THEORY OF COMPUTATION

Noman amin
Lecturer, Dept. of CSE, Southeast University

Introduction

- The mathematical study of what problems can be solved by computers and how efficiently they can be solved.
- Core questions:
  - *Computability — Is there any algorithm that always decides the problem?*
  - *Complexity — If yes, how much time/space does it need?*

Introduction

- Key models of computation:
  Finite Automata (regular languages) → Pushdown Automata (context-free) → Turing Machines (general algorithms).

Why study Finite Automata?

- Finite automata are a useful model for many important kinds of hardware and software.
- Software for designing and checking the behavior of digital circuits
- The lexical analyzer of a typical compiler that is the compiler component that breaks the input text into logical units such as identifiers keywords and punctuation
- Software for scanning large bodies of text such as collections of Web pages to find occurrences of words phrases or other patterns
- Software for verifying systems of all types that have a finite number of distinct states such as communications protocols or protocols for secure exchange of information

Introduction to Finite Automata

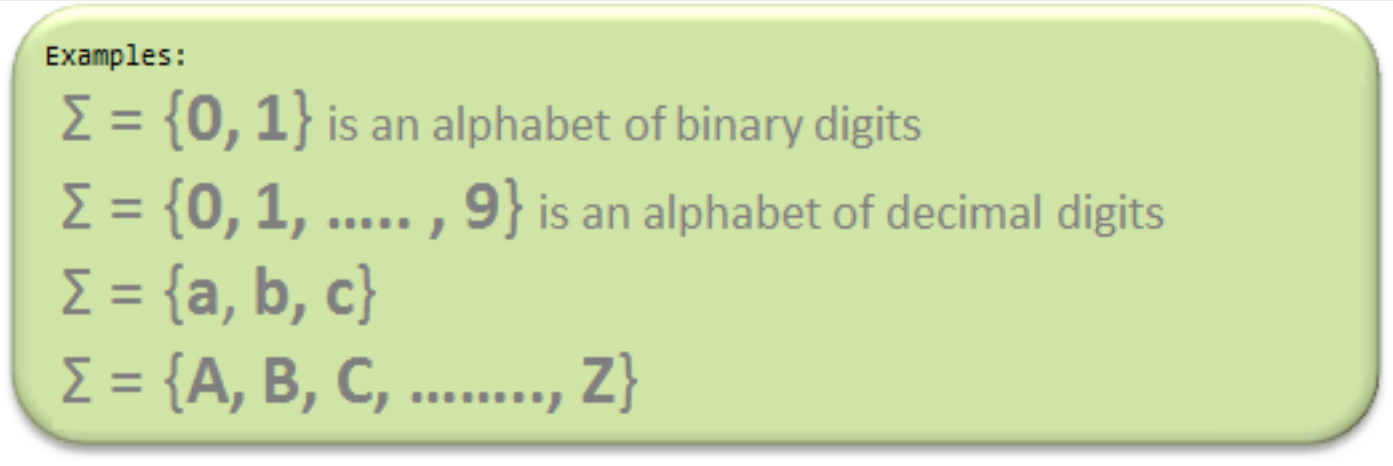
- Automata theory, also known as the Theory of Computation, is a field within computer science and mathematics that focuses on studying abstract machines to understand the capabilities and limitations of computation by analyzing mathematical models of how machines can perform calculations.

Symbol

- A symbol (often also called a character) is the smallest building block, which can be any alphabet, letter, or picture.
- a, b, c, 0, 1, ...

Alphabets (Σ)

- A finite, non-empty set of symbols used to construct strings and languages. For example, Σ = {a, b}.



String

- A string is a finite sequence of symbols from some alphabet. A string is generally denoted as w and the length of a string is denoted as |w|. Empty string is the string with zero occurrence of symbols, represented as ε.
- Number of Strings (of length 2)
  that can be generated over the alphabet {a, b}:
  - -
  a a
  a b
  b a
  b b

  Length of String |w| = 2
  Number of Strings = 4

  Conclusion:
  For alphabet {a, b} with length n, number of
  strings can be generated = 2n.
- Automata theory is used in modeling computational problems hence enhancing the understanding and design of systems such as compilers, interpreters among others.

String

- **Concatenation**: ω=abd, α=ce, $\omega\alpha$=abdce
- **Exponentiation**: ω=abd, ω^3=abdabdabd, $\omega^0=\varepsilon$
- **Reversal**: ω=abd, ω^R = dba
- Σ^k = set of all k-length strings formed by the symbols in Σ

  e.g., Σ={a,b}, Σ^2={ab, ba, aa, bb}, $\Sigma^0=\{\varepsilon\}$

String

- **Kleene Closure** $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots = \cup_{k \geq 0} \Sigma^k$

  e.g., $\Sigma = \{a, b\}$, $\Sigma^* = \{\varepsilon, a, b, ab, aa, ba, bb, aaa, aab, abb, \ldots\}$ is the set of all strings formed by a's and b's.

- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots = \cup_{k > 0} \Sigma^k$

  i.e., Σ^* without the empty string.

