Overview of CS 301:Theory of Computation

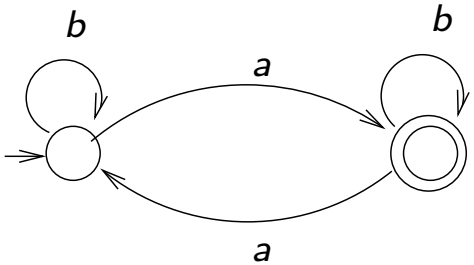
Sumesh Divakaran

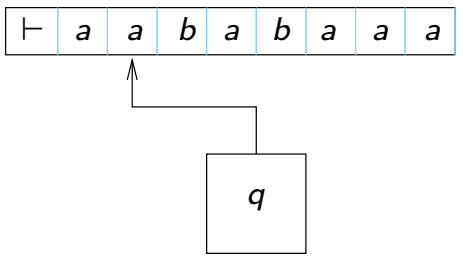
Department of Computer Science and Engineering
College of Engineering Trivandrum

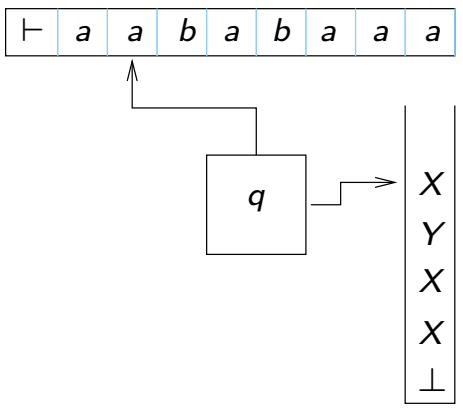
1^{st} August 2019

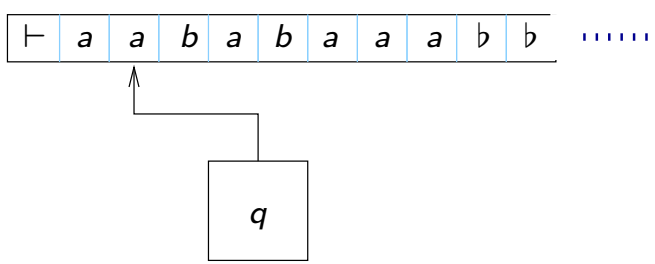
Outline

Different Kinds of "Automata" or "State Machines"









- Finite-State Automata
- Pushdown Automata
- Turing Machines

Why study automata theory?

Corner stone of many subjects in Computer Science:

1. Compilers
   - Lexical analysis, parsing, regular expression search
2. Digital circuits (state minimization, analysis)
3. Complexity Theory (algorithmic hardness of problems)
4. Mathematical Logic
   - Decision procedures for logical problems
5. Formal Verification
   - Is $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$

Uses in Verification

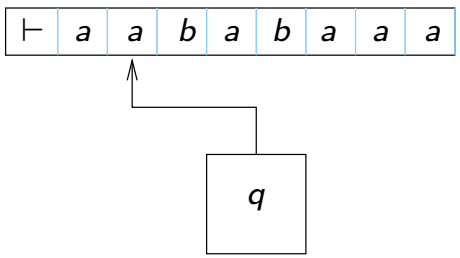
1. System models are natural extensions of automata models
   - Programs with no dynamic memory allocation, no procedures = Finite State systems
   - No dynamic memory allocation = Pushdown systems
   - General program = Turing machine
   - Programs with integer variables = Counter machines

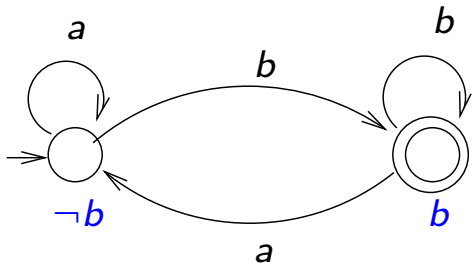
   Decision procedures for emptiness, configuration reachability, etc, directly translate to decision procedures for programs
2. To solve model-checking problem for logics that talk about infinite behaviour

