**Theory of Computing**

**CSE - 203**

**Course Objective:**

* Introduce concepts in automata theory and theory of computation
* Identify different formal language classes and their relationships
* Design grammars and recognizers for different formal languages
* Prove or disprove theorems in automata theory using its properties
* Determine the decidability and intractability of computational problems

**Why should I study CSE-203?**

* Regular expressions are used in many systems, Like UNIX.
* Context-free grammars are used to describe the syntax of essentially every programming language.
* When developing solutions to real problems, we often confront the limitations of what software can do. CSE-203 gives you the tools.

**Syllabus**

|  |  |
| --- | --- |
| 1. Introduction to Automata 2. Deterministic Finite Automata 3. Nondeterministic Finite Automata 4. Regular Expressions 5. Context-Free Languages | 1. Non Context Free Language (NCFL) 2. Parse Trees 3. PushDown Automata (PDA) 4. Turing Machines 5. Undecidable Problems |

**Text Book**

|  |  |
| --- | --- |
| Reference | Introduction to the Theory of Computation  Michael Sipser: 3rd Edition |
| Further Reading | Introduction to automata theory, languages and computation  (JE Hopcroft, R Motwani and JD Ullman) Addison Wesley/ Pearson; 3rd Edition |

**Marks Distribution**

|  |  |  |  |
| --- | --- | --- | --- |
| S.L | Exam | Mark | Syllabus |
| 1 | Midterm | 30 | 1 – 5 |
| 2 | Assignment | 10 | A-1, A-2 |
| 5 | Final | 40 | 6 - 10 |
| 4 | Teacher Assessment | 10 | Class attendance, Class Performance |
| Total | | 100 |  |

**Assignment policy:**

* Assignment must be submitted in class as hardcopy on the due date mentioned in the homework
* All Assignment must be done individually.
* No late submissions will be allowed on any Assignment.

**Introduction to Automata**

**Automata – What is it?**

Study of abstract computing devices, or *“machines”*

The term "Automata" is derived from the Greek word "αὐτόματα" which means "selfacting". An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically. An automaton with a finite number of states is called a **Finite Automaton** (FA) or **Finite** **State Machine** (FSM).

**Alphabet**

An alphabetis any finite, non-empty set of symbols. We use the symbol Σ (sigma) to denote an alphabet. Examples:

* Binary: Σ = {0,1}
* All lower case letters: Σ = {a, b, c, ..z}
* Alphanumeric: Σ = {a-z, A-Z, 0-9}
* DNA molecule letters: Σ = {a,c,g,t}

Where ‘a’, ‘b’, ‘c’, and ‘d’ are *symbols*.

**String**

A *string*is a finite sequence of symbols taken from Σ.

Example: ‘cabcad’ is a valid string on the alphabet set Σ = {a, b, c, d}

**Length of a String**

It is the number of symbols present in a string. (Denoted by **|S|**).

Examples:

* If S=‘cabcad’, |S|= 6
* If |S|= 0, it is called an empty string(Denoted by λor **ε**)

**Kleene Star**

The Kleene star, **Σ\***, is a unary operator on a set of symbols or strings, **Σ**, that gives the infinite set of all possible strings of all possible lengths over **Σ** including **λ**.

Representation: Σ\* = Σ0 U Σ1 U Σ2 …. Where Σp is the set of all possible strings of length p.

Example: If Σ = {a, b}, Σ\*= {λ, a, b, aa, ab, ba, bb, ………}

**Kleene Closure / Plus**

The set **Σ+** is the infinite set of all possible strings of all possible lengths over Σ excluding λ.

Representation: Σ+ = Σ1 U Σ2 U Σ3 U…….

Σ+ = Σ\* − { λ }

Example: If Σ = {a, b } , Σ+ ={ a, b, aa, ab, ba, bb,………..}

**Language**

A language is a subset of Σ\* for some alphabet Σ. It can be finite or infinite.

Example: If the language takes all possible strings of length 2 over Σ = {a, b},

then L = { ab, bb, ba, bb}

Finite Automaton can be classified into two types −

* Deterministic Finite Automaton (DFA)
* Non-deterministic Finite Automaton (NDFA / NFA)

**Deterministic Finite Automaton (DFA)**

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called Deterministic Automaton. As it has a finite number of states, the machine is called Deterministic Finite Machine or Deterministic Finite Automaton.

**Formal Definition of a DFA**

A DFA can be represented by a 5-tuple (Q, ∑, δ, q0, F) where −

* **Q** is a finite set of states.
* **∑** is a finite set of symbols called the alphabet.
* **δ** is the transition function where δ: Q × ∑ → Q
* **q0** is the initial state from where any input is processed (q0 ∈ Q).
* **F** is a set of final state/states of Q (F ⊆ Q).

**Graphical Representation of a DFA**

A DFA is represented by digraphs called **state diagram**.

* The vertices represent the states.
* The arcs labeled with an input alphabet show the transitions.
* The initial state is denoted by an empty single incoming arc.
* The final state is indicated by double circles.