Language

- A language is a set of strings formed using the symbols of a given alphabet Σ.
- Formally, a language is a subset of Σ* is the set of all possible strings (including ε) over the alphabet Σ.
- Finite Language:
  L1 = { set of string of 2 }
  L1 = { xy, yx, xx, yy }

  Infinite Language:
  L1 = { set of all strings starts with 'b' }
  L1 = { babb, baa, ba, bbb, baab, ... }

Problem

- In automata theory, a problem is to decide whether a given string is a member of some particular language.
- This formulation is general enough to capture the difficulty levels of all problems.

Finite Automata/ Finite State Machine

- This is the simplest kind of machine.
- We will study 3 types of Finite Automata:
  - *Deterministic Finite Automata (DFA)*
  - *Non-deterministic Finite Automata (NFA)*
  - *Finite Automata with ε-transitions (ε-NFA)*

Core Areas of the Theory of Computation

- The field of computation theory can be broadly divided into four major areas:

Core Areas of the Theory of Computation

- **Automata Theory:**

  Automata theory studies abstract computational models and their applications. It forms the basis for understanding how machines process inputs and produce outputs. Key components include:

  - ***Finite Automata:*** *Used to model simple systems like lexical analyzers in compilers.*
  - ***Pushdown Automata:*** *A more powerful model capable of recognizing context-free languages, essential for parsing programming languages.*
  - ***Turing Machines:*** *The most powerful automata, used as a standard for defining what is computable.*

Core Areas of the Theory of Computation

- **Formal Languages and Grammars**

  This area examines the syntax and structure of languages used in computation. It involves: Regular languages, Context-free Languages

Core Areas of the Theory of Computation

- **Computability and Decidability**
  - *Computability theory addresses the question: What problems can a computer solve? It studies concepts like:*
  - *Decidable Problems: Problems with an algorithmic solution.*
  - *Undecidable Problems: Problems, such as the Halting Problem, for which no algorithm can determine a solution for all inputs.*

Core Areas of the Theory of Computation

- **Complexity Theory**
  - *Complexity theory focuses on the efficiency of algorithms by analyzing the time and space resources they require. It categorizes problems into classes such as: P, NP, NP-Complete and NP-Hard.*

Example

Example 1.1: Perhaps the simplest nontrivial finite automaton is an on/off switch. The device remembers whether it is in the "on" state or the "off" state, and it allows the user to press a button whose effect is different, depending on the state of the switch. That is, if the switch is in the off state, then pressing the button changes it to the on state, and if the switch is in the on state, then pressing the same button turns it to the off state.

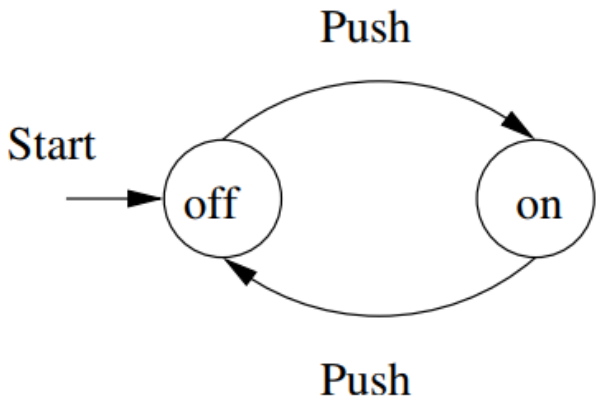


Figure 1.1: A finite automaton modeling an on/off switch

Example 1.2: Sometimes, what is remembered by a state can be much more complex than an on/off choice. Figure 1.2 shows another finite automaton that could be part of a lexical analyzer. The job of this automaton is to recognize the keyword `then`. It thus needs five states, each of which represents a different

position in the word `then` that has been reached so far. These positions correspond to the prefixes of the word, ranging from the empty string (i.e., nothing of the word has been seen so far) to the complete word.

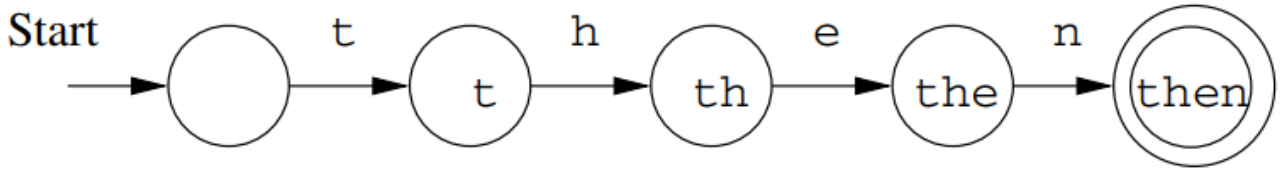


Figure 1.2: A finite automaton modeling recognition of `then`

Definition of DFA

- A DFA is a machine that reads an input string one symbol at a time and moves between a **finite** set of states. It accepts if it stops in an accepting state.
- **Why "deterministic"?**
  For every state $q \in Q$ and symbol $a \in \Sigma$, **there is exactly one next state** $\delta(q, a)$.
- So, for any input string, the DFA has **one unique computation path**—no choices, no branching, no ε-moves.
- A DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,
  - *Q is a finite set of states*
  - Σ *is a finite input alphabet*
  - δ *is the transition function mapping* $Q \times \Sigma$ *to Q*
  - q_0 *in Q is the initial state (only one)*
  - $F \subseteq Q$ *is a set of final states / accepting states (zero or more)*