Uses in Logic

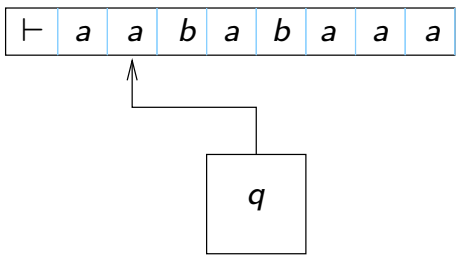
- Obtain decision procedure for satisfiability of a logic by translating a formula to an automaton and checking emptiness
- Argue undecidability/incompleteness of a proof system

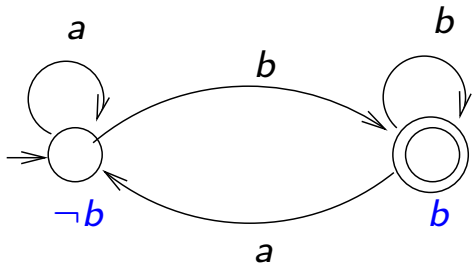
Finite State Automata





Finite State Automata



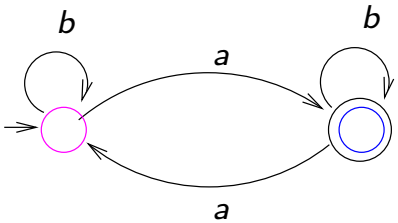


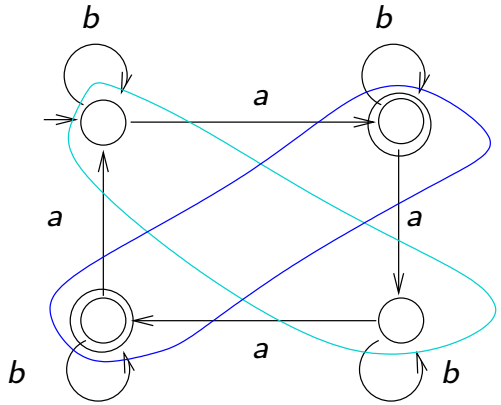
Language accepted

$$\mathcal{L} = \{x \cdot b \mid x \text{ is a string of } a \text{ and } b\}$$

Myhill-Nerode Theorem

Every regular language has a canonical DFA accepting it





Some consequences:

For a given regular language $\mathcal{L}$

1. Any DFA is a refinement of the canonical DFA for $\mathcal{L}$
2. "minimal" DFA's for $\mathcal{L}$ are isomorphic

Regular Grammar

$$S \rightarrow aS \tag{1}$$
$$S \rightarrow bS \tag{2}$$
$$S \rightarrow b \tag{3}$$

