UNIVERSITY INSTITUTE OF ENGINEERING

COMPUTER SCIENCE ENGINEERING

Bachelor of Engineering

Theory of Computation (CST-353)

Topic: Introduction

Learning Objectives & Outcomes

Objective:

- To understand the basics to automata.

Outcome:

- Student will understand the
  - Basics of automata

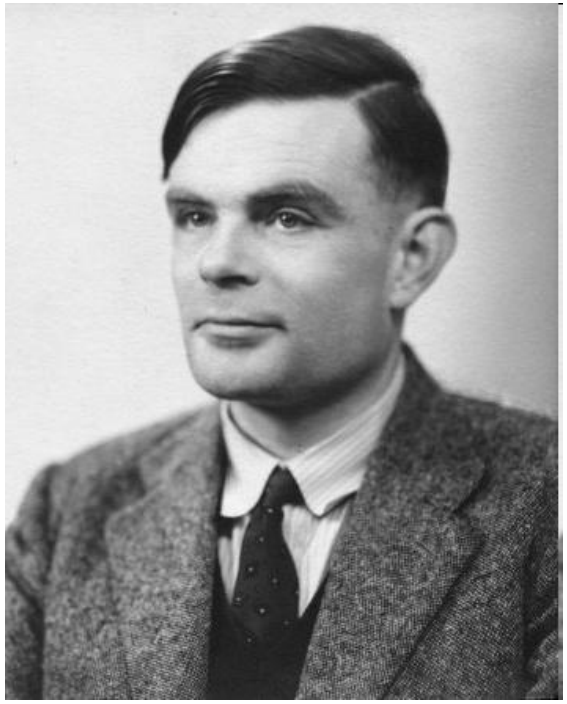
What is Automata Theory?

- *Study of abstract computing devices, or "machines"*
- Automaton = an abstract computing device
- A fundamental question in computer science:
  - Find out what different models of machines can do and cannot do
  - The *theory of computation*
- Computability vs. Complexity

(A pioneer of automata theory)

Alan Turing (1912-1954)

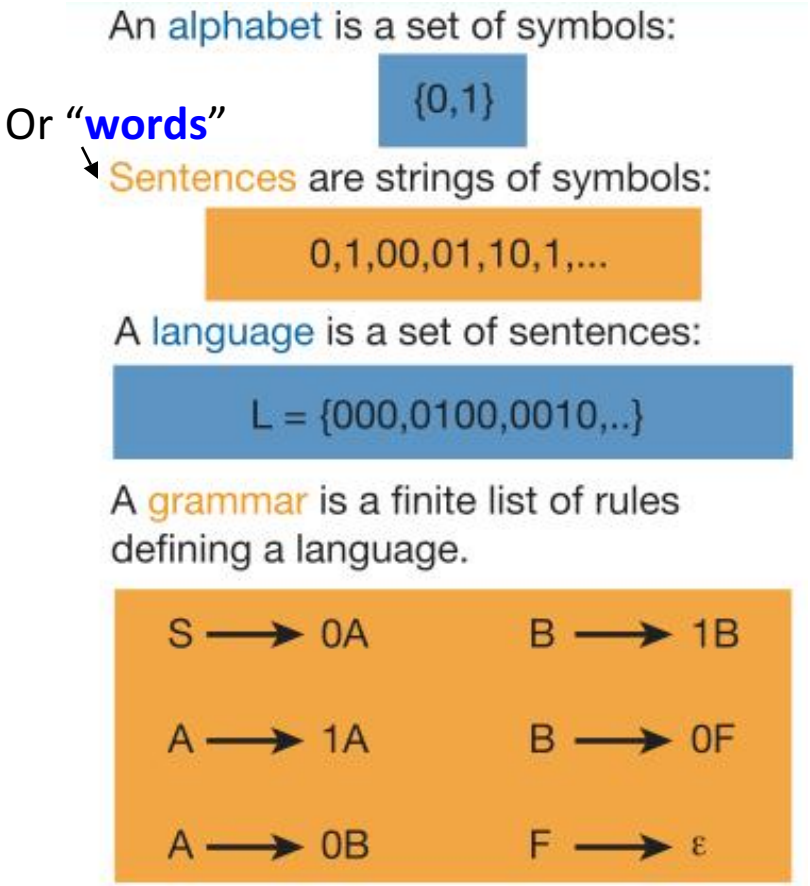
- Father of Modern Computer Science
- English mathematician
- Studied abstract machines called ***Turing machines*** even before computers existed
- Heard of the Turing test?



Theory of Computation: A Historical Perspective

1930s	• Alan Turing studies Turing machines • Decidability • Halting problem
1940-1950s	• "Finite automata" machines studied • Noam Chomsky proposes the "Chomsky Hierarchy" for formal languages
1969	Cook introduces "intractable" problems or "NP-Hard" problems
1970-	Modern computer science: compilers, computational & complexity theory evolve

Languages & Grammars

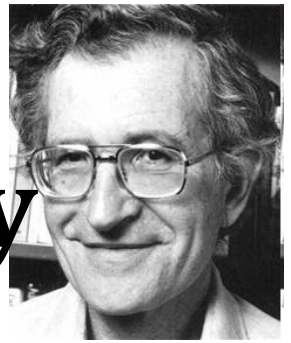


- Languages: "*A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols*"
- Grammars: "*A grammar can be regarded as a device that enumerates the sentences of a language*" - nothing more, nothing less

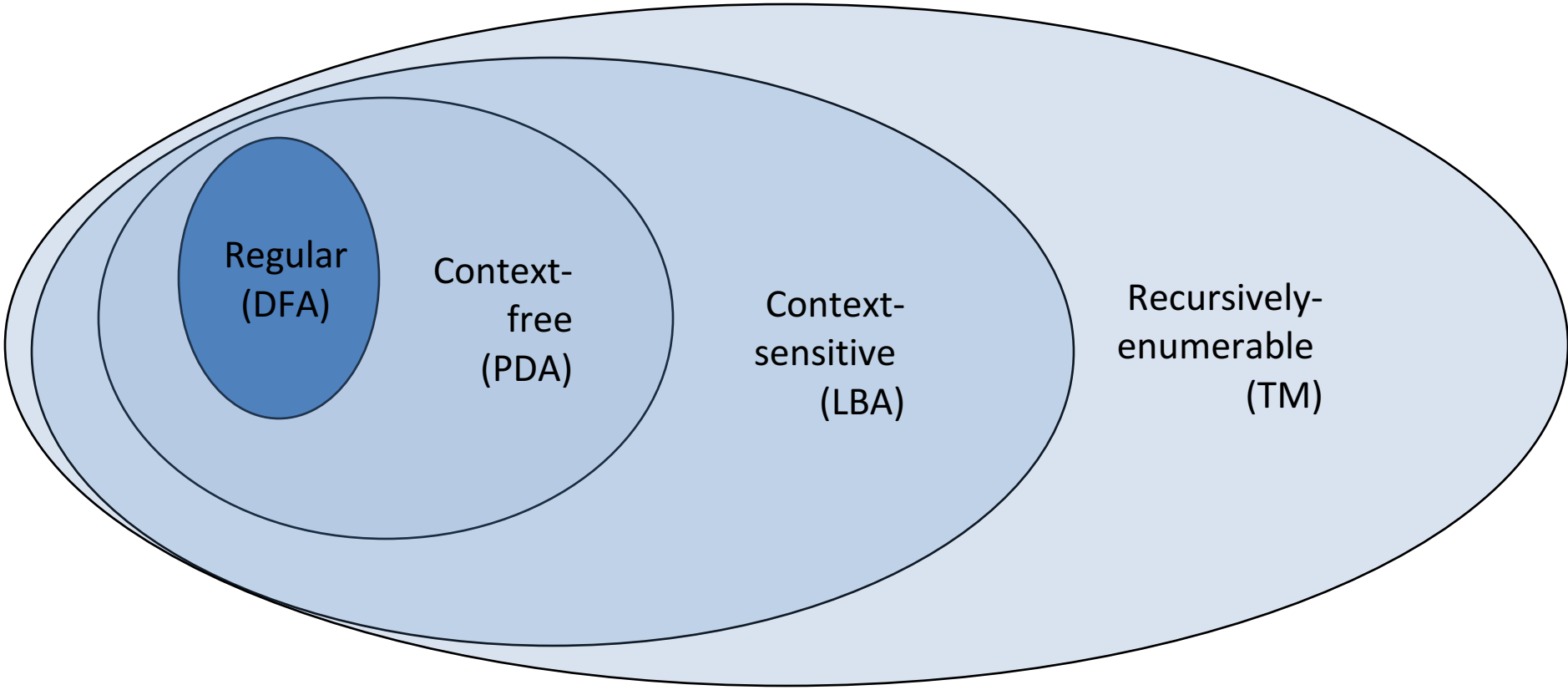
- *N. Chomsky, Information and Control, Vol 2, 1959*

Image source: Nowak et al. Nature, vol 417, 2002

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



Finite Automata

1. Finite automata are used to recognize patterns.

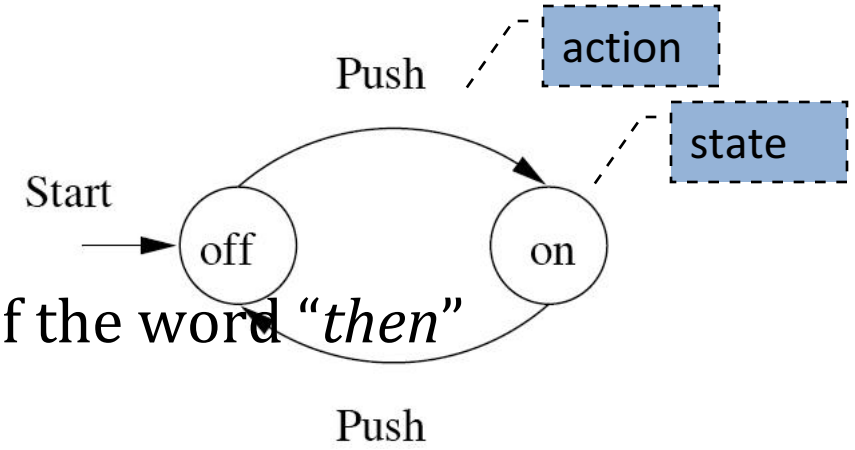
2. It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.

3. At the time of transition, the automata can either move to the next state or stay in the same state.

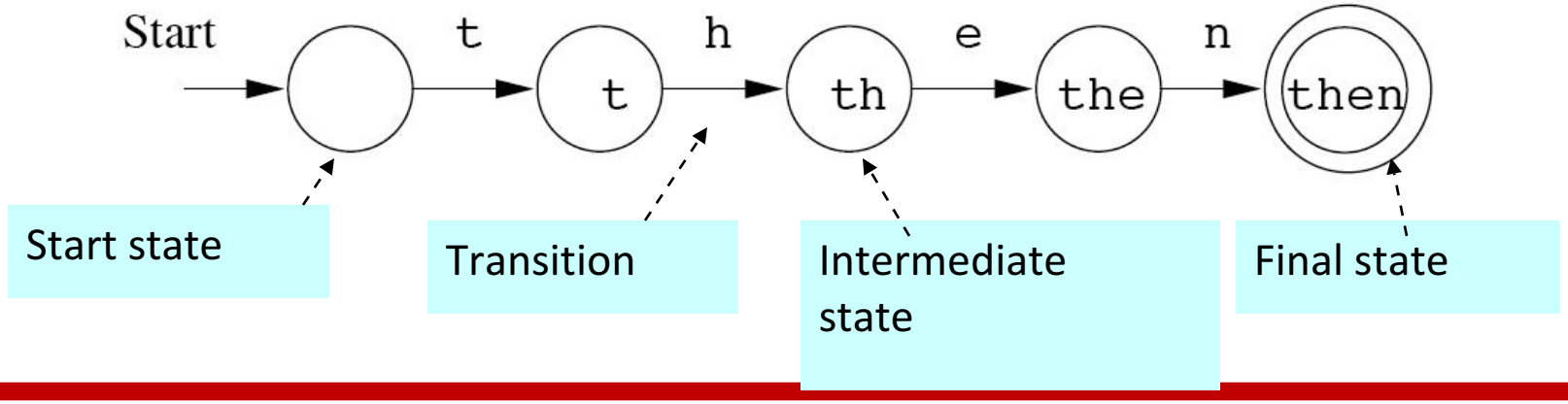
4. Finite automata have two states, **Accept state** or **Reject state**. When the input string is processed successfully, and the automata reached its final state, then it will accept.

