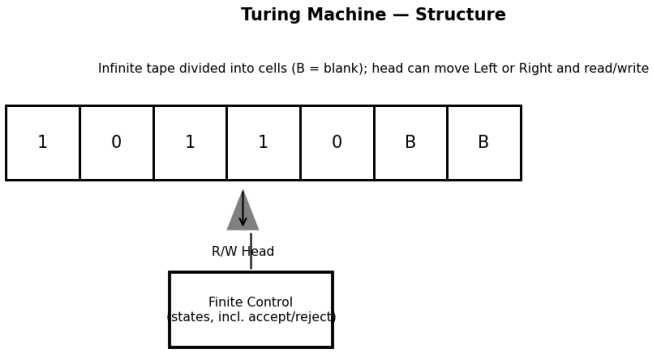
6. Context-free languages are NOT closed under:
a) Union
b) Concatenation
c) Kleene star
d) Intersection (in general)
✓Answer: d) Intersection (in general)
Why: Unlike regular languages, CFLs are not closed under intersection or complement in general.

7. In the Chomsky hierarchy, Type 1 languages are recognized by a:
a) Finite Automaton
b) Pushdown Automaton
c) Linear Bounded Automaton
d) Turing Machine only
✓Answer: c) Linear Bounded Automaton
Why: Context-sensitive (Type 1) languages correspond to Linear Bounded Automata.

8. BNF is commonly used to:
a) Optimize machine code
b) Define programming language syntax
c) Encrypt data
d) Design digital circuits
✓Answer: b) Define programming language syntax
Why: BNF is the standard notation style for writing CFGs that specify language grammar.

6.3 Turing Machine

Turing Machine — Formal Definition



TM = (Q, Σ, Γ, δ, q_0, B, F)

Symbol	Meaning
Γ	Complete TAPE alphabet (includes Σ and the blank symbol)
B	BLANK symbol (B ∈ Γ, B ∉ Σ)
δ	Q × Γ → Q × Γ × {L, R} (read symbol, write symbol, move Left/Right)

TM role	Description
Language Recognizer	Accepts a string by HALTING in an accepting state; may reject by halting in a non-accepting state or looping forever
Computing Function	Computes an OUTPUT value left on the tape after halting — models any computable function
Enumerator	Systematically GENERATES/lists every string of a language, one at a time, without necessarily halting

Turing Machine Variants — All Equivalent in Power

Variant	Structure	Power compared to standard TM	Why used
Multi-track TM	One tape divided into several parallel tracks	EQUAL power	Convenience — e.g. marking positions without a second tape
Multi-tape TM	Several independent tapes, each with its own head	EQUAL power (simulated by 1 tape, just slower)	Faster / more natural algorithm design
Non-Deterministic TM (NDTM)	Multiple possible transitions allowed	EQUAL power (any NDTM has an equivalent DTM)	Simplifies certain algorithm descriptions; may differ in TIME complexity
2-way infinite tape TM	Tape extends infinitely in both directions	EQUAL power	More natural for some problems

Easy Explanation

A recurring exam theme: multi-tape, multi-track, and non-deterministic TMs are all EQUIVALENT IN COMPUTATIONAL POWER to the basic single-tape deterministic TM. None of these variants can compute anything a basic TM cannot — they only affect convenience or (for NDTM) time complexity, never what is ultimately computable.

Church-Turing Thesis and Universal Turing Machine

Concept	Description
Church-Turing Thesis	Any function that is intuitively/algorithmically computable CAN be computed by some Turing Machine. It is a THESIS (a claim about the world), not a mathematically provable theorem
Universal Turing Machine (UTM)	A single TM that can SIMULATE any other TM, given that TM's own description plus its input as its own input tape content — the theoretical basis of the modern stored-program computer

Decidability

Category	Definition	Example
Decidable (Recursive)	A TM exists that ALWAYS halts, accepting or rejecting correctly	Checking if a string matches a regular expression
Undecidable	NO algorithm can solve it for all cases (proven impossible)	The HALTING PROBLEM — does program P halt on input I?
Recursively Enumerable (Semi-decidable)	A TM halts and accepts if the string IS in the language, but may loop forever if it is NOT	Testing general TM language membership

The Halting Problem is the most famous undecidable problem: it is **proven impossible** to write a general algorithm that determines whether an arbitrary program halts on an arbitrary input, for ALL possible programs and inputs.