Regular Grammar

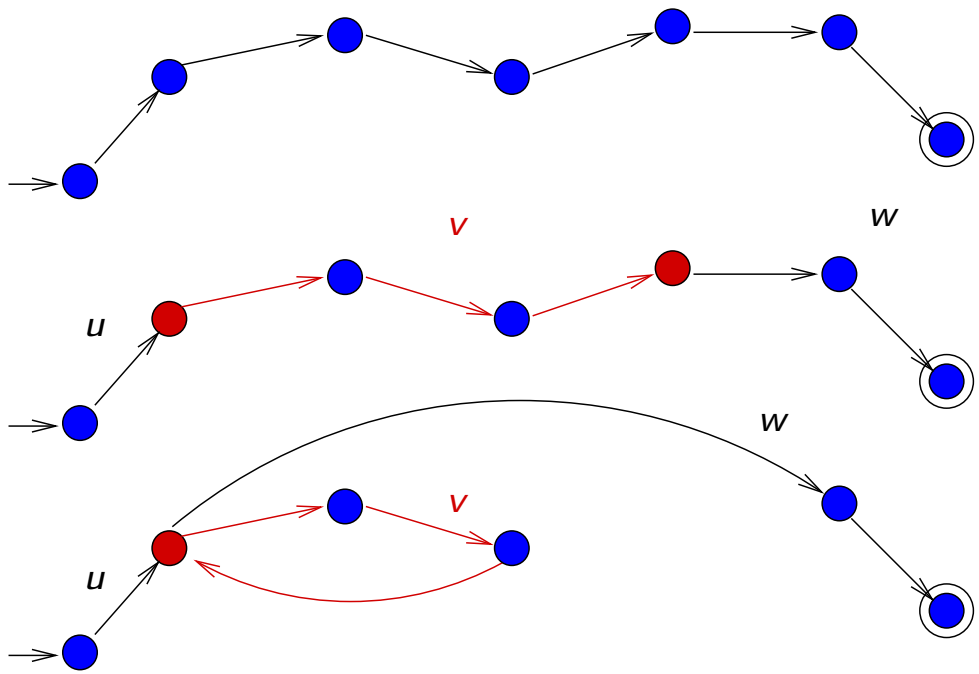
$$S \rightarrow aS \tag{1}$$
$$S \rightarrow bS \tag{2}$$
$$S \rightarrow b \tag{3}$$

Language accepted

$$\mathcal{L} = \{x \cdot b \mid x \text{ is a string of } a \text{ and } b\}$$

Pumping lemma for regular languages

Based on a simple observation: In a given DFA $\mathcal{A}$, if a path p in it is longer than the number of states in $\mathcal{A}$ then p must have a loop in it.



So if uvw is accepted along this path, then so is uw, uv^2w, ...
Can be used to prove that certain languages are non-regular (Ex. $\mathcal{L} = \{a^n \cdot b^n \mid n \geq 0\}$)

Context-Free Grammar

$$S \rightarrow aSb$$
$$S \rightarrow \epsilon$$

Context-Free Grammar

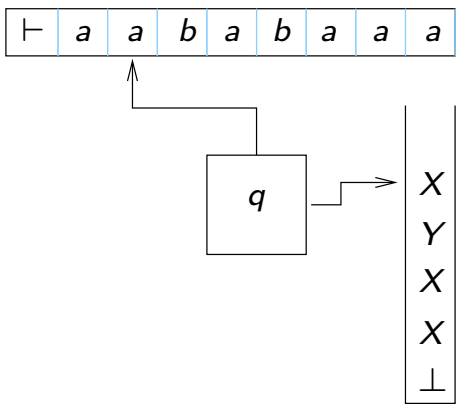
$$S \rightarrow aSb$$
$$S \rightarrow \epsilon$$

Language accepted

$$\mathcal{L} = \{a^n \cdot b^n \mid n \geq 0\}$$

which is known to be a Context-Free Language

Pushdown Automata



Can recognize $\mathcal{L} = \{a^n \cdot b^n \mid n \geq 0\}$

Context-Sensitive Grammar

$$S \rightarrow aSBc$$
$$S \rightarrow abc$$
$$cB \rightarrow Bc$$
$$bB \rightarrow bb$$

Context-Sensitive Grammar

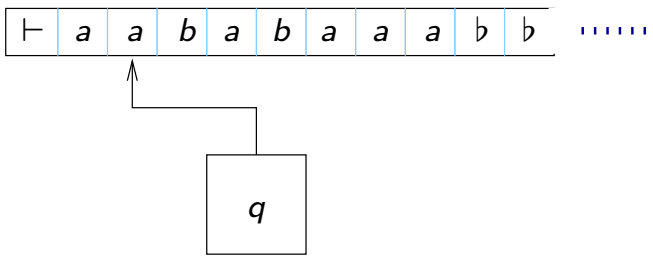
$$S \rightarrow aSBc$$
$$S \rightarrow abc$$
$$cB \rightarrow Bc$$
$$bB \rightarrow bb$$