Finite Automata : Examples

- On/Off switch



- Modeling recognition of the word "*then*"



Formal Definition of FA

A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

Q: finite set of states

∑: finite set of the input symbol

q0: initial state

F: **final** state

δ: Transition function

Finite Automata Model:

Finite automata can be represented by input tape and finite control.

Input tape: It is a linear tape having some number of cells. Each input symbol is placed in each cell.

Finite control: The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read.

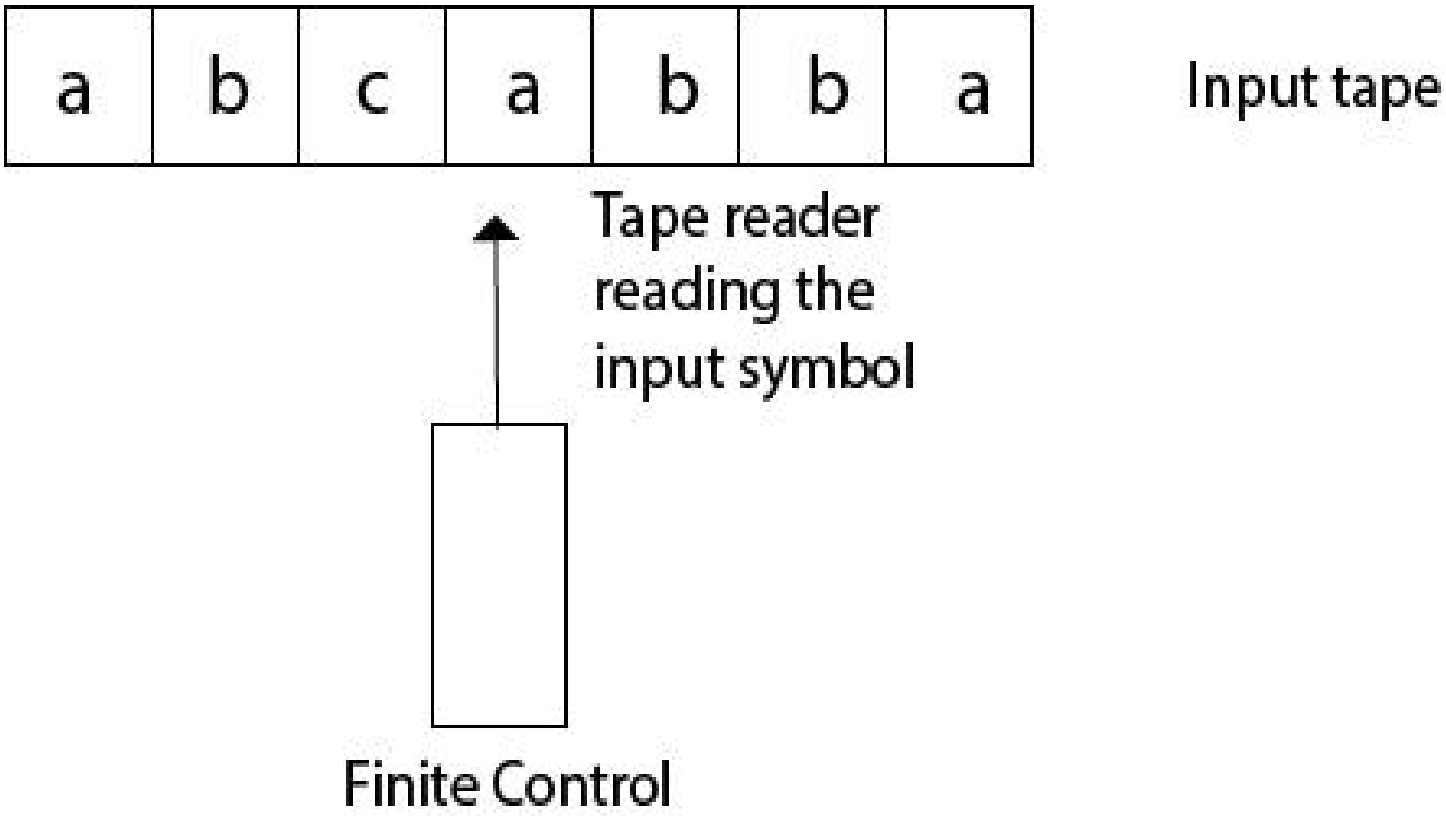


Fig :- Finite automata model

Types of Automata:

There are two types of finite automata:

DFA(deterministic finite automata)

NFA(non-deterministic finite automata)

Some important points about DFA and NFA:

Every DFA is NFA, but NFA is not DFA.

There can be multiple final states in both NFA and DFA.

DFA is used in Lexical Analysis in Compiler.

NFA is more of a theoretical concept.

Alphabet

An alphabet is a finite, non-empty set of symbols

- We use the symbol ∑ (sigma) to denote an alphabet
- Examples:
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}
  - DNA molecule letters: ∑ = {a,c,g,t}
  - …

Powers of an alphabet

Let $\sum$ be an alphabet.

- $\sum^k$ = the set of all strings of length k
- $\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup \ldots$
- $\sum^+ = \sum^1 \cup \sum^2 \cup \sum^3 \cup \ldots$

Strings

A string or word is a finite sequence of symbols chosen from $\sum$

- ***Empty string is*** ε ***(or "epsilon")***

- Length of a string *w*, denoted by "|*w*|", is equal to the *number of (non-* ε*) characters in the string*
  - *E.g., x = 010100* — *|x| = 6*
  - *x = 01* ε *0* ε *1* ε *00* ε — *|x| = ?*

  - *xy = c*oncatentation of two strings *x* and *y*

Summary

- Automata theory & a historical perspective
- Chomsky hierarchy
- Finite automata

FAQ :

- What are types of finite automata.

2. Explain model of finite automata.

3. Explain tuples of finite automata.

References :

- Martin J.C., "Introduction *to Languages and Theory of Computation*", Tata McGraw-Hill Publising Company Limited, 3rd Edition.
- https://youtu.be/S3cOulqSAmU
- Https://en.wikipedia.org/wiki/Finite-state_machine
- https://www.safaribooksonline.com
- https://nptel.ac.in/courses/106/103/106103070/

CU
CHANDIGARH UNIVERSITY

THANK YOU

UNIVERSITY INSTITUTE OF ENGINEERING

COMPUTER SCIENCE ENGINEERING

Bachelor of Engineering

Theory of Computation (CST-353)

Topic: Introduction

Learning Objectives & Outcomes

Objective:

- To understand the concept of Formal Language, Operations and examples of formal languages.

Outcome:

- Student will understand the
  - Formal Language
  - Operation on formal language

Languages

L is a said to be a language over alphabet ∑, only if $L \subseteq \sum^*$

➔ this is because ∑* is the set of all strings (of all possible length including 0) over the given alphabet ∑

Examples:

- Let L be *the* language of all strings consisting of *n* 0's followed by *n* 1's:
  L = {ε,01,0011,000111,…}
- Let L be *the* language of all strings of with equal number of 0's and 1's:
- L = {ε,01,10,0011,1100,0101,1010,1001,…}

- **Definition: Ø denotes the Empty language**

- Let L = {ε}; Is L=Ø? NO

Language:

A language is a collection of appropriate string. A language which is formed over Σ can be **Finite** or **Infinite**.

Example: 1

•L1 = {Set of string of length 2}

= {aa, bb, ba, bb} **Finite Language**

Example: 2

•L2 = {Set of all strings starts with 'a'}

= {a, aa, aaa, abb, abbb, ababb} **Infinite Language**

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)

Set operations on languages :

Union: A string $x \in L_1 \cup L_2$ iff $x \in L_1$ or $x \in L_2$

Example: $\{0, 11, 01, 011\} \cup \{1, 01, 110\} = \{0, 11, 01, 011, 111\}$

Intersection: A string $x \in L_1 \cap L_2$ iff $x \in L_1$ and $x \in L_2$.

Example: $\{0, 11, 01, 01\} \cap \{1, 01, 110\} = \{01\}$

Complement: Usually, Σ^* is the universe that a complement is taken with respect to. Thus for a language L, the complement is L(bar) = $\{ x \in \Sigma^* \mid x \notin L \}$.

Example: Let $L = \{x \mid |x| \text{ is even}\}$. Then its complement is the language $\{x \in \Sigma^* \mid |x| \text{ is odd}\}$. Similarly we can define other usual set operations on languages like relative complement, symmetric difference, etc.

Reversal of a language :

The reversal of a language L, denoted as L^R, is defined as: $L^R = \{w^R \mid w \in L\}$.

Example:

1. Let $L = \{0, 11, 01, 011\}$. Then $L^R = \{0, 11, 10, 110\}$.
2. Let $L = \{ 1^n0^n \mid n \text{ is an integer}\}$. Then $L^R = \{ 1^n0^n \mid n \text{ is an integer}\}$.

Language concatenation: The concatenation of languages L_1 and L_2 is defined as $L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$.

Example: $\{a, ab\}\{b, ba\} = \{ab, aba, abb, abba\}$.

Example of Formal Language:

•A language is a collection of appropriate string. A language which is formed over Σ can be **Finite** or **Infinite**.

Example: 1

•L1 = {Set of string of length 2}

= {aa, bb, ba, bb} **Finite Language**

Example 2 :

•Example of Finite Language: L1 = { set of string of 2 } L1 = { xy, yx, xx, yy }

Example: 3

•L2 = {Set of all strings starts with 'a'}

= {a, aa, aaa, abb, abbb, ababb} **Infinite Language**

Example 4 :

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

Summary

- Operations on Formal language
- Alphabets
- strings
- languages

FAQ :

1. Construct the powerset for the following sets.

- {a, b}
- {0, 1}∪{1, 2}
- {z}
- {0, 1, 2, 3, 4}∩{1, 3, 5, a}
- {0, 1, 2, 3}−{1, 3, 5, a}
- ∅ (the empty set)

2. Prove that $12 + 22 + 32 + \ldots\ldots + n2 = \sum n2 = (n(n+1)(2n+1))/6$.

References :

- Martin J.C., "Introduction *to Languages and Theory of Computation*", Tata McGraw-Hill Publising Company Limited, 3rd Edition.
- https://youtu.be/S3cOulqSAmU
- Https://en.wikipedia.org/wiki/Finite-state_machine
- https://www.safaribooksonline.com
- https://nptel.ac.in/courses/106/103/106103070/

CU
CHANDIGARH UNIVERSITY

THANK YOU

UNIVERSITY INSTITUTE OF ENGINEERING

COMPUTER SCIENCE ENGINEERING

Bachelor of Engineering

Theory of Computation (CST-353)

Topic: Finite Automata

Learning Objectives & Outcomes

Objective:

- To understand the concept of Finite Automata, properties and limitation of Finite Automata.

Outcome:

- Student will understand the
  - Finite Automata- DFA, NFA
  - Properties of FA

Finite Automaton (FA)

Finite automata is an abstract computing device. It is a mathematical model of a system with discrete inputs, outputs, states and a set of transitions from state to state that occurs on input symbols from the alphabet Σ.