Computational Complexity

Complexity measure	Definition
Time complexity	Number of steps as a function of input size n — e.g. $O(n)$, $O(n^2)$, $O(2^n)$
Space complexity	Amount of tape/memory used as a function of n

P = problems SOLVABLE in polynomial time

NP = problems whose solution can be VERIFIED in polynomial time

P ⊆NP (whether P = NP is a famous open problem — Millennium Prize)

Class	Meaning	Example problems
P	Solvable efficiently (polynomial time)	Sorting, shortest path, linear programming

Complexity measure	Definition	
NP	Solution VERIFIABLE efficiently, not necessarily solvable efficiently	SAT, Traveling Salesman (decision version), Subset Sum
NP-Complete	The HARDEST problems in NP — every NP problem reduces to it	SAT (the first proven NP-Complete problem, by Cook's theorem), 3-SAT, Clique
NP-Hard	At least as hard as NP-Complete, but not necessarily IN NP itself	Halting Problem (also undecidable), optimization version of TSP

- **Intractability:** problems solvable only in **exponential (non-polynomial) time** are considered intractable in practice for large inputs — most **NP-hard** problems fall here.
- **Reducibility:** transforming problem A into problem B in polynomial time — if B is solvable/decidable then so is A. Used to **prove NP-completeness** (reduce a KNOWN NP-complete problem to the new one) and to **prove undecidability** (reduce the Halting Problem to the new problem).

Easy Explanation

Remember the automaton-language-complexity ladder together: Finite Automaton ↔ Regular ↔ solvable in constant/linear space; Pushdown Automaton ↔ Context-Free; Linear Bounded Automaton ↔ Context-Sensitive ↔ PSPACE; Turing Machine ↔ Recursively Enumerable ↔ potentially undecidable. Each step up trades more computational power for weaker guarantees about halting/efficiency.

Probable MCQs: Turing Machines and Complexity

1. A language is accepted by a Turing Machine when the machine:

a) Never halts
b) Halts in an accepting state
c) Runs out of tape
d) Reaches the start state again
✓Answer: b) Halts in an accepting state
Why: TM acceptance is defined by halting in a designated accepting state.

2. A Non-Deterministic Turing Machine (NDTM) is:

a) More powerful than a Deterministic TM
b) Equal in power to a Deterministic TM
c) Less powerful than a finite automaton
d) Unable to simulate any computation
✓Answer: b) Equal in power to a Deterministic TM
Why: NDTMs can always be simulated by DTMs — same computational power, possibly different time complexity.

3. The Church-Turing Thesis is:

a) A mathematically proven theorem
b) A thesis defining 'computable' in terms of Turing Machines

c) A proof that P=NP

d) An algorithm for the halting problem

✓Answer: b) A thesis defining 'computable' in terms of Turing Machines

Why: It is a widely accepted claim, not a provable theorem, about what 'computable' means.

4. The Halting Problem is an example of a problem that is:

a) Decidable

b) Undecidable

c) NP-Complete only

d) Solvable in polynomial time

✓Answer: b) Undecidable

Why: It is proven that no general algorithm can decide halting for all program-input pairs.

5. A Universal Turing Machine can:

a) Solve only one specific problem

b) Simulate any other Turing Machine given its description

c) Run faster than all other TMs

d) Avoid the halting problem entirely

✓Answer: b) Simulate any other Turing Machine given its description

Why: A UTM takes a TM's description + input and simulates that TM's behavior.

6. NP-Complete problems are:

a) The easiest problems in NP

b) The hardest problems in NP, to which all NP problems reduce

c) Always undecidable

d) Only theoretical, with no known examples

✓Answer: b) The hardest problems in NP, to which all NP problems reduce

Why: By definition, every NP problem polynomial-time reduces to an NP-Complete problem.

7. Reducibility is a technique mainly used to prove:

a) That a problem is efficient

b) Undecidability or NP-completeness of a problem

c) Regularity of a language

d) That P = NP conclusively

✓Answer: b) Undecidability or NP-completeness of a problem

Why: Reducing a known hard/undecidable problem to a new one shows the new one is at least as hard.

