ACOL216: Tutorial-1

February 2 (G1) and February 5 (G2), 2026

Turing Machines

Definition (Deterministic Turing Machine). A deterministic Turing Machine (TM) is formally defined as a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$$

where:

- Q is a finite set of states,
- Σ is the input alphabet,
- Γ is the tape alphabet with $\Sigma \subseteq \Gamma$,
- \$ is a distinguished delimiter symbol in Γ,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_{\text{acc}}, q_{\text{rej}} \in Q$ are halting states.

Meaning of the transition function. For a given current state $q \in Q$ and tape symbol $\gamma \in \Gamma$,

$$\delta(q, \gamma) = (q', \gamma', D)$$

means:

(i) overwrite γ with γ' on the tape,

(ii) update the state register to q',

(iii) move the tape head one cell in direction $D \in \{L, R\}$.

Operational interpretation. The Turing Machine can be viewed as consisting of:

(i) an infinite tape (memory),

(ii) a tape head (address pointer),

(iii) a state register,

(iv) an action table implementing δ.

At each step, the machine:

1. reads the current tape symbol,
2. reads the current state from the state register,
3. consults the action table,
4. writes a new symbol,
5. updates the state,
6. moves the head left or right.

Key restriction. Only *local* access is allowed: the head moves one cell at a time, and all computation proceeds sequentially.

Universality and Instruction Set Completeness

A Turing Machine is a **universal machine**: it can simulate any algorithmic computation.

Church-Turing Thesis. Any computation that can be performed by a physically realizable computer can be computed by a Turing Machine.

Architectural implication. If a processor's instruction set can simulate: (i) reading and writing memory, (ii) conditional state transitions, and (iii) unbounded repetition, then it is **computationally complete**. This formally justifies why modern instruction sets rely on a small set of primitive operations (arithmetic, control flow, memory access).

From Turing Machines to Bits

Although Turing Machines are universal, they are inefficient. Real computers optimize computation by encoding information as finite bit strings, performing arithmetic operations using fixed-width binary representations, and executing additions using carry-based logic. Thus, to understand real machines, we must understand how computation behaves under *finite precision.*

Modern AI systems (e.g., neural networks) execute billions of operations of the form:

$$y = \sum_{i=1}^{n} x_i w_i$$

At the lowest level: numbers are stored as *finite bit strings*, additions rely on *binary carry propagation*, and overflow is a fundamental limitation. *Training and inference in AI systems are thus constrained not by mathematics, but by bit-width, carry propagation, and overflow – exactly the concepts introduced in these lectures.*

Turing Machine Problems

Problem TM.1: Incrementing a Number

Design a deterministic Turing Machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathbf{acc}}, q_{\mathbf{rej}})$$

that increments a non-negative integer by 1**.**

Input format.

- The input number is written in *binary*.
- The tape contains $\$\, b_{k-1}b_{k-2}\cdots b_0\, \$$ where $b_i \in \{0,1\}$, where b_0 is the least significant bit (LSB).
- The tape head initially points to the LSB b_0.

Output. After halting, the tape should contain the binary representation of the input number plus one.

Problem TM.2: Checking for a Palindrome

Design a deterministic Turing Machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathbf{acc}}, q_{\mathbf{rej}})$$

that decides whether a string over $\{a,b\}$ is a palindrome.

Input format.

- Tape contains $\$\, w\, \$$ where $w \in \{a,b\}^*$.
- Head initially points to the rightmost symbol.

Output. Enter q_{acc} iff w is a palindrome.

Passage 1: Binary Addition as a Computational Primitive

A binary adder computes the sum of two n-bit numbers starting from the least significant bit (LSB). At each bit position, the output depends on:

- the two input bits,
- the carry from the previous position.

In the worst case, adding 1 to an n-bit number consisting entirely of 1s produces a carry that propagates across all n bits.

Questions:

P1.1 Let A and B be two n-bit unsigned integers. Derive the maximum possible value of $A+B$ and determine how many bits are required to represent the sum without overflow.

P1.2 Suppose a system performs repeated additions of K unsigned n-bit integers. Derive the minimum bit-width required to store the exact sum.

Passage 2: Finite Precision in AI Computation

In many AI accelerators, numbers are represented using fixed-width binary formats. Let an unsigned n-bit integer represent values in $[0, 2^n - 1]$.

A dot product accumulates multiple terms into a finite-width register. If the register width is insufficient, *overflow occurs*, silently corrupting the result.

Questions:

P2.1 Suppose each product term is guaranteed to be at most 2^p. If K such terms are summed, derive the minimum number of bits needed to store the exact sum.

P2.2 Assume a fixed accumulator of width m bits is used. Derive the maximum value of K such that summing K numbers each of value 2^p does not overflow.

P2.3 Conceptually explain why increasing numerical precision in AI systems is fundamentally a hardware resource tradeoff, using only bit-level reasoning.

Additional Problems

Q1. Consider the `sbn` instruction that subtracts the second operand from the first operand, and branches to the instruction specified by the label (third operand), if the result is negative. Write a small program using only the `sbn` instruction to compute the factorial of a positive number.

Q2. Show that any n-bit unsigned integer can be incremented using only bit flips and a carry signal.

Q3. Let S_n be the set of all n-bit binary strings. What is $|S_n|$?

Q4. How many distinct Boolean functions $f : \{0, 1\}^n \to \{0, 1\}$ exist?

Q5. Suppose a computation requires representing integers up to M. Derive the minimum number of bits required.

Q6. Derive the condition under which unsigned addition overflows.