Language accepted

$$\mathcal{L} = \{a^n \cdot b^n \cdot c^n \mid n \geq 0\}$$

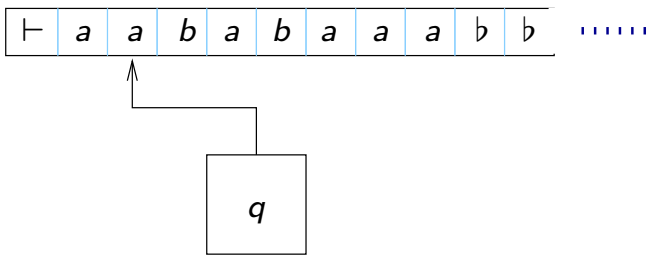
which is a non-Context-Free Language

Linear Bounded Automata



Tape length: limited by a linear function of the input length

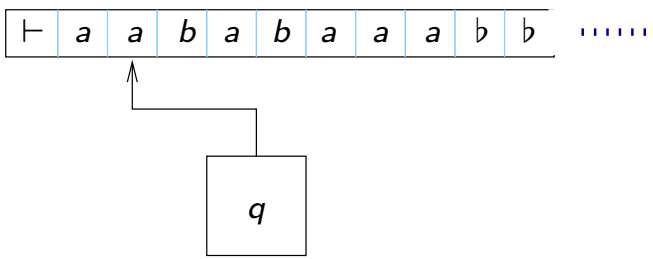
Linear Bounded Automata



Tape length: limited by a linear function of the input length

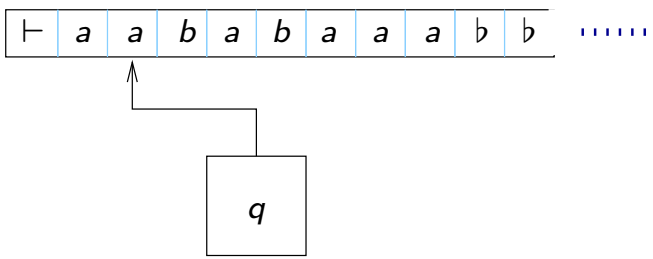
Can recognize $\mathcal{L} = \{a^n \cdot b^n \cdot c^n \mid n \geq 0\}$