8. SAT was the first problem proven NP-Complete, via:

a) The Pumping Lemma

b) Cook's theorem (Cook-Levin theorem)

c) The Church-Turing Thesis

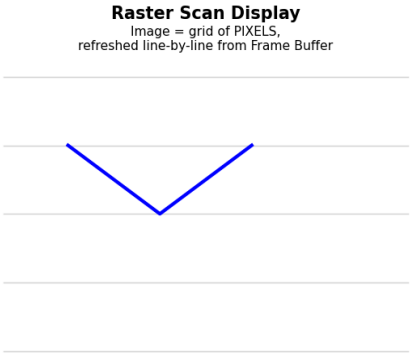
d) Myhill-Nerode theorem

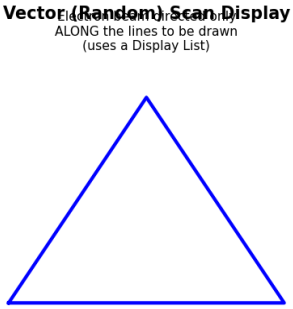
✓Answer: b) Cook's theorem (Cook-Levin theorem)

Why: Cook's theorem (1971) established SAT as NP-Complete, founding NP-completeness theory.

6.4 Introduction to Computer Graphics

Graphics Display Hardware





Feature	Raster Scan	Vector (Random) Scan
Image built from	A GRID of pixels (frame buffer)	LINE segments only (display list)
Refresh method	Entire screen redrawn line by line, top to bottom	Beam directed only along the lines to draw
Realism / shading	Can display realistic, filled, shaded images	Only outline/wireframe drawings
Line quality	`Jagged edges possible (aliasing)`	Very smooth, sharp lines
Memory required	Frame buffer proportional to resolution	Display list proportional to number of lines
Modern usage	STANDARD — all modern monitors/TVs	Obsolete (old oscilloscopes, early CAD/radar)

Display Technologies

Technology	Principle	Note
CRT (Cathode Ray Tube)	`Electron beam excites phosphor dots on a screen`	Older, bulky, largely obsolete
LCD (Liquid Crystal Display)	`Liquid crystals block/pass backlight per pixel`	Thin, energy-efficient, common today
LED / OLED	`Light-emitting diodes/organic LEDs emit light directly`	OLED: no backlight needed, true blacks
Plasma display	`Ionized gas cells emit light`	Mostly obsolete for consumer use

Component	Role
Frame buffer	Memory area holding the color/intensity value of EVERY pixel to be displayed
Display processor / Graphics controller	Dedicated hardware that offloads rendering computation from the main CPU
Video controller	Reads the frame buffer and generates the actual video signal, refreshing the screen (typically 60+ times/second)

Frame buffer size (bytes) = Horizontal resolution × Vertical resolution × (bits per pixel / 8)

Example: a 1920×1080 display at 24-bit color needs `1920 × 1080 × 3 bytes ≈ 5.93 MB` of frame buffer memory.

Input and Output Devices in Graphics Systems

Category	Devices
Input (pointing/locator)	`Mouse, trackball, joystick, light pen, touch screen, digitizer/graphics tablet`
Input (image capture)	`Scanner, digital camera`
Output	`Monitor (CRT/LCD/LED), printer, plotter, projector`

Graphics Software and Standards

Standard / Library	Purpose
OpenGL	Open, cross-platform, cross-language graphics API
DirectX (Direct3D)	Microsoft's graphics API, mainly for Windows/gaming
GKS (Graphical Kernel System)	One of the earliest standardized 2D graphics APIs
PHIGS	Programmer's Hierarchical Interactive Graphics System — 3D standard
Vulkan	Modern low-overhead, cross-platform graphics/compute API

Why standards matter: they ensure **portability** — graphics code written against a standard API works across different hardware/vendors without rewriting.

⍰ Probable MCQs: Computer Graphics Overview

1. A raster scan display builds an image using:

a) A display list of lines only
b) A grid of pixels stored in a frame buffer
c) Vector equations only
d) No memory at all
✓Answer: b) A grid of pixels stored in a frame buffer
Why: Raster displays refresh from a frame buffer holding pixel-by-pixel values.

2. Vector (random scan) displays direct the electron beam:

a) Across every pixel row by row
b) Only along the lines that need to be drawn
c) Randomly with no pattern
d) Only in a circular pattern
✓Answer: b) Only along the lines that need to be drawn
Why: Vector systems trace only the required line segments, guided by a display list.