Finite Automata Representation

The finite automata can be represented in three ways, as given below –

- Graphical (Transition diagram)
- Tabular (Transition table)
- Mathematical (Transition function)

Formal definition of Finite Automata

- Finite automata is defined as a 5-tuples

M=(Q, Σ, δ,q0,F)

Where,

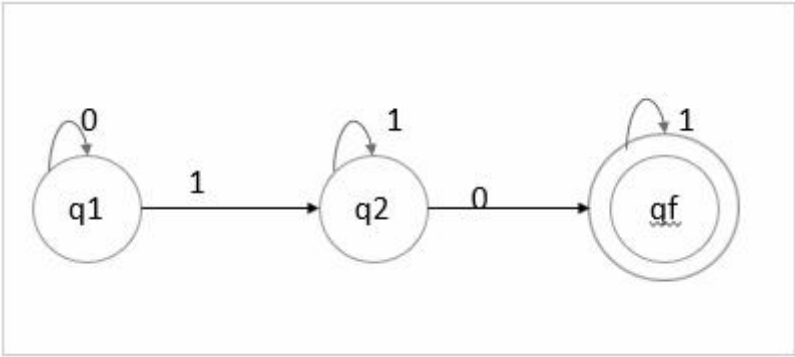
- Q: Finite set called states.
- Σ: Finite set called alphabets.
- δ: $Q \times \Sigma \rightarrow Q$ is the transition function.
- $q0 \in Q$ is the start or initial state.
- F: Final or accept state.

Finite Automata can be represented as follows –

- Transition diagram
- Transition table
- Transition function

Transition Diagram

- It is a directed graph associated with the vertices of the graph corresponding to the state of finite automata.
- An example of transition diagram is given below –

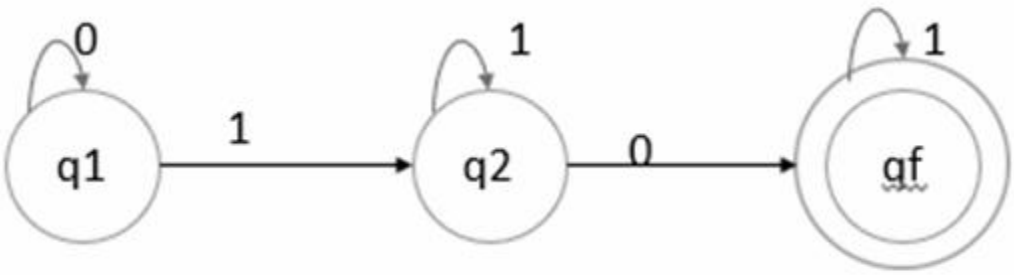


Transition table

It is basically a tabular representation of the transition function that takes two arguments (a state & a symbol) and returns a value (the 'next state').

- **$\delta : Q \times \Sigma \rightarrow Q$**
- In transition table, the following factors are considered –
- Rows correspond to state.
- Column corresponds to the input symbol.
- Entries correspond to the next state.
- The start state is marked with ->.
- The accept state marked with *.

An example of transition table is as follows –



The transition table is as follows −

State/input symbol	0	1
->q1	q1	q2
q2	qf	q2
qf	-	qf

Transition function

The transition function is denoted by δ. The two parameters mentioned below are the passes to this transition function.

- Current state
- Input symbol
- The transition function returns a state which can be called as the next state.

δ (current_state, current_input_symbol) = next_state

- For example, δ(q0,a)=q1

Properties and Limitations of Finite State Machines

There are many properties available when defining an FSM. These properties are available at the various levels of the FSM hierarchy.

- **FSM Level Properties**
- **State Properties**
- **Resolution Action Properties**
- **Transition properties**
- **Event Type Properties**
- **Event Context Properties**

Disadvantages of Finite State Machine

The **disadvantages of the finite state machine** include the following

- The expected character of deterministic finite state machines can be not needed in some areas like computer games
- The implementation of huge systems using FSM is hard for managing without any idea of design.
- Not applicable for all domains
- The orders of state conversions are inflexible.

References

- Https://en.wikipedia.org/wiki/Finite-state_machine
- https://www.safaribooksonline.com
- http://studentsfocus.com/
- http://www.francisxavier.ac.in/

CU
CHANDIGARH UNIVERSITY

THANK YOU

UNIVERSITY INSTITUTE OF ENGINEERING

COMPUTER SCIENCE ENGINEERING

Bachelor of Engineering

Theory of Computation (CST-353)

Topic: Finite Automata

Learning Objectives & Outcomes

Objective:

- To understand the concept of Finite Automata, properties and limitation of Finite Automata.

Outcome:

- Student will understand the
  - Finite Automata- DFA, NFA

Finite Automaton (FA)

- Informally, a state diagram that comprehensively captures all possible states and transitions that a machine can take while responding to a stream or sequence of input symbols
- Recognizer for "Regular Languages"

- Deterministic Finite Automata (DFA)
  - The machine can exist in only one state at any given time
- Non-deterministic Finite Automata (NFA)
  - The machine can exist in multiple states at the same time

Deterministic Finite Automata - Definition

- A Deterministic Finite Automaton (DFA) consists of:
  - Q ==> a finite set of states
  - ∑ ==> a finite set of input symbols (alphabet)
  - q^0 ==> a start state
  - F ==> set of final states
  - δ ==> a transition function, which is a mapping between Q x ∑ ==> Q
- A DFA is defined by the 5-tuple:
  - {Q, ∑ , q^0,F, δ }

What does a DFA do on reading an input string?

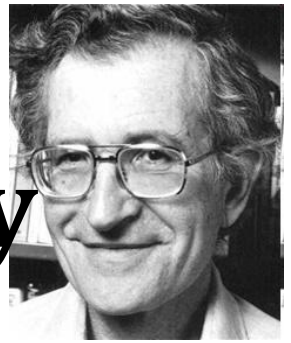
- Input: a word w in $\sum^*$
- Question: Is w acceptable by the DFA?
- Steps:
  - Start at the "start state" q^0
  - For every input symbol in the sequence w do
    - Compute the next state from the current state, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed, the current state is one of the final states (F) then *accept w;*
  - Otherwise, *reject w.*

Regular Languages

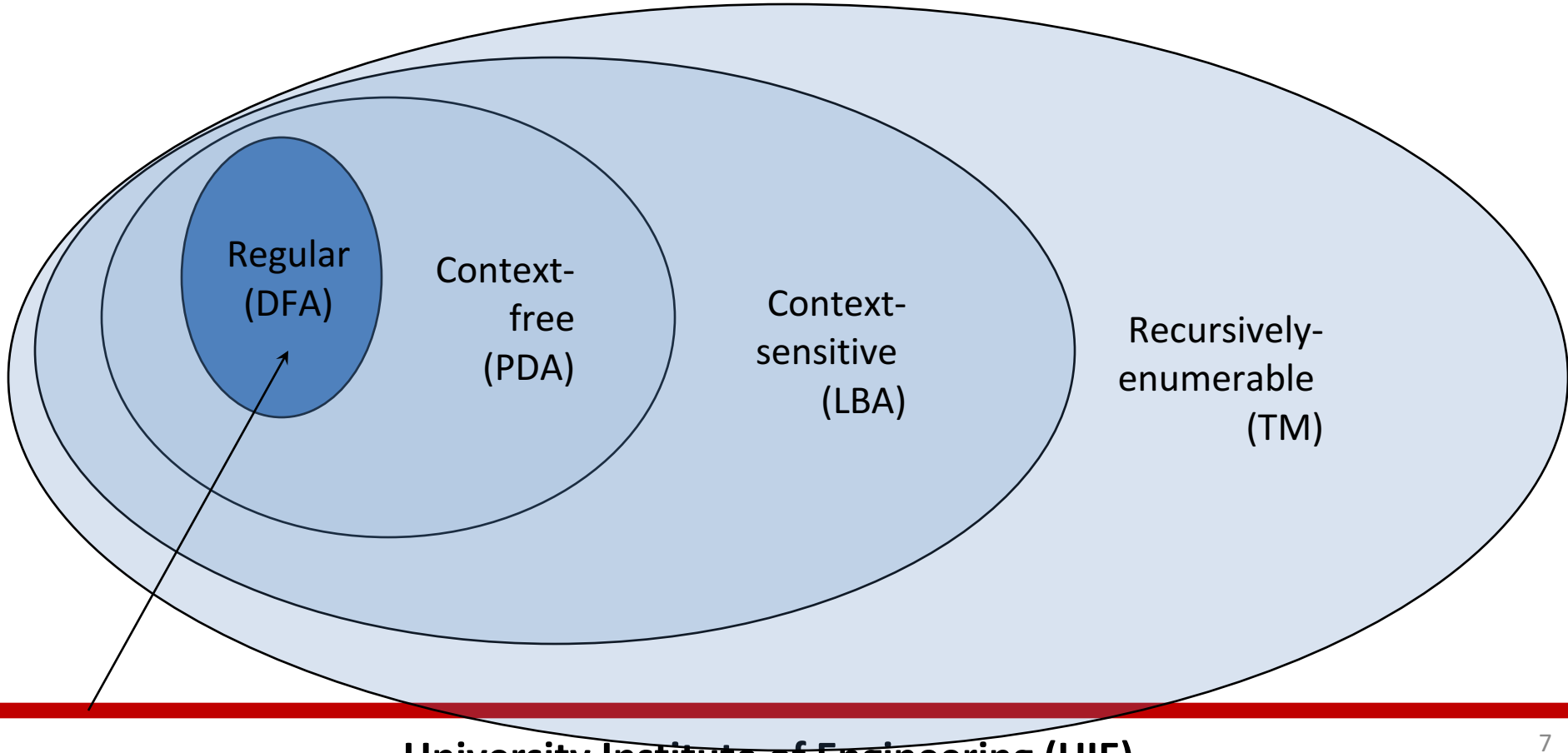
- Let L(A) be a language *recognized* by a DFA A.
  - Then L(A) is called a "*Regular Language*".

- Locate regular languages in the Chomsky Hierarchy

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



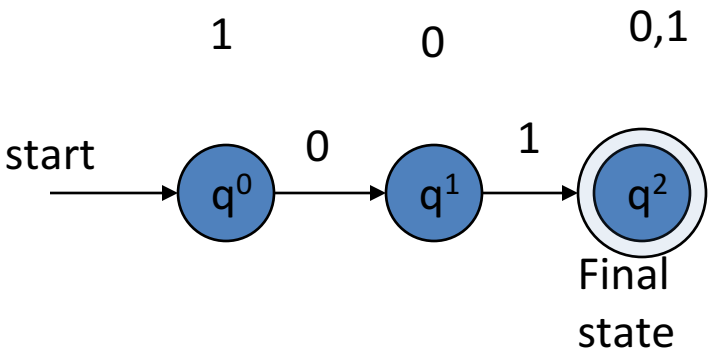
Example #1

- Build a DFA for the following language:
  - L = {w | w is a binary string that contains 01 as a substring}
- Steps for building a DFA to recognize L:
  - ∑ = {0,1}
  - Decide on the states: Q
  - Designate start state and final state(s)
  - δ: Decide on the transitions:
- Final states == same as “accepting states”
- Other states == same as “non-accepting states”

Regular expression: (0+1)*01(0+1)*

DFA for strings containing 01

- What makes this DFA deterministic?



- What if the language allows empty strings?

- $Q = \{q^0, q^1, q^2\}$
- $\Sigma = \{0,1\}$
- start state = q^0
- $F = \{q^2\}$
- Transition table

states	δ	0	1
→	q^0	q^1	q^0
	q^1	q^1	q^2
	$*q^2$	q^2	q^2

symbols

Example #2

Clamping Logic:

- A clamping circuit waits for a ”1” input, and turns on forever. However, to avoid clamping on spurious noise, we’ll design a DFA that waits for *two consecutive 1s* in a row before clamping on.

- Build a DFA for the following language:

  L = { w | w is a bit string which contains the substring 11}

- State Design:

- q^0 : start state (initially off), also means the most recent input was not a 1
  - q^1: has never seen 11 but the most recent input was a 1
  - q^2: has seen 11 at least once

Example #3

- Build a DFA for the following language:

  L = { w | w is a binary string that has even number of 1s and even number of 0s}

- ?

