

Lecture notes on Automata Theory and Computability(subject code: 15CS54) – Module -1: **By Prof B I Khodanpur, DSCE**

Module – 1: Syllabus:-

Why study the theory of computation(ch-1)

Languages and strings(ch-2)

A Language Hierarchy(ch-3)

Computation(ch-4)

Finite State Machines(ch-5 from 5.1 to 5.10)

Why study the theory of computation(ch-1)

Defn: Automata is an abstract machine for modelling computations.

Why Abstract machines?

Abstract machine allows us to model the essential parameters, and ignore the non-essential parameters.

What is computability?

It is very difficult to define, but Our notion of computation: Examples are

Add 2 numbers

Find the roots of a quadratic equation

Multiply 2 matrices

And so on.....

Important to note that: all the above have algorithms

What is not computable: Example-

- **Halting problem of a program:**

simply write a program that examines other programs to determine if they halt or loop forever. Obviously whether or not a program halts depends on the data it is fed so in this case we mean program to be code plus the data it operates on.

- **Why it not computable:**

simple answer – **No algorithm exists**

Some computations take lot of time to be meaning full: Example

Travelling salesman problem

- **When computations are not finished within a reasonable time, such computations are useless, also known as NP-problem(non-deterministic polynomial problems)**

Tractable/Intractable Problems:

Tractable Problem: a problem that is solvable by a polynomial-time algorithm.

The upper bound is polynomial. Examples: Quick sort($O(n\log n)$)

Intractable Problem: a problem that cannot be solved by a polynomial-time algorithm. The lower bound is exponential. Examples: Travelling Salesman problem

Some important applications of automata theory

- **In compilers:**
 - Lexical analysis
 - Parser generators
- **Modelling the circuits:** Example On-Off switch.



Some important applications of automata theory in general:

Word search and Translation of Natural Languages

Parity checkers, Vending machines, communication protocols

Video games

DNA

Security

Artificial Intelligence

To model organic structures of molecules

Fluid Flow

Snowflake and crystal formation

Chaos theory

Cosmology