3. The frame buffer size for a 1024×768 display at 8 bits per pixel is:

a) 768 KB
b) 786,432 bytes ≈ 768 KB
c) 1024 bytes
d) 8 MB
✓Answer: b) 786,432 bytes ≈ 768 KB
Why: 1024 × 768 × (8/8) = 786,432 bytes.

4. OLED displays differ from LCD mainly because OLED:

a) Requires a backlight
b) Emits light directly from each pixel, no backlight needed
c) Uses electron beams
d) Cannot display color
✓Answer: b) Emits light directly from each pixel, no backlight needed
Why: OLED pixels are self-emissive, unlike LCD which blocks/passes a separate backlight.

5. A light pen is classified as a/an:

a) Output device
b) Input (pointing) device
c) Storage device
d) Display technology
✓Answer: b) Input (pointing) device
Why: A light pen is used to point at or draw on the screen — an input device.

6. OpenGL and DirectX are examples of:

a) Input devices

b) Graphics software APIs/standards

c) Display processors

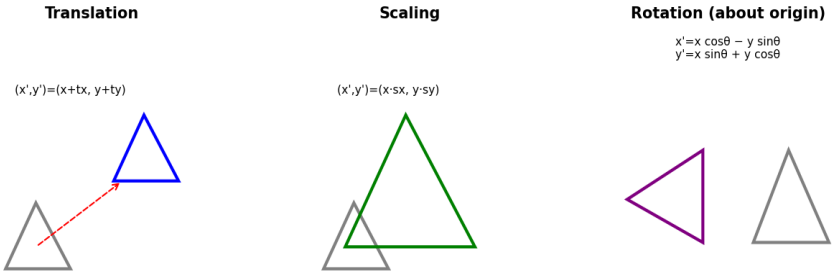
d) Storage formats

✓Answer: b) Graphics software APIs/standards

Why: They are standardized programming interfaces for graphics rendering.

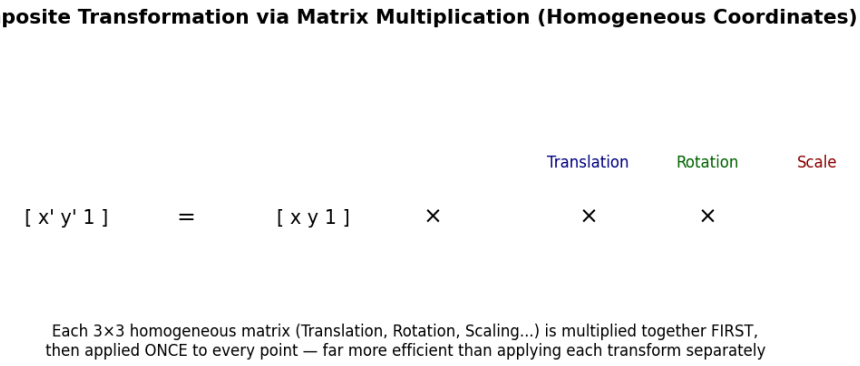
6.5 Two-Dimensional Transformation

Basic 2D Transformations — Formulas Table



Transformation	Formula	Matrix form (homogeneous)
Translation	x'=x+tx, y'=y+ty	`[1 0 tx; 0 1 ty; 0 0 1]`
Scaling	x'=x·sx, y'=y·sy	`[sx 0 0; 0 sy 0; 0 0 1]`
Rotation (about origin, angle θ)	x'=x cosθ − y sinθ, y'=x sinθ + y cosθ	`[cosθ −sinθ 0; sinθ cosθ 0; 0 0 1]`
Reflection about x-axis	x'=x, y'=−y	`[1 0 0; 0 −1 0; 0 0 1]`
Reflection about y-axis	x'=−x, y'=y	`[−1 0 0; 0 1 0; 0 0 1]`
Reflection about origin	x'=−x, y'=−y	`[−1 0 0; 0 −1 0; 0 0 1]`
Shear (x-direction)	x'=x+sh·y, y'=y	`[1 sh 0; 0 1 0; 0 0 1]`
Shear (y-direction)	x'=x, y'=y+sh·x	`[1 0 0; sh 1 0; 0 0 1]`

Homogeneous Coordinates and Composite Transformation



A 2D point `(x,y)` is represented as `(x,y,1)` in homogeneous coordinates. This trick lets **translation** (normally an addition) be expressed as **matrix multiplication** — so ALL transformations, including translation, can be combined into ONE matrix via multiplication.