Turing Machines



Tape length: unlimited

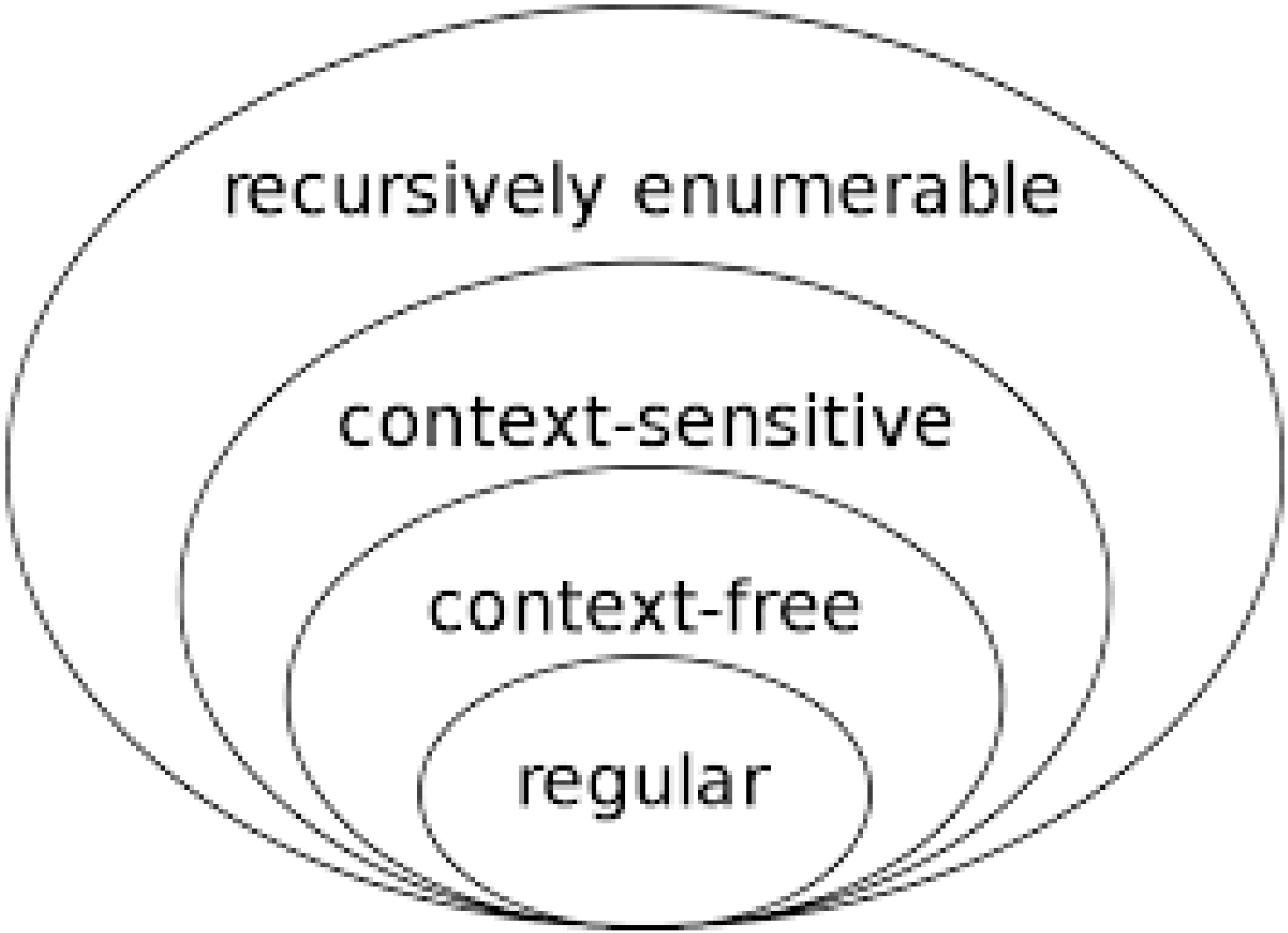
Turing Machines



Tape length: unlimited

It is equivalent in power to a digital computer

Chomsky Classification of Languages



Halting Problem

The problem of determining, from a description of an arbitrary Turing Machine M and an input x to M, whether M will finish its computation on x (i.e., halt) or continue to run on x forever.