Extension of transitions (δ) to Paths (δ)

- $\hat{\delta}(q,w)$ = *destination state* from state q on input string w
- $\hat{\delta}(q,wa) = \delta(\hat{\delta}(q,w), a)$
  - Work out example #3 using the input sequence w=10010, a=1:
    - $\hat{\delta}(q^0,wa)$ = ?

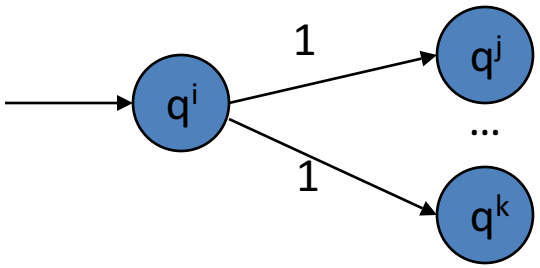
Language of a DFA

A DFA A accepts string w if there is a path from q^0 to an accepting (or final) state that is labeled by w

- *i.e.,* $L(A) = \{ w \mid \delta(q^0, w) \in F \}$
- *I.e., L(A) = all strings that lead to a final state from* q^0

Non-deterministic Finite Automata (NFA)

- A Non-deterministic Finite Automaton (NFA)
  - is of course "non-deterministic"
    - Implying that the machine can exist in more than one state at the same time
    - Transitions could be non-deterministic



• Each transition function therefore maps to a set of states

Non-deterministic Finite Automata (NFA)

- A Non-deterministic Finite Automaton (NFA) consists of:
  - Q ==> a finite set of states
  - ∑ ==> a finite set of input symbols (alphabet)
  - q^0 ==> a start state
  - F ==> set of final states
  - δ ==> a transition function, which is a mapping between Q x ∑ ==> subset of Q
- An NFA is also defined by the 5-tuple:
  - {Q, ∑ , q^0,F, δ }

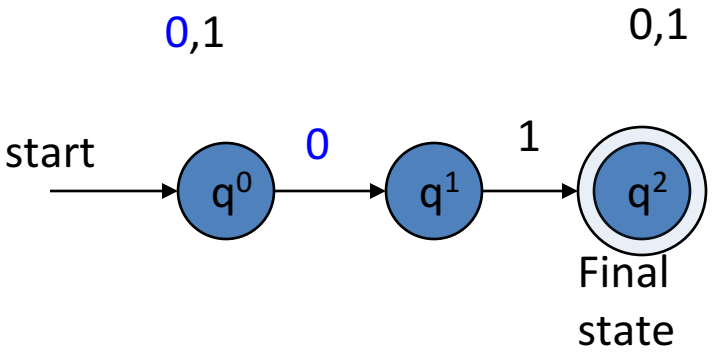
How to use an NFA?

- Input: a word w in $\sum^*$
- Question: Is w acceptable by the NFA?
- Steps:
  - Start at the "start state" q^0
  - For every input symbol in the sequence w do
    - Determine all possible next states from all current states, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed and if at least one of the current states is a final state then *accept w;*
  - Otherwise, *reject w.*

Regular expression: (0+1)*01(0+1)*

NFA for strings containing 01

Why is this non-deterministic?



What will happen if at state q^1 an input of 0 is received?

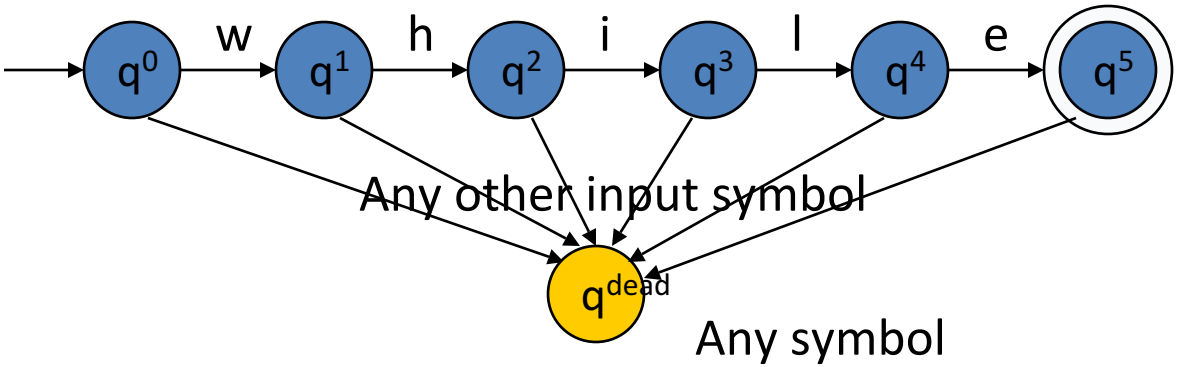
- $Q = \{q^0, q^1, q^2\}$
- $\Sigma = \{0,1\}$
- start state = q^0
- $F = \{q^2\}$
- Transition table

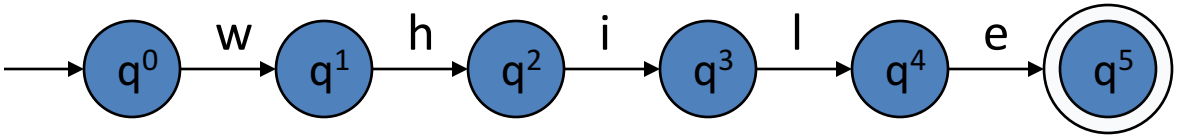
states	δ	symbols 0	1
→	q^0	$\{q^0, q^1\}$	$\{q^0\}$
	q^1	Φ	$\{q^2\}$
	$*q^2$	$\{q^2\}$	$\{q^2\}$

Transitions into a dead state are implicit

What is a "dead state"?

- A DFA for recognizing the key word "*while*"





Example #2

- Build an NFA for the following language:

  L = { w | w ends in 01}

- ?
- Other examples
- Keyword recognizer (e.g., if, then, else, while, for, include, etc.)
  - Strings where the first symbol is present somewhere later on at least once

Extension of δ to NFA Paths

- Basis: $\delta(q,\varepsilon) = \{q\}$
- Induction:
  - Let $\delta(q^0,w) = \{p^1,p^2\ldots,p^k\}$
  - $\delta(p^i,a) = S^i$ for $i=1,2\ldots,k$
  - Then, $\delta(q^0,wa) = S^1 \cup S^2 \cup \ldots \cup S^k$

Language of an NFA

- An NFA accepts *w* if *there exists at least one* path from the start state to an accepting (or final) state that is labeled by *w*
- $L(N) = \{ w \mid \delta(q^0, w) \cap F \neq \Phi \}$

Advantages & Caveats for NFA

- Great for modeling regular expressions
  - String processing - e.g., grep, lexical analyzer
- Could a non-deterministic state machine be implemented in practice?
  - A parallel computer could exist in multiple “states” at the same time
  - Probabilistic models could be viewed as extensions of non-deterministic state machines
    (e.g., toss of a coin, a roll of dice)

DFAs and NFAs are equivalent in their power to capture langauges !!

Differences: DFA vs. NFA

- **DFA**
- All transitions are deterministic
  - Each transition leads to exactly one state
- For each state, transition on all possible symbols (alphabet) should be defined
- Accepts input if the last state is in F
- Sometimes harder to construct because of the number of states
- Practical implementation is feasible

- **NFA**
- Some transitions could be non-deterministic
  - A transition could lead to a subset of states
- Not all symbol transitions need to be defined explicitly (if undefined will go to a dead state – this is just a design convenience, not to be confused with "non-determinism")
- Accepts input if *one of* the last states is in F
- Generally easier than a DFA to construct
- Practical implementation has to be deterministic (convert to DFA) or in the form of parallelism

Summary

- DFA
  - Definition
  - Transition diagrams & tables
- Regular language
- NFA
  - Definition
  - Transition diagrams & tables
- DFA vs. NFA
- NFA to DFA conversion using subset construction

FAQ

1. How a Non deterministic finite state automaton (NFA) differs from a Deterministic finite state automaton (DFA).
2. Write RE which denotes the language L over the set = {a,b} such that all the strings do not contain the substring ab.
3. Give a regular expression for the set of all strings having odd number of 1's
4. Prove that 12 +22+32+……+ n2= $\sum$n2 = (n(n+1)(2n+1))/6.

References

- Https://en.wikipedia.org/wiki/Finite-state_machine
- https://www.safaribooksonline.com
- http://studentsfocus.com/
- http://www.francisxavier.ac.in/

CU
CHANDIGARH UNIVERSITY

THANK YOU

UNIVERSITY INSTITUTE OF ENGINEERING

COMPUTER SCIENCE ENGINEERING

Bachelor of Engineering

Theory of Computation (CST-353)

Topic: Equivalence of Finite Automata

Learning Objectives & Outcomes

Objective:

- To understand the concept of Finite Automata, and equivalence of Finite Automata.

Outcome:

- Student will understand the
  - Equivalence of DFA and NDFA
  - NFA with null transition

Equivalence of DFA & NFA

- Theorem:
  - A language L is accepted by a DFA *if and only if* it is accepted by an NFA.

Should be true for any L

- Proof:
  - If part:
    - Prove by showing every NFA can be converted to an equivalent DFA (in the next few slides…)
  - Only-if part is trivial:
    - Every DFA is a special case of an NFA where each state has exactly one transition for every input symbol. Therefore, if L is accepted by a DFA, it is accepted by a corresponding NFA.

Proof for the if-part

- If-part: A language L is accepted by a DFA if it is accepted by an NFA
- rephrasing…
- Given any NFA N, we can construct a DFA D such that L(N)=L(D)
- How to convert an NFA into a DFA?
  - Observation: In an NFA, each transition maps to a *subset* of states
  - Idea: Represent:

    each "subset of NFA_states" ➔ a single "DFA_state"

NFA to DFA by subset construction

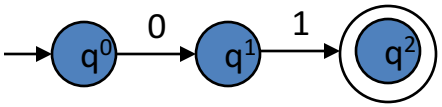
- Let $N = \{Q^N, \sum, \delta^N, q^0, F^N\}$
- Goal: Build $D=\{Q^D, \sum, \delta^D, \{q^0\}, F^D\}$ s.t. $L(D)=L(N)$
- Construction:
  - Q^D= all subsets of Q^N (i.e., power set)
  - F^D=set of subsets S of Q^N s.t. $S \cap F^N \neq \Phi$
  - δ^D: for each subset S of Q^N and for each input symbol a in $\sum$:
    - $\delta^D(S,a) = \bigcup_{p \text{ in } s} \delta^N(p,a)$

NFA to DFA construction: Example

- *L = {w | w ends in 01}*

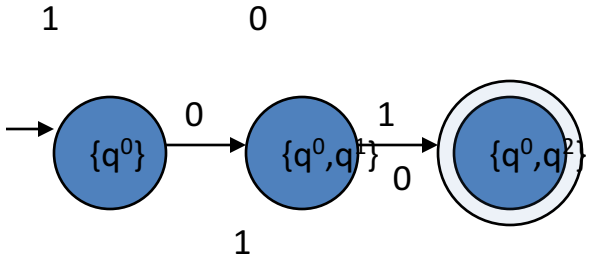
Idea: To avoid enumerating all of power set, do "lazy creation of states"

NFA:



δ^N	0	1
q^0	$\{q^0,q^1\}$	$\{q^0\}$
q^1	Ø	$\{q^2\}$
$*q^2$	Ø	Ø

DFA:



δ^D	0	1
Ø	Ø	Ø
$\{q^0\}$	$\{q^0,q^1\}$	$\{q^0\}$
$\{q^1\}$	Ø	$\{q^2\}$
$*\{q^2\}$	Ø	Ø
$\{q^0,q^1\}$	$\{q^0,q^1\}$	$\{q^0,q^2\}$
$*\{q^0,q^2\}$	$\{q^0,q^1\}$	$\{q^0\}$
$*\{q^1,q^2\}$	Ø	$\{q^2\}$
$*\{q^0,q^1,q^2\}$	$\{q^0,q^1\}$	$\{q^0,q^2\}$