Property	Detail
Combining N transformations	Multiply their N matrices together FIRST, then apply the single result matrix to every point — far more efficient than applying each separately
Order MATTERS	Matrix multiplication is NOT commutative — rotate-then-translate ≠ translate-then-rotate
Rotation/Scaling about an arbitrary pivot point	(1) Translate the pivot to the origin, (2) Rotate/Scale, (3) Translate back — a 3-matrix composite

Worked example: rotate a point 90° about the origin, then translate by (2,3). Point (1,0):

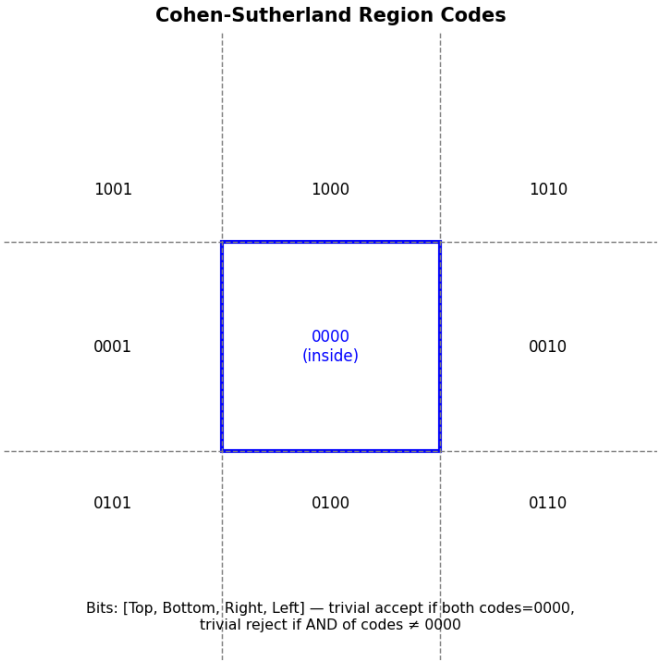
Step	Calculation	Result
1. Rotate 90°	x'=1×cos90−0×sin90=0, y'=1×sin90+0×cos90=1	`(0, 1)`
2. Translate (+2,+3)	x''=0+2, y''=1+3	`(2, 4)`

2D Viewing Pipeline

World Coordinates → Window (Clipping) → Viewport → Device/Screen Coordinates

Term	Meaning
Window	A rectangular region of the WORLD/scene selected for display
Viewport	A rectangular region of the SCREEN/device where the window's contents are mapped
Window-to-Viewport mapping	Scales and translates world coordinates into device coordinates, preserving relative position

Line Clipping Algorithms



Region code bit	Set to 1 when the point is...
`Bit 1 (leftmost)`	ABOVE the window (Top)
`Bit 2`	BELOW the window (Bottom)
`Bit 3`	RIGHT of the window
`Bit 4 (rightmost)`	LEFT of the window

Cohen-Sutherland test	Condition	Action
Trivial ACCEPT	Both endpoint codes = 0000	Draw the line as-is — entirely inside
Trivial REJECT	Bitwise AND of both codes ≠ 0000	Discard the line — entirely on one outside side
Needs clipping	Neither of the above	Clip against each boundary the line crosses, then re-test

Feature	Cohen-Sutherland	Liang-Barsky
Approach	Region codes + boundary-by-boundary clipping	Parametric line equation, solves inequalities for t∈[0,1]
Efficiency	Good, but can require multiple clip iterations	GENERALLY MORE EFFICIENT — fewer intersection calculations
Works for	Rectangular clip windows	Rectangular clip windows (extendable to convex polygons)

Easy Explanation

Cohen-Sutherland region code bit order to memorize: Top-Bottom-Right-Left (TBRL). Example: a point above and to the right of the window gets code 1010. Trivial accept needs BOTH codes = 0000; trivial reject needs the bitwise AND ≠ 0000 (meaning both points violate the SAME boundary).

Polygon Clipping

Algorithm	Approach
Sutherland-Hodgman	`Clips the polygon against ONE window edge at a time, processing all vertices, producing a new polygon each pass – repeated for all 4 edges`

Probable MCQs: 2D Transformation and Clipping

1. The transformation equation x'=x+tx, y'=y+ty represents:

a) Rotation

b) Translation

c) Scaling

d) Shearing

✓Answer: b) Translation

Why: This equation shifts a point by (tx, ty) — the definition of translation.