Halting Problem

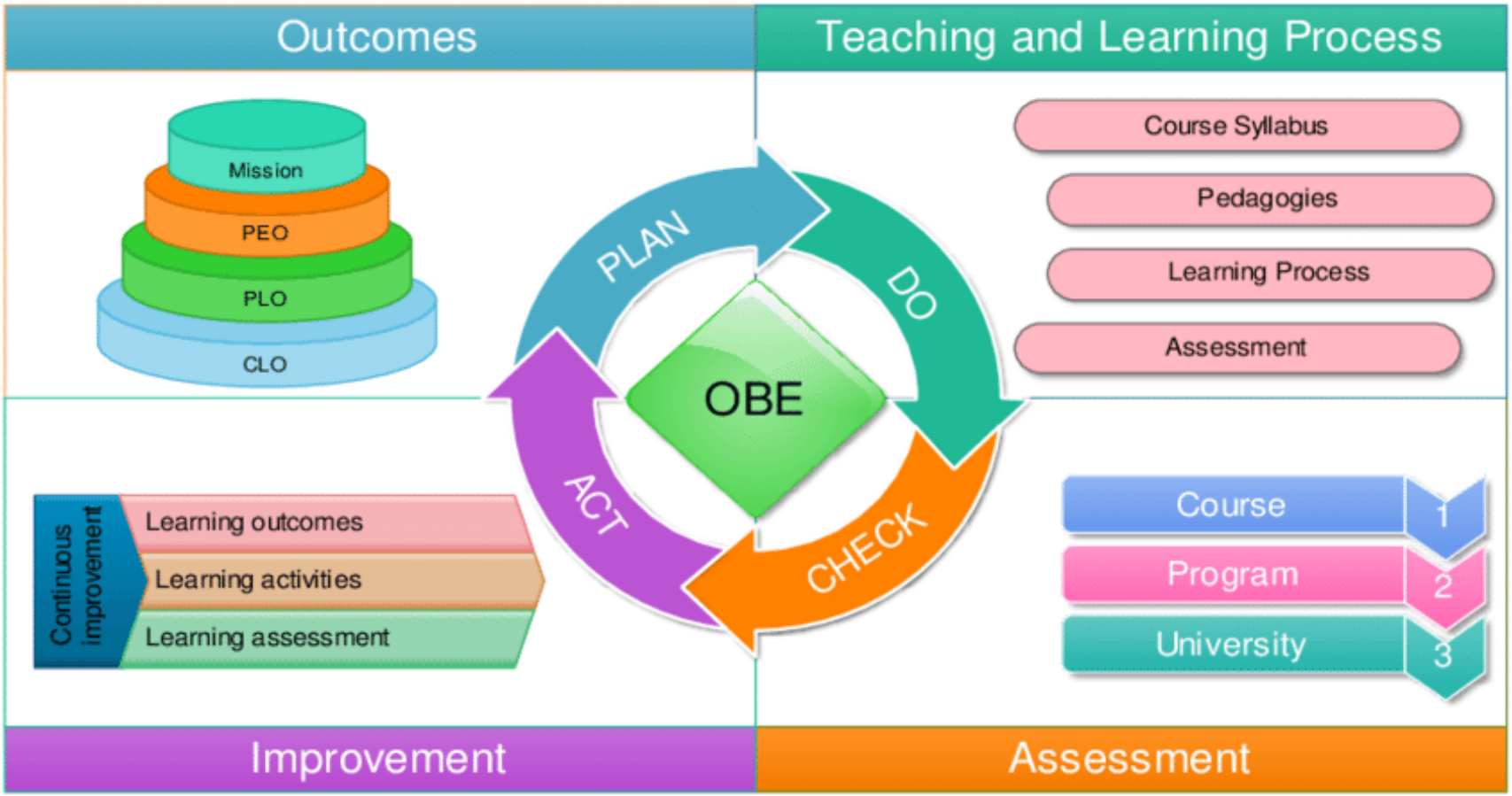
The problem of determining, from a description of an arbitrary Turing Machine M and an input x to M, whether M will finish its computation on x (i.e., halt) or continue to run on x forever.

It is proved by Alan Turing that this problem is undecidable

Course details

- Continuous Internal Evaluation: 50 marks
  - First Internal Exam - 20 Marks
  - Assignment - 10 Marks
  - Second Internal Exam - 20 Marks
- End Semester Examination: 100 Marks
- Eligibility Conditions for writing End Semester Examination
  1. Attendance - 75% or above
  2. Marks obtained for Continuous Internal Evaluation - 22.5 or above

Outcome Based Education



Outcome Based Education

- Student-centric Teaching and Learning Methodology
- Course Outcome:

  A statement which says an action that a student shall be able to perform at the end of learning the course
- Typically a course in BTech curriculum is expected to have 4 to 6 course outcomes

Curse Outcomes for CS 301 Theory of Computation

1. Classify formal languages into Regular, Context-Free, Context Sensitive and Unrestricted languages
2. Design Finite State Automata, Regular Grammar, Regular Expression and Myhill-Nerode Relation representations for regular languages
3. Design Pushdown Automata and Context-Free Grammar representations for Context-Free Languages
4. Design Turing Machines for accepting Recursively Enumerable Languages
5. Understand the notions of decidability and undecidability of problems