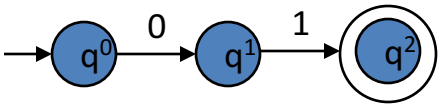
δ^D	0	1
$\{q^0\}$	$\{q^0,q^1\}$	$\{q^0\}$
$\{q^0,q^1\}$	$\{q^0,q^1\}$	$\{q^0,q^2\}$
$*\{q^0,q^2\}$	$\{q^0,q^1\}$	$\{q^0\}$

0. Enumerate all possible subsets
1. Determine transitions
2. Retain only those states reachable from $\{q^0\}$

NFA to DFA: Repeating the example using *LAZY CREATION*

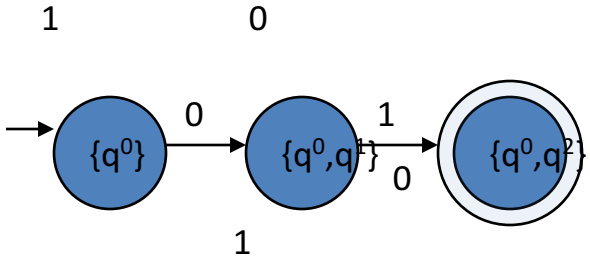
- *L = {w | w ends in 01}*

NFA:



δ^N	0	1
→ q^0	$\{q^0,q^1\}$	$\{q^0\}$
q^1	Ø	$\{q^2\}$
*q^2	Ø	Ø

DFA:



δ^D	0	1
→ $\{q^0\}$	$\{q^0,q^1\}$	$\{q^0\}$
$\{q^0,q^1\}$	$\{q^0,q^1\}$	$\{q^0,q^2\}$
*$\{q^0,q^2\}$	$\{q^0,q^1\}$	$\{q^0\}$

Main Idea: Introduce states as you go(on a need basis)

Correctness of subset construction

Theorem: If D is the DFA constructed from NFA N by subset construction, then L(D)=L(N)

- Proof:
  - Show that $\delta^D(\{q^0\},w) \equiv \delta^N(q^0,w\}$, for all w
  - Using induction on w's length:
    - Let w = xa
    - $\delta^D(\{q^0\},xa) \equiv \delta^D(\hat{\delta}^N(q^0,x\}, a) \equiv \delta^N(\hat{q^0},w\}$

A bad case where #states(DFA)>>#states(NFA)

- L = {w | w is a binary string s.t., the k^{th} symbol from its end is a 1}
  - NFA has k+1 states
  - But an equivalent DFA needs to have at least 2^k states

(Pigeon hole principle)

- *m* holes and >*m* pigeons
  - => at least one hole has to contain two or more pigeons

Applications

- Text indexing
  - inverted indexing
  - For each unique word in the database, store all locations that contain it using an NFA or a DFA
- Find pattern P in text T
  - Example: Google querying
- Extensions of this idea:
  - PATRICIA tree, suffix tree

A few subtle properties of DFAs and NFAs

- The machine never really terminates.
  - It is always waiting for the next input symbol or making transitions.
- The machine decides when to consume the next symbol from the input and when to ignore it.
  - (but the machine can never skip a symbol)
- => A transition can happen even *without* really consuming an input symbol (think of consuming ε as a free token) – if this happens, then it becomes an ε-NFA (see next few slides).
- A single transition *cannot* consume more than one (non-ε) symbol.

FA with ε-Transitions

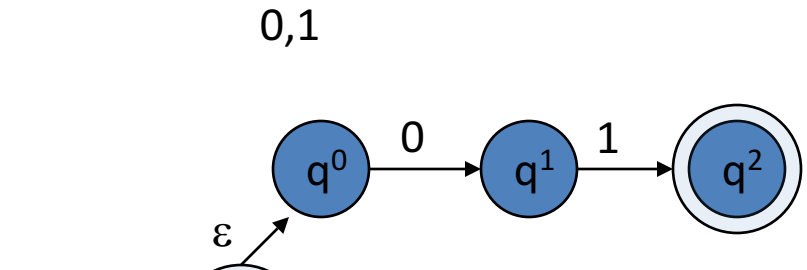
- We can allow <u>explicit</u> ε-transitions in finite automata
  - i.e., a transition from one state to another state without consuming any additional input symbol
  - Makes it easier sometimes to construct NFAs

<u>Definition:</u> ε -NFAs are those NFAs with at least one explicit ε-transition defined.

- ε -NFAs have one more column in their transition table

Example of an ε-NFA

L = {w | w is empty, or if non-empty will end in 01}



δ^E	0	1	ε
$*q'^0$	Ø	Ø	$\{q'^0, q^0\}$
→ q^0	$\{q^0, q^1\}$	$\{q^0\}$	$\{q^0\}$
q^1	Ø	$\{q^2\}$	$\{q^1\}$
$*q^2$	Ø	Ø	$\{q^2\}$

- ε-closure of a state q, ***ECLOSE(q)***, is the set of all states (including itself) that can be *reached* from q by repeatedly making an arbitrary number of ε-transitions.

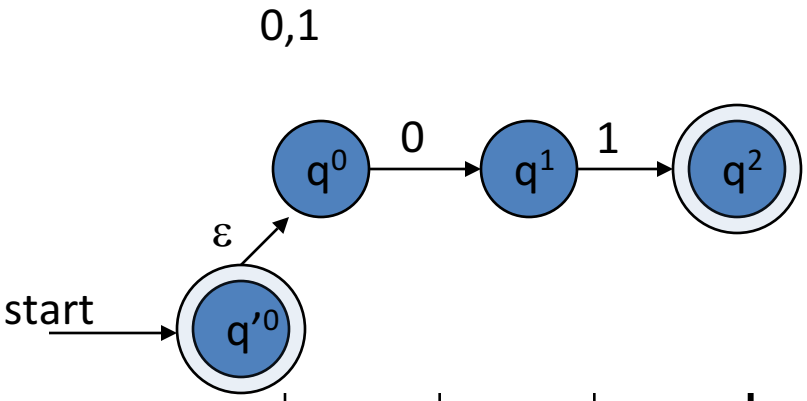
Example of an ε-NFA

To simulate any transition:

Step 1) Go to all immediate destination states.

Step 2) From there go to all their ε-closure states as well.

L = {w | w is empty, or if non-empty will end in 01}

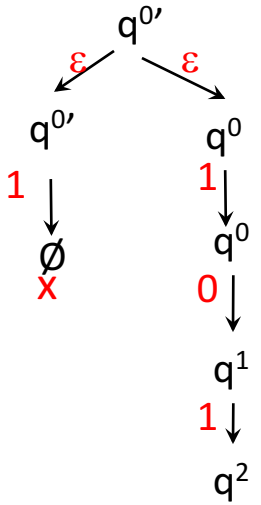


δ^E	0	1	ε
*q'^0	Ø	Ø	{q'^0,q^0}
→ q^0	{q^0,q^1}	{q^0}	{q^0}
q^1	Ø	{q^2}	{q^1}
*q^2	Ø	Ø	{q^2}

ECLOSE(q'^0)

ECLOSE(q^0)

Simulate for w=101:

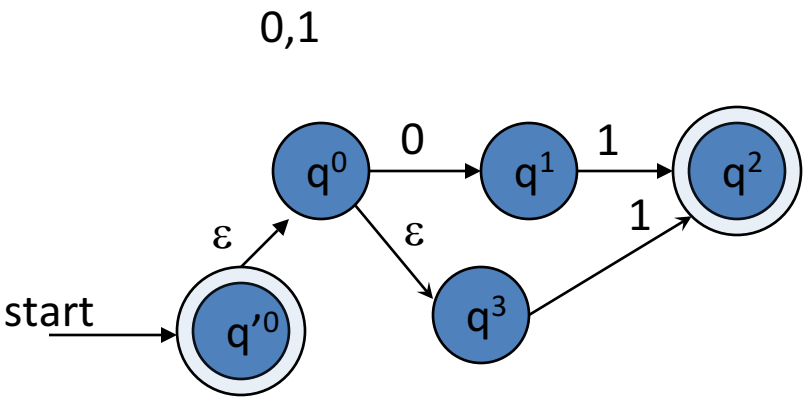


Example of another ε-NFA

To simulate any transition:

Step 1) Go to all immediate destination states.

Step 2) From there go to all their ε-closure states as well.



δ^E	0	1	ε
*q'^0	Ø	Ø	{q'^0,q^0,q^3}
q^0	{q^0,q^1}	{q^0}	{q^0,q^3}
q^1	Ø	{q^2}	{q^1}
*q^2	Ø	Ø	{q^2}
q^3	Ø	{q^2}	{q^3}

Simulate for w=101: ?

Equivalency of DFA, NFA, ε-NFA

- Theorem: A language L is accepted by some ε-NFA if and only if L is accepted by some DFA

- Implication:
  - DFA ≡ NFA ≡ ε-NFA
  - (all accept Regular Languages)

Eliminating ε-transitions

Let $E = \{Q^E, \sum, \delta^E, q^0, F^E\}$ be an ε-NFA

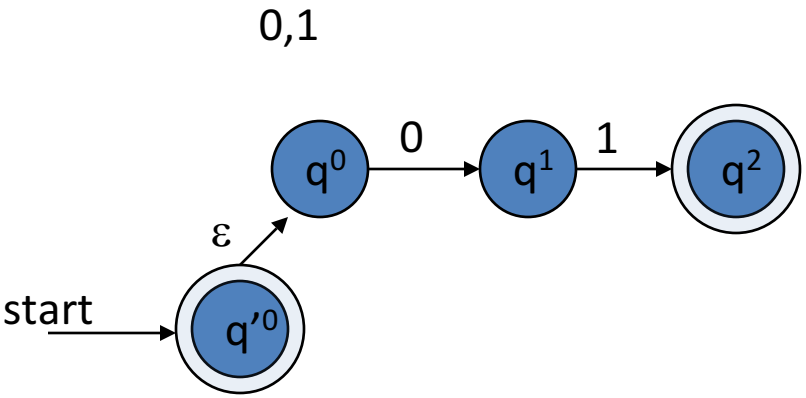
Goal: To build DFA $D=\{Q^D, \sum, \delta^D, \{q^D\}, F^D\}$ s.t. L(D)=L(E)

Construction:

- Q^D= all reachable subsets of Q^E factoring in ε-closures
- q^D = ECLOSE(q^0)
- F^D=subsets S in Q^D s.t. $S \cap F^E \neq \Phi$
- δ^D: for each subset S of Q^E and for each input symbol $a \in \sum$:
  - Let $R = \bigcup_{p \text{ in } s} \delta^E(p,a)$ // go to destination states
  - $\delta^D(S,a) = \bigcup_{r \text{ in } R} \text{ECLOSE}(r)$ // from there, take a union of all their ε-closures

Example: ε-NFA ➔ DFA

L = {w | w is empty, or if non-empty will end in 01}



δ^E	0	1	ε
$*q'^0$	Ø	Ø	$\{q'^0, q^0\}$
q^0	$\{q^0, q^1\}$	$\{q^0\}$	$\{q^0\}$
q^1	Ø	$\{q^2\}$	$\{q^1\}$
$*q^2$	Ø	Ø	$\{q^2\}$

δ^D	0	1
$*\{q'^0, q^0\}$		
...		

Summary

- Equivalency of DFA & NFA
- Removal of redundant states and including dead states
- Ɛ-transitions in NFA
- Pigeon hole principles
- Text searching applications

References

- Https://en.wikipedia.org/wiki/Finite-state_machine
- https://www.safaribooksonline.com
- http://studentsfocus.com/
- http://www.francisxavier.ac.in/

CU
CHANDIGARH UNIVERSITY

THANK YOU

UNIVERSITY INSTITUTE OF ENGINEERING

COMPUTER SCIENCE ENGINEERING

Bachelor of Engineering

Theory of Computation (CST-353)

Topic: Regular Expression

Learning Objectives & Outcomes

Objective:

- To understand the concept of Regular expression.