2. Homogeneous coordinates represent a 2D point as:

a) (x,y)

b) (x,y,0)

c) (x,y,1)

d) (1,x,y)

✓Answer: c) (x,y,1)

Why: Appending a 1 allows translation to be expressed as matrix multiplication.

3. To rotate a shape about an arbitrary pivot point (not the origin), the correct sequence is:

a) Rotate only

b) Translate pivot to origin → rotate → translate back

c) Scale then rotate

d) Reflect then rotate

✓Answer: b) Translate pivot to origin → rotate → translate back

Why: Rotation formulas are defined about the origin, so the pivot must first be moved there.

4. In the Cohen-Sutherland algorithm, a line is trivially ACCEPTED when:

a) Both endpoint codes are non-zero

b) Both endpoint codes are 0000

c) The AND of both codes is non-zero

d) Only one endpoint has code 0000

✓Answer: b) Both endpoint codes are 0000

Why: Code 0000 for both endpoints means both lie inside the window — trivial accept.

5. In Cohen-Sutherland, a line is trivially REJECTED when:

a) Both codes are 0000

b) The bitwise AND of both codes is non-zero

c) The bitwise OR of both codes is zero

d) Neither endpoint has a code

✓Answer: b) The bitwise AND of both codes is non-zero

Why: A non-zero AND means both endpoints violate the SAME boundary, so the line is entirely outside.

6. Compared to Cohen-Sutherland, the Liang-Barsky algorithm is generally:

a) Less efficient

b) More efficient (parametric approach)

c) Unusable for line clipping

d) Only for 3D clipping

✓Answer: b) More efficient (parametric approach)

Why: Liang-Barsky's parametric method typically needs fewer computations.

7. The viewport in the 2D viewing pipeline refers to:

a) The region of world coordinates selected for viewing

b) The region of the screen/device where content is displayed

c) A type of clipping algorithm

d) A transformation matrix

✓Answer: b) The region of the screen/device where content is displayed

Why: The viewport is the screen-space area onto which the window's contents are mapped.

8. The Sutherland-Hodgman algorithm is used for:

a) Line clipping only

b) Polygon clipping

c) 3D projection

d) Text rendering

✓Answer: b) Polygon clipping

Why: It clips a polygon successively against each edge of the clip window.

6.6 Three-Dimensional Transformation

Basic 3D Transformations — Formulas Table

Transformation	Formula
Translation	x'=x+tx, y'=y+ty, z'=z+tz
Scaling	x'=x·sx, y'=y·sy, z'=z·sz
Rotation about X-axis (Rx)	y'=y cosθ−z sinθ, z'=y sinθ+z cosθ, x'=x
Rotation about Y-axis (Ry)	z'=z cosθ−x sinθ, x'=z sinθ+x cosθ, y'=y
Rotation about Z-axis (Rz)	x'=x cosθ−y sinθ, y'=x sinθ+y cosθ, z'=z

Key difference from 2D: 3D rotation needs **THREE separate matrices** (Rx, Ry, Rz) — one per axis — since rotation is now about a LINE (an axis) rather than just a point.

Feature	2D	3D
Coordinates	(x,y)	(x,y,z)
Homogeneous form	(x,y,1) — 3 values	(x,y,z,1) — 4 values
Transformation matrix size	3×3	4×4
Rotation axes	1 (about a point, the origin)	3 independent axes (X, Y, Z)
Reflection planes	About a line (x-axis, y-axis)	About a PLANE (xy-plane, yz-plane, xz-plane)

3D Viewing Pipeline

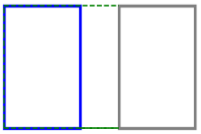
World Coordinates → Viewing Coordinates → Projection → Normalized Device Coordinates → Screen Coordinates

Stage	Purpose
Viewing transformation	Converts world coordinates into the camera/eye's own coordinate system
Projection	Converts 3D viewing coordinates into a 2D representation (see below)
Clipping	Removes parts of the scene outside the view volume
Viewport transformation	Maps normalized coordinates onto actual screen pixels