Components of 5-Tuple

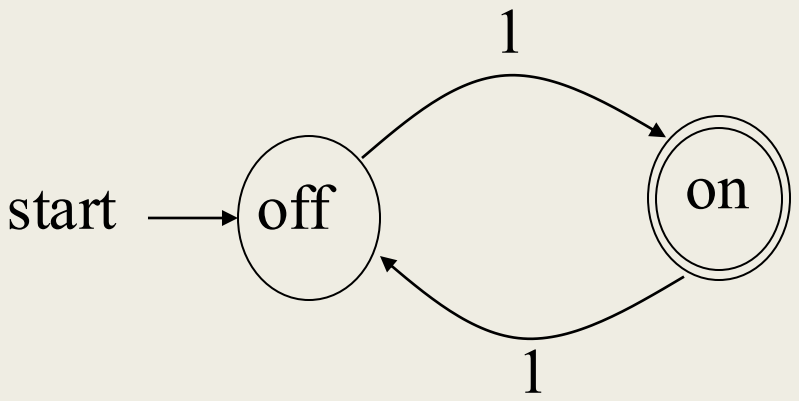
- **States (Q)** – Q is a finite, non-empty set of states. Each state represents a unique condition or configuration of the system.
- **Input Alphabets (Σ)** – Σ is the finite, non-empty set of input symbols that the automaton reads to determine state transitions.
- **Transition Function (δ)** – δ is a set of functions that defines how the automaton transitions from one state to another based on the current state and input symbol. It can be expressed as: **δ:Q×Σ→Q**.
- **Initial State (q0)** – q0 is the initial state from which the automaton starts its computation. It belongs to the set.
- **Final States (F)** – F is a subset of containing one or more final states. These states signify the acceptance of the input string by the automaton.

Transition Function (δ)

- A transition function that takes as arguments a state and and input symbol and returns a state. The transition function will commonly be denoted δ. In our informal graph representation of automata, δ was represented by arcs between states and the labels on the arcs. If *q* is a state and *a* is an input symbol then, δ *(q, a)* is that state *p* such that there is an arc labeled *a* from *q* to *p*.

Definition of DFA

- For example:



- Q is the set of states: {on, off}
- Σ is the set of input symbols: {1}
- δ is the transitions: off $\times$ 1 $\rightarrow$ on; on $\times$ 1 $\rightarrow$ off
- q_0 is the initial state: off
- F is the set of final states (double circle): {on}

Language of DFA

- Given a DFA M, the language accepted (or recognized) by M is the set of all strings that, starting from the initial state, will reach one of the final states after the whole string is read.
- For example, the language accepted by the previous example is the set of all 0 and 1 strings with even number of 0's and 1's.

Applications of Finite Automata

Application Area	Description
Language Processing	Used in email filters and smart assistants like Alexa, Siri, and Google Assistant.
Compiler Construction	Translates high-level programming languages into machine code.
Computer Networks	Underlies the design of network protocols such as HTTP and HTTPS.
Video Games	Manages game mechanics and AI, as seen in games like Warcraft 3.
Digital Circuit Design	Designs digital circuits in devices such as fans, refrigerators, and washing machines.
Biomedical Problem Solving	Utilized in medical imaging technologies like X-ray diffraction and tomography.

Limitations of Finite Automata

- With respect to computation, there are certain limitations of finite automata. Finite automata are connected to regular languages, which is the simplest class of formal languages in the Chomsky hierarchy. These languages can be recognized by finite automata or described by regular expressions.
- They include useful patterns like strings with even 0s, strings ending with "**ing**", and valid email addresses, etc. But for a little complex problem like common patterns matching as we see in balanced parentheses checking, arbitrary palindromes, and strings with equal symbols. The finite automata fail due to its finite memory.

Limitations of Finite Automata Considering Memory Constraints

- The main problem for finite automata is that finite automata have limited memory beyond the current state. This means they can only remember a limited amount of information about the input they've seen.
- Once transitioning to a new state, they lose all information except for the current one. This makes it impossible for finite automata to handle tasks that require tracking an unbounded amount of information. Due to this issue they are not capable of counting unbounded quantities.

Examples of Languages beyond Finite Automata

- **$\{a^n b^n \mid n \geq 0\}$** – Strings with an equal number of 'a's followed by 'b's. This requires counting and comparing the number of 'a's and 'b's, which is impossible with finite memory.
- **{ww | w is any string}** – Strings that consist of two identical halves. Recognizing this language would require remembering the entire first half of the string to compare it with the second half.
- **$\{a^n b^m c^n \mid n, m \geq 0\}$** – Strings where the number of 'a's matches the number of 'c's, with any number of 'b's in between. This language requires counting and remembering two separate quantities across an arbitrary distance.

Conclusion

- Finite automata is a powerful concept used to model and analyze computational processes, however it has some limitations.
- Finite automata suffers from finite memory problem and it cannot keep anything in memory except the current state.