Outcome:

- Student will understand the
  - Regular Language
  - Regular expression
  - Arden's Theorem

Regular Languages

Definition

•We define regular languages as the smallest class of languages which contains all finite languages and closed with respect to union, concatenation and Kleene closure.

•Every regular expression denotes a regular language.

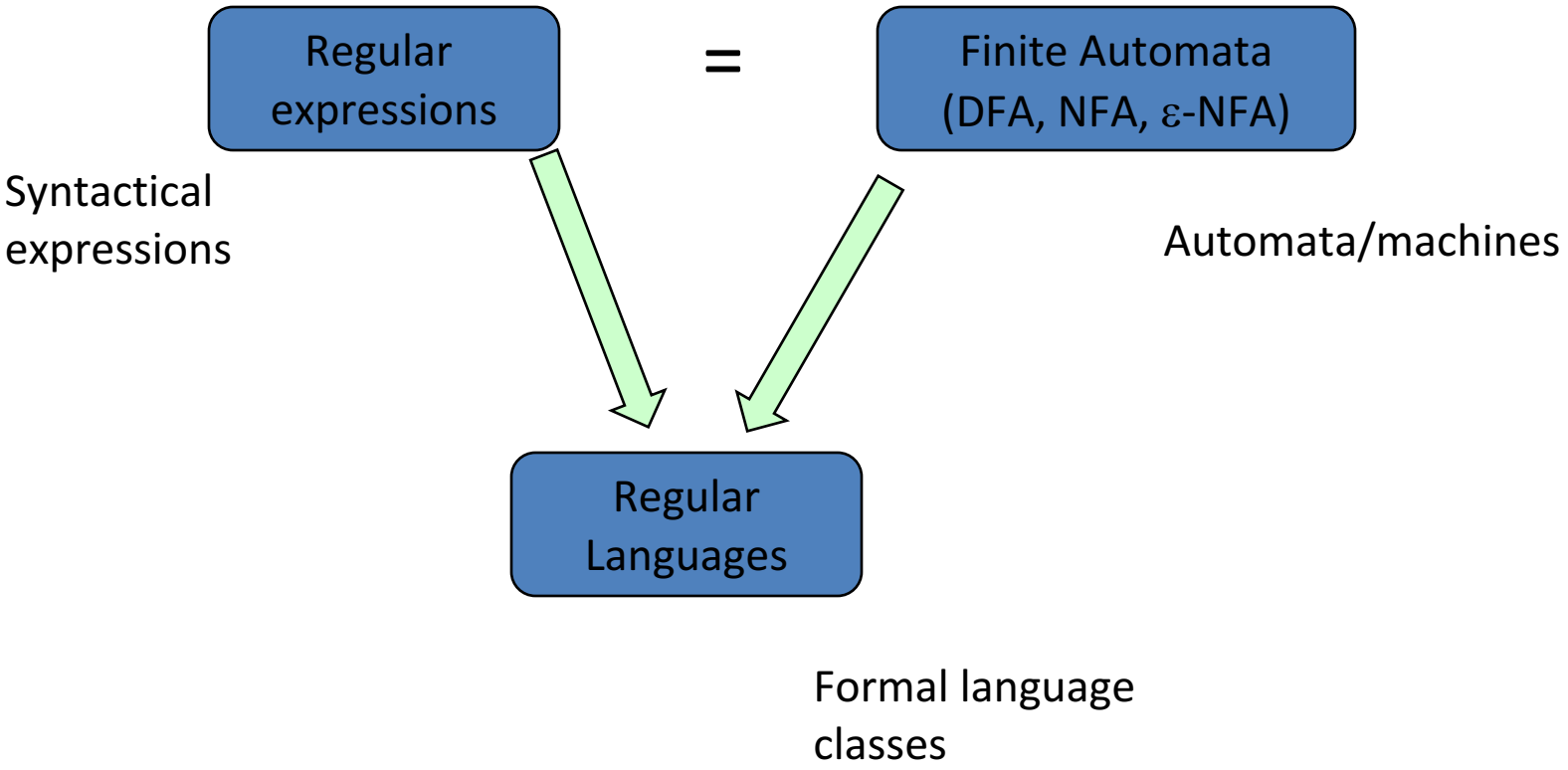
Example

•(a*b) is a regular expression that denotes the set of strings formed by a sequence, also empty, of a's followed by a b.

Regular Expressions vs. Finite Automata

- Offers a declarative way to express the pattern of any string we want to accept
  - E.g., 01*+ 10*
- Automata => more machine-like
  < input: string , output: [accept/reject] >
- Regular expressions => more program syntax-like
- Unix environments heavily use regular expressions
  - E.g., bash shell, grep, vi & other editors, sed
- Perl scripting – good for string processing
- Lexical analyzers such as Lex or Flex

Regular Expressions



Language Operators

- Union of two languages:
  - **L U M** = all strings that are either in L or M
  - Note: A union of two languages produces a third language

- Concatenation of two languages:
  - **L . M** = all strings that are of the form *xy*
    s.t., x ∈ L and y ∈ M
  - The *dot* operator is usually omitted
  - i.e., **LM** is same as L.M

Kleene Closure (the * operator)

- Kleene Closure of a given language L:
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{w \mid \text{for some } w \in L\}$
  - $L^2 = \{ w^1w^2 \mid w^1 \in L, w^2 \in L \text{ (duplicates allowed)}\}$
  - $L^i = \{ w^1w^2\ldots w^i \mid \text{all w's chosen are} \in L \text{ (duplicates allowed)}\}$
  - (Note: the choice of each w^i is independent)
  - **$L^* = \bigcup_{i \geq 0} L^i$** (arbitrary number of concatenations)

Example:

- Let L = { 1, 00}
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{1,00\}$
  - $L^2 = \{11,100,001,0000\}$
  - $L^3 = \{111,1100,1001,10000,000000,00001,00100,0011\}$
  - **$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$**

Kleene Closure (special notes)

- L^* is an infinite set iff $|L| \geq 1$ and $L \neq \{\varepsilon\}$
- If $L = \{\varepsilon\}$, then $L^* = \{\varepsilon\}$
- If $L = \Phi$, then $L^* = \{\varepsilon\}$

Σ^* denotes the set of all words over an alphabet Σ

- Therefore, an abbreviated way of saying there is an arbitrary language L over an alphabet Σ is:
  - $L \subseteq \Sigma^*$

Building Regular Expressions

- Let E be a regular expression and the language represented by E is L(E)
- Then:
  - (E) = E
  - L(E + F) = L(E) U L(F)
  - L(E F) = L(E) L(F)
  - L(E*) = (L(E))*

Example: how to use these regular expression properties and language operators?

- L = { w | w is a binary string which does *not* contain two consecutive 0s or two consecutive 1s anywhere)
  - E.g., w = 01010101 is in L, while w = 10010 is not in L
- Goal: Build a regular expression for L
- Four cases for w:
  - Case A: w starts with 0 and |w| is even
  - Case B: w starts with 1 and |w| is even
  - Case C: w starts with 0 and |w| is odd
  - Case D: w starts with 1 and |w| is odd
- Regular expression for the four cases:
  - Case A: (01)*
  - Case B: (10)*
  - Case C: 0(10)*
  - Case D: 1(01)*
- Since L is the union of all 4 cases:
  - Reg Exp for L = (01)* + (10)* + 0(10)* + 1(01)*
- If we introduce ε then the regular expression can be simplified to:
  - Reg Exp for L = (ε +1)(01)*(ε +0)

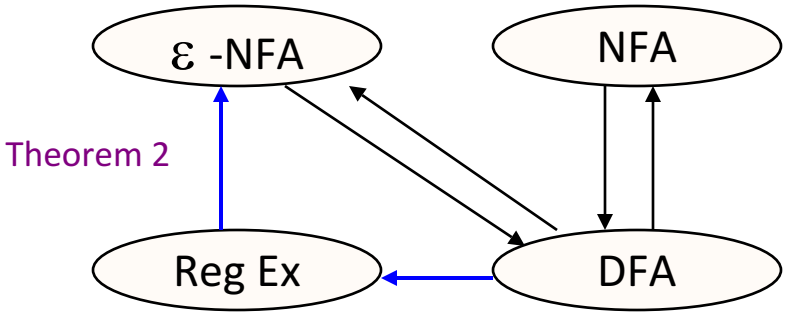
Precedence of Operators

- Highest to lowest
  - * operator (star)
  - . (concatenation)
  - + operator
- Example:
  - 01* + 1 =(0 . ((1)*)) + 1

Finite Automata (FA) & Regular Expressions (Reg Ex)

- To show that they are interchangeable, consider the following theorems:
  - *Theorem 1: For every DFA A there exists a regular expression R such that L(R)=L(A)*
  - *Theorem 2: For every regular expression R there exists an ε -NFA E such that L(E)=L(R)*

Proofs in the book

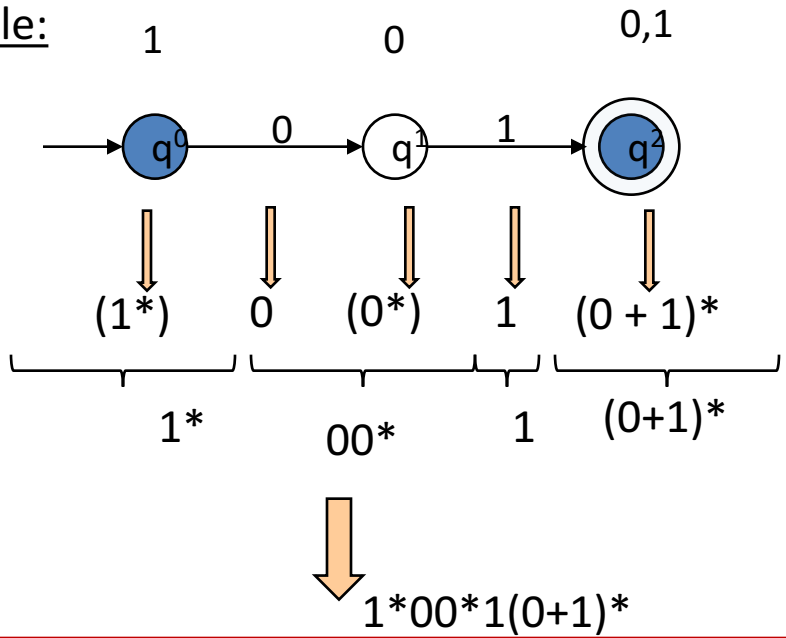


Kleene Theorem

DFA to RE construction

Informally, trace all distinct paths (traversing cycles only once)
from the start state to *each of the* final states
and enumerate all the expressions along the way

Example:



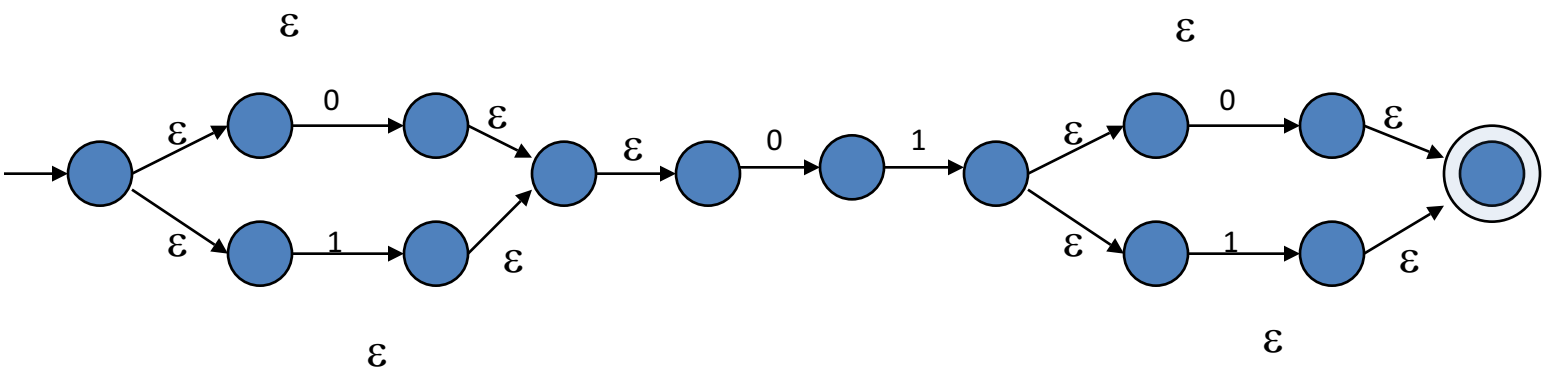
RE to ε-NFA construction

Reg Ex → ε -NFA

Theorem 2

Example: (0+1)*01(0+1)*

(0+1)* 01 (0+1)*



Algebraic Laws of Regular Expressions

- Commutative:
  - E+F = F+E
- Associative:
  - (E+F)+G = E+(F+G)
  - (EF)G = E(FG)
- Identity:
  - E+Φ = E
  - ε E = E ε = E
- Annihilator:
  - ΦE = EΦ = Φ

Algebraic Laws…

- Distributive:
  - E(F+G) = EF + EG
  - (F+G)E = FE+GE
- Idempotent: E + E = E
- Involving Kleene closures:
  - $(E^*)^*$ = E^*
  - Φ^* = ε
  - ε^* = ε
  - E^+ =EE^*
  - E? = ε +E

True or False?