Projections — Complete Comparison

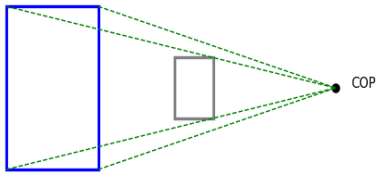
Parallel (Orthographic) Projection

Projection lines are PARALLEL —
object size unchanged regardless of distance



Perspective Projection

Projection lines CONVERGE to a
Center of Projection — distant objects appear smaller



Feature	Parallel Projection	Perspective Projection
Projection lines	PARALLEL to each other	CONVERGE at a Center of Projection (COP)
Center of Projection	At infinity (conceptually)	A finite point
Object size vs distance	UNCHANGED regardless of distance — preserves true relative dimensions	Distant objects appear SMALLER (foreshortening) — realistic
Realism	Less realistic, but geometrically accurate	Highly realistic, mimics human vision/camera
Use case	Engineering/CAD drawings, blueprints (exact measurements needed)	Games, movies, simulations, realistic rendering

Sub-type	Parent category	Description
Orthographic	`Parallel`	Projection lines PERPENDICULAR to the viewing plane — true shape and size (front/top/side views)
Oblique	`Parallel`	Projection lines at an ANGLE (not perpendicular) to the viewing plane — shows some depth (e.g. Cavalier, Cabinet projection)
One-point perspective	`Perspective`	ONE vanishing point — used when one axis (usually depth) recedes
Two-point perspective	`Perspective`	TWO vanishing points — common in architectural drawing
Three-point perspective	`Perspective`	THREE vanishing points — most realistic, used for dramatic views (looking up/down at tall objects)

Oblique sub-type	Depth line scale	Effect
Cavalier projection	`FULL scale (1:1) along the depth axis`	Depth looks exaggerated/stretched
Cabinet projection	`HALF scale (1:2) along the depth axis`	More realistic-looking depth

? Easy Explanation

Simple memory trick: Parallel projection = 'engineering/technical' view (keeps true measurements, used in blueprints — think CAD software); Perspective projection = 'photograph/eye' view (mimics how human vision makes far things look smaller — think video games and movies). The number of vanishing points in perspective (1, 2, or 3) equals the number of principal axes that are NOT parallel to the viewing plane.

? Probable MCQs: 3D Transformation and Projection

1. In 3D graphics, rotation can occur about how many independent axes?

a) 1
b) 2
c) 3
d) 4
✓Answer: c) 3

Why: 3D rotation can occur about the x, y, and z axes independently (Rx, Ry, Rz).

2. 3D composite transformations are typically represented using matrices of size:

a) 2×2
b) 3×3
c) 4×4 (homogeneous coordinates)
d) 5×5
✓Answer: c) 4×4 (homogeneous coordinates)
Why: 3D graphics use 4×4 homogeneous transformation matrices.

3. In parallel (orthographic) projection, an object's apparent size:

a) Shrinks with distance from viewer
b) Stays the same regardless of distance
c) Randomly changes
d) Only changes for certain colors
✓Answer: b) Stays the same regardless of distance
Why: Parallel projection lines never converge, so size is preserved regardless of depth.

4. Which projection technique is most commonly used for realistic, photograph-like rendering?

a) Orthographic projection
b) Oblique projection
c) Perspective projection
d) Isometric projection only
✓Answer: c) Perspective projection
Why: Perspective projection mimics human vision, making distant objects appear smaller.

5. Oblique projection is a sub-type of which projection category?

a) Perspective projection
b) Parallel projection
c) Isometric projection only
d) None of these
✓Answer: b) Parallel projection
Why: Oblique projection is parallel, but with projection lines at an angle rather than perpendicular.

6. Cabinet projection uses what scale along the depth axis compared to Cavalier?

a) Same full scale
b) Half scale
c) Double scale
d) Zero scale
✓Answer: b) Half scale
Why: Cabinet projection halves the depth axis scale for a more realistic appearance.

7. A two-point perspective projection has how many vanishing points?

a) 1
b) 2
c) 3
d) 0
✓Answer: b) 2
Why: Two-point perspective uses two vanishing points, common in architectural drawings.

8. Reflection in 3D graphics is performed about a:

a) Point only

b) Line only

c) Plane

d) Vector

✓Answer: c) Plane

Why: Unlike 2D reflection about a line, 3D reflection is about a plane (e.g. the xy-plane).