**Example**

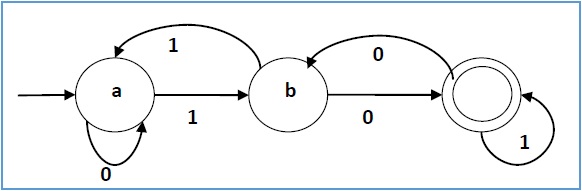
Let a deterministic finite automaton be →

* Q = {a, b, c},
* ∑ = {0, 1},
* q0 = {a},
* F = {c}, and

Transition function δ as shown by the following table −

|  |  |  |
| --- | --- | --- |
| **Present State** | **Next State for Input 0** | **Next State for Input 1** |
| **a** | a | b |
| **b** | c | a |
| **c** | b | c |

Its graphical representation would be as follows −



**Finite Automata**

Some Applications

* Software for designing and checking the behavior of digital circuits
* Lexical analyzer of a typical compiler
* Software for scanning large bodies of text (e.g., web pages) for pattern finding
* Software for verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol)

**Example**

|  |  |
| --- | --- |
| Fig: A finite automation modeling an on/off switch | Fig: A finite automation modeling recognition of “then” |

**Structural Representations**

There are two important notations that are not automaton like, but play an important role in the study of automata and their applications.

1. **Grammars** are useful models when designing software that processes data with a recursive structure. The best known example is a "parser" the component of a compiler that deals with the recursively nested features of the typical programming language, such as expressions, arithmetic, conditional, and so on.

2. **Regular Expressions** also denote the structure of data especially text strings.

The UNIX style regular expression “[A-Z] [a-z]\*[ ] [A-Z] [A-Z]”represents capitalized words followed by a space and two capital letters. This expression represents patterns in text that could be a city and state, e.g., Ithaca NY.

It misses multiword city names such as **Palo Alto CA** which could be captured by the more complex expression

“[A-Z] [a-z]\*([ ][A-Z][a-z]\*)\*[ ][A-Z][A-Z]”

**Automata and Complexity**

Automata are essential for the study of the limits of computation. There are two important issues.

* What can a computer do at all? This study is called “decidability”, and the problems that can be solved by computer are called “decidable”.
* What can a computer do efficiently? This study is called “intractability”, and the problems that can be solved by a computer using no more time than some slowly growing function of the size of the input are called “tractable”.

Meet “ABA” The Automaton!

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | **Input String** | **Result** | | aba | Accept | | aabb | Reject | | aabba | Accept | | ε | Accept | |

**How Machine M operates**

* M “reads” one letter at a time from the input string (going from left to right)
* M starts in state q0.
* If M is in state *qi* reads the letter *a* then

If δ(qi. a) is undefined then CRASH. Otherwise M moves to state δ(qi, a)

**Finite Automation**

|  |  |  |
| --- | --- | --- |
| Finite set of states |  | Q = {qo, q1, q2,…., qk} |
| A start state |  | qo |
| A set of accepting states |  | F = {qio, qi1, qi2,…., qir} |
| A finite alphabet | *a, b, #*, *x, 1* | ∑ |
| State transition instructions |  | &: Q x ∑ -> Q  d(qi, a) = qj |

**Let *M =* (*Q,* ∑*, F,* &) be a finite automaton.**

*M* *accepts* the string *x* if when *M* reads *x* it ends in an accepting state.

*M rejects* the string *x* if when *M* reads *x* it ends in a non-accepting state.

*M* *crashes* on *x* if *M* crashes while reading *x*.

|  |
| --- |
| What is the language accepted by this machine? |
| *L =* {*a,b*}*\* =* all finite strings of *a*’s and *b*’s |

|  |
| --- |
| What is the language accepted by this machine? |
| *L = all even length strings of a’s and b’s* |

|  |
| --- |
| What machine accepts this language? |
| *L =* all strings in {*a,b*}\* that contain at least one *a* |

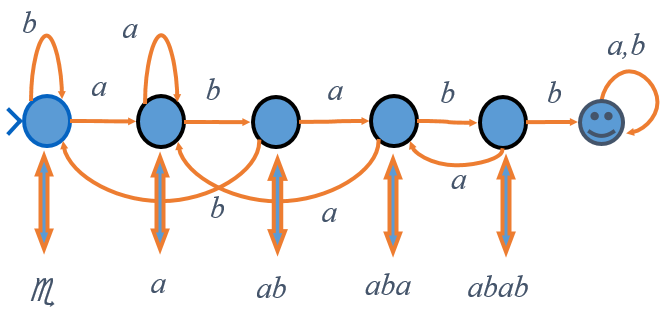
|  |
| --- |
| What machine accepts this language? |
| *L = strings with an odd number of b’s and any number of a’s* |

|  |
| --- |
| What is the language accepted by this machine? |
| *L =* any string ending with a *b* |
| What is the language accepted by this machine? |
| *L = any string with at least two a’s* |

|  |
| --- |
| What machine accepts this language? |
| *L =* any string with an *a* and a *b* |

|  |
| --- |
| What machine accepts this language? |
| *L = strings with an even number of ab pairs* |

*L =* all strings containing *ababb* as a consecutive substring



**Formal Proofs**

1. Inductive Proof
2. Deductive Proof

(Self-Study………see reference book chapter 0)

**Quantifiers**

* *For all”* or “*For every”*
  + Universal proofs
  + Notation = 
* *“There exists”*
  + Used in existential proofs
  + Notation = 
* Implication is denoted by =>

E.g., “IF A THEN B” can also be written as “A=>B”

**Homework**