Let R and S be two regular expressions. Then:

- ((R*)*)* = R* ?
- (R+S)* = R* + S*?
- (RS + R)* RS = (RR*S)*?

Summary

- Regular expressions
- DFA to regular expression conversion
- Regular expression to ε-NFA conversion
- Algebraic laws of regular expressions.

FAQ

1. Design FA which accepts even number of 0's and even no of 1's.
2. Prove that for every integer $n \geq 0$ the number $42n+1+ 3 n+2$ is multiple of 13
3. Prove that $6n \equiv 0 \pmod 9$ for all integers $n \geq 2$.

References

- K.L.P MISHRA,"*Introduction to computer Theory*" .
- https://youtu.be/yKOmggxrYmY
- https://youtu.be/cFd3pz5H-TA
- https://www.tutorialspoint.com/automata_theory/ardens_theorem.htm

CU
CHANDIGARH UNIVERSITY

THANK YOU

UNIVERSITY INSTITUTE OF ENGINEERING

COMPUTER SCIENCE ENGINEERING

Bachelor of Engineering

Theory of Computation (CST-353)

Topic: Kleen's Theorem

Learning Objectives & Outcomes

Objective:

- To understand the concept of Kleene's Theorem.

Outcome:

- Student will understand the
  - Kleene's Theorem

Kleene's Theorem

Theorem 1 (Part 1 of Kleene's theorem): Any regular language is accepted by a finite automaton.

Proof: This is going to be proven by (general) induction following **the recursive definition of regular language.**

Inductive Step

- **Inductive Step:** We are going to show that for any languages L^1 and L^2 if they are accepted by FAs, then $L^1 \cup L^2$, L^1L^2 and L^{1*} are accepted by FAs. Since any regular language is obtained from { λ } and { a } for any symbol a in Σ by using union, concatenation and Kleene star operations, that together with the Basis Step would prove the theorem.

Inductive Step (Continue..)

- Suppose that L^1 and L^2 are accepted by FAs $M^1 = < Q^1 , \Sigma , q^{1,0} , \delta^1 , A^1 >$ and $M^2 = < Q^2 , \Sigma , q^{2,0} , \delta^2 , A^2 >$, respectively. We assume that $Q^1 \cap Q^2 = \emptyset$ without loss of generality since states can be renamed if necessary.
  Then $L^1 \cup L^2$, $L^1 L^2$ and L^{1*} are accepted by the FAs $M^u = < Q^u , \Sigma , q^{u,0} , \delta^u , A^u >$, $M^c = < Q^c , \Sigma , q^{c,0} , \delta^c , A^c >$ and $M^k = < Q^2 , \Sigma , q^{k,0} , \delta^k , A^k >$, respectively, which are given below.

  $M^u = < Q^u , \Sigma , q^{u,0} , \delta^u , A^u >$:

  $Q^u = Q^1 \cup Q^2 \cup \{ q^{u,0} \}$, where $q^{u,0}$ is a state which is neither in Q^1 nor in Q^2.
  $\delta^u = \delta^1 \cup \delta^2 \cup \{ (q^{u,0}, \lambda , \{ q^{1,0} , q^{2,0} \}) \}$, that is $\delta^u(q^{u,0}, \lambda) = \{ q^{1,0} , q^{2,0} \}$. Note that $\delta^u(q^{u,0}, a) = \emptyset$ for all a in Σ.
  $A^u = A^1 \cup A^2$

Inductive Step (Continue..)

$\mathbf{M^c = < Q^c , \Sigma , q^{c,0} , \delta^c , A^c > :}$

$Q^c = Q^1 \cup Q^2$

$q^{c,0} = q^{1,0}$

$\delta^c = \delta^1 \cup \delta^2 \cup \{ (q, \Lambda , \{ q^{2,0} \}) \mid q \in A^1 \}$

$A^c = A^2$

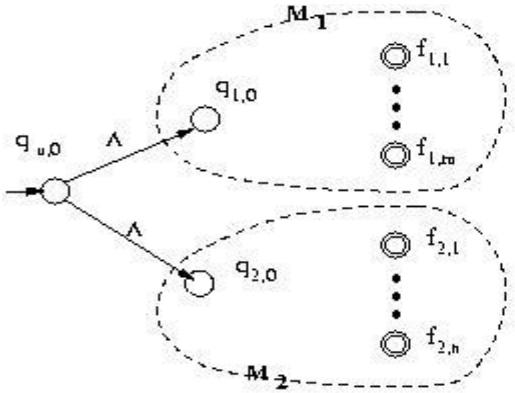
Inductive Step (Continue..)

$\mathbf{M^k = < Q^k , \Sigma , q^{k,0} , \delta^k , A^k > :}$

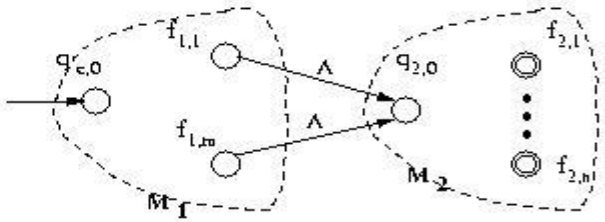
$Q^k = Q^1 \cup \{ q^{k,0} \}$, where $q^{k,0}$ is a state which is not in Q^1 .
$\delta^k = \delta^1 \cup \{ (q^{k,0}, \lambda , \{ q^{1,0} \}) \} \cup \{ (q, \lambda , \{ q^{k,0} \}) \mid q \in A^1 \}$
$A^k = \{ q^{k,0} \}$

These NFA- λ s are illustrated below.

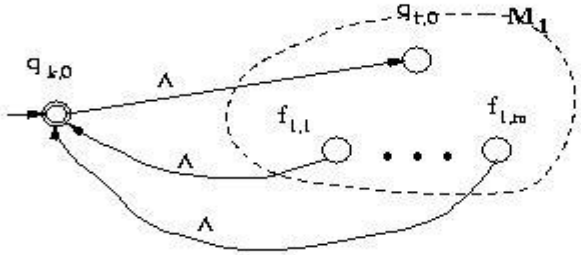
It can be proven, though we omit proofs, that these NFA- Λ s , M^u, M^c and M^k , in fact accept $L^1 \cup L^2$, L^1L^2 and L^{1*}, respectively. End of Proof



Union of FAs M_u



Concatenation of FAs M_c



Kleene Star of FA M_k

Kleene's Theorem Part-2

Kleene's Theorem Part-2

In this section we prove that if L is accepted by a finite automaton, then L is regular. The proof will provide an algorithm for starting with an FA that accepts L and finding a regular expression that describes L.

Theorem 3.30 Kleene's Theorem, Part 2

For every finite automaton $M = (Q, \Sigma, q_0, A, \delta)$, the language $L(M)$ is regular.

Proof

For states p and q, we introduce the notation $L(p, q)$ for the language

$$L(p, q) = \{x \in \Sigma^* \mid \delta^*(p, x) = q\}$$

If we can show that for every p and q in Q, $L(p, q)$ is regular, then it will follow that $L(M)$ is, because

$$L(M) = \bigcup \{L(q_0, q) \mid q \in A\}$$

and the union of a finite collection of regular languages is regular.

We will show that each language $L(p, q)$ is regular by expressing it in terms of simpler languages that are regular. Strings in $L(p, q)$ cause M to move from p to q in any manner whatsoever. One way to think about simpler ways of moving from p to q is to think about the number of transitions involved; the problem with this approach is that there is no upper limit to this number, and so no obvious way to obtain a final regular expression. A similar approach that is more promising is to consider the distinct states through which M passes as it moves from p to q. We can start by considering how M can go from p to q without going through any states, and at each step add one more state to the set through which M is allowed to go. This procedure will terminate when we have enlarged the set to include all possible states.

If $x \in L(p, q)$, we say x causes M to go from p to q *through* a state r if there are nonnull strings x_1 and x_2 such that $x = x_1x_2$, $\delta^*(p, x_1) = r$, and $\delta^*(r, x_2) = q$. In using a string of length 1 to go from p to q, M does not go through any state. (If M loops from p back to p on the symbol a, it does not go through p even though the string a causes it to leave p and enter p.) In using a string of length $n \geq 2$, it goes through a state $n - 1$ times, but if $n > 2$ these states may not be distinct.

Now we assume that Q has n elements and that they are numbered from 1 to n. For $p, q \in Q$ and $j \geq 0$, we let $L(p, q, j)$ be the set of strings in $L(p, q)$ that cause M to go from p to q without going through any state numbered higher than j.

The set $L(p, q, 0)$ is the set of strings that allow M to go from p to q without going through any state at all. This includes the set of alphabet symbols σ for which $\delta(p, \sigma) = q$, and in the case when $p = q$ it also includes the string Λ. In any case, $L(p, q, 0)$ is a finite set of strings and therefore regular.

Suppose that for some number $k \geq 0$, $L(p, q, k)$ is regular for every p and every q in Q, and consider how a string can be in $L(p, q, k + 1)$. The easiest way is for it to be in $L(p, q, k)$, because if M goes through no state numbered higher than k, it certainly goes through no state numbered higher than $k + 1$. The other strings in $L(p, q, k + 1)$ are those that cause M to go from p to q by going through state $k + 1$ and no higher-numbered states. A path of this type goes from p to $k + 1$; it may return to $k + 1$ one or more times; and it finishes by going from $k + 1$ to q (see Figure 3.31). On each of these individual portions, the path starts or stops at state $k + 1$ but doesn't go through any state numbered higher than k.

Every string in $L(p, q, k + 1)$ can be described in one of these two ways, and every string that has one of these two forms is in $L(p, q, k + 1)$. The resulting formula is

$$L(p, q, k + 1) = L(p, q, k) \cup L(p, k + 1, k)L(k + 1, k + 1, k)^* \\ L(k + 1, q, k)$$

We have the ingredients, both for a proof by mathematical induction that $L(p, q)$ is regular and for an algorithm to obtain a regular expression for this language. $L(p, q, 0)$ can be described by a regular expression; for each $k < n$, $L(p, q, k + 1)$ is described by the formula above; and $L(p, q, n) = L(p, q)$, because the condition that the path go through no state numbered higher than n is no restriction at all if there are no states numbered higher than n. As we observed at the beginning of the proof, the last step in obtaining a regular expression for $L(M)$ is to use the $+$ operation to combine the expressions for the languages $L(q_0, q)$, where $q \in A$.

Summary

- Kleene's Theorem
  - Part 1
  - Part 2

References

- Martin J.C., "Introduction *to Languages and Theory of Computation*", Tata McGraw-Hill Publishing Company Limited, 3rd Edition
- https://www.geeksforgeeks.org/kleenes-theorem-in-toc-part-1/
- https://www.cs.odu.edu/~toida/nerzic/390teched/regular/fa/kleene-1.html#:~:text=It%20states%20that%20any%20regular,by%20an%20FA%20is%20regular

CU
CHANDIGARH UNIVERSITY

THANK YOU

UNIVERSITY INSTITUTE OF ENGINEERING

COMPUTER SCIENCE ENGINEERING

Bachelor of Engineering

Theory of Computation (CST-353)

Topic: Arden's Theorem

Learning Objectives & Outcomes

Objective:

- To understand the concept of Arden's Theorem.

Outcome:

- Student will understand the
  - Arden's Theorem

Arden's Theorem

In order to find out a regular expression of a Finite Automaton, we use Arden's Theorem along with the properties of regular expressions.

Statement –

- Let **P** and **Q** be two regular expressions.
- If **P** does not contain null string, then **R = Q + RP** has a unique solution that is **R = QP***

PROOF

$R = Q + (Q + RP)P$ [After putting the value $R = Q + RP$]

$= Q + QP + RPP$

When we put the value of **R** recursively again and again, we get the following equation −

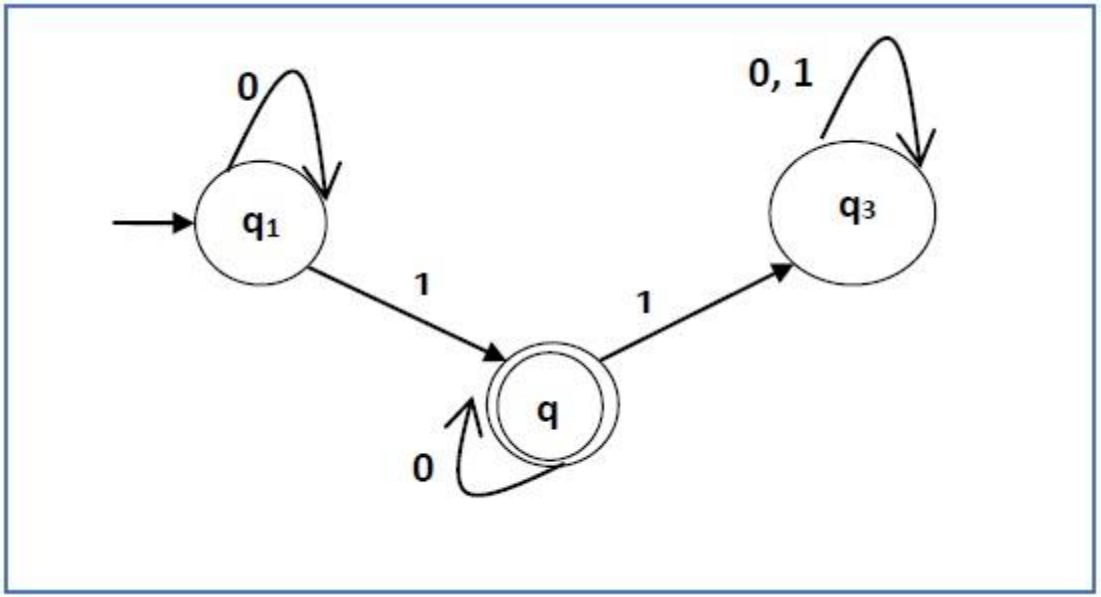
$R = Q + QP + QP^2 + QP^3 \ldots..$

$R = Q (\varepsilon + P + P^2 + P^3 + \ldots.)$

$R = QP^*$ [As P^* represents $(\varepsilon + P + P2 + P3 + \ldots.)$]

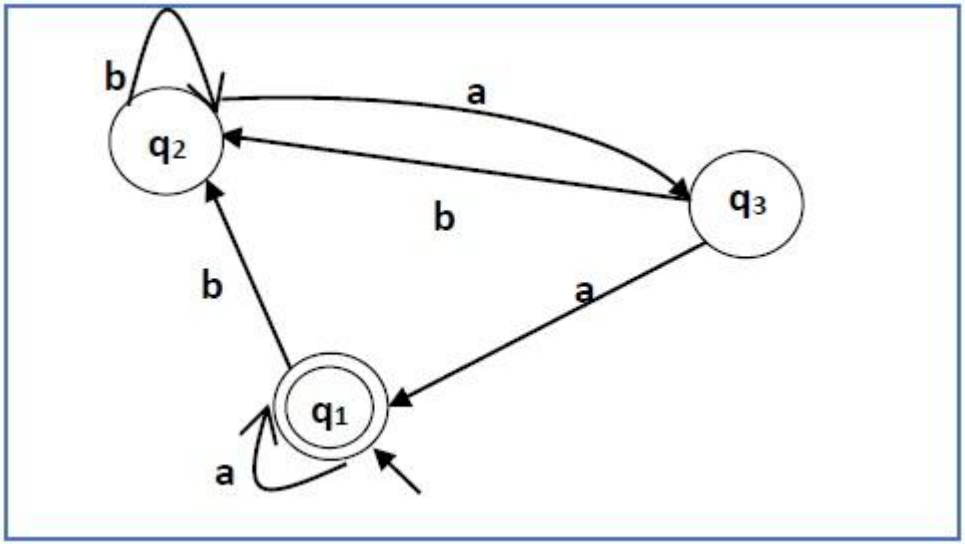
Hence, proved.

Construct a regular expression corresponding to the automata given below –



- **Solution –**
- Here the initial state is q^1 and the final state is q^2
- Now we write down the equations –
- $q^1 = q^10 + \varepsilon$
- $q^2 = q^11 + q^20$
- $q^3 = q^21 + q^30 + q^31$
- Now, we will solve these three equations –
- $q^1 = \varepsilon 0^*$ [As, $\varepsilon R = R$]
- So, $q^1 = 0^*$
- $q^2 = 0^*1 + q^20$
- So, $q^2 = 0^*1(0)^*$ [By Arden's theorem]
- Hence, the regular expression is 0^*10^*.

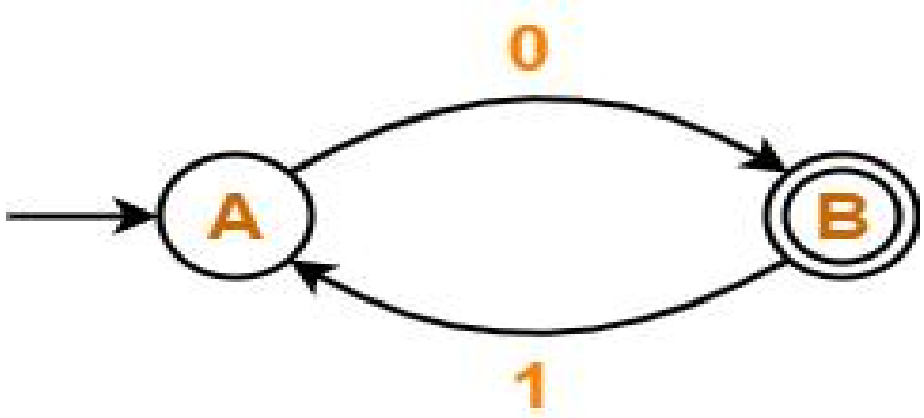
Construct a regular expression corresponding to the automata given below –



- **Solution** –
- Here the initial state and final state is $\mathbf{q^1}$.
- The equations for the three states q1, q2, and q3 are as follows –
- $q^1 = q^1a + q^3a + \varepsilon$ (ε move is because q1 is the initial state0
- $q^2 = q^1b + q^2b + q^3b$
- $q^3 = q^2a$
- Now, we will solve these three equations –
- $q^2 = q^1b + q^2b + q^3b$
- $= q^1b + q^2b + (q^2a)b$ (Substituting value of q^3)
- $= q^1b + q^2(b + ab)$
- $= q^1b\ (b + ab)^*$ (Applying Arden's Theorem)
- $q^1 = q^1a + q^3a + \varepsilon$

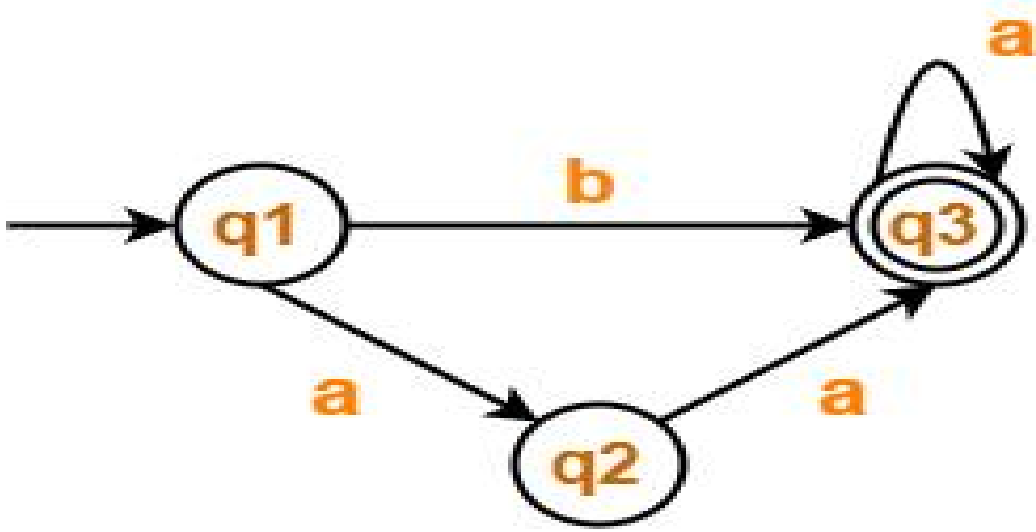
- $q^1a + q^2aa + \varepsilon$ (Substituting value of q^3)
- $= q^1a + q^1b(b + ab^*)aa + \varepsilon$ (Substituting value of q^2)
- $= q^1(a + b(b + ab)^*aa) + \varepsilon$
- $= \varepsilon\ (a+ b(b + ab)^*aa)^*$
- $= (a + b(b + ab)^*aa)^*$
- Hence, the regular expression is $(a + b(b + ab)^*aa)^*$.

Find regular expression for the following DFA using Arden's Theorem-



- **Solution-**
-
- **Step-01:**
-
- Form a equation for each state-
- A = ∈ + B.1 ……(1)
- B = A.0 ……(2)
-
- **Step-02:**
-
- Bring final state in the form R = Q + RP.
-
- Using (1) in (2), we get-
- B = (∈ + B.1).0
- B = ∈.0 + B.1.0
- B = 0 + B.(1.0) ……(3)
-
- Using Arden's Theorem in (3), we get-
- B = 0.(1.0)*
-
- Thus, Regular Expression for the given DFA = 0(10)*

Find regular expression for the following DFA using Arden's Theorem-



- **Solution-**
- **Step-01:**
- Form a equation for each state-
- q1 = ∈ ……(1)
- q2 = q1.a ……(2)
- q3 = q1.b + q2.a + q3.a …….(3)
- **Step-02:**
- Bring final state in the form R = Q + RP.
- Using (1) in (2), we get-
- q2 = ∈.a
- q2 = a …….(4)
- Using (1) and (4) in (3), we get-
- q3 = q1.b + q2.a + q3.a
- q3 = ∈.b + a.a + q3.a
- q3 = (b + a.a) + q3.a …….(5)
- Using Arden's Theorem in (5), we get-
- q3 = (b + a.a)a*

- Thus, Regular Expression for the given DFA = (b + aa)a*

Summary

- Arden's Theorem
  - Conversion of FA to R.E

References

- K.L.P MISHRA,"*Introduction to computer Theory*" .
- https://www.tutorialspoint.com/automata_theory/ardens_theorem.htm

CU
CHANDIGARH UNIVERSITY

THANK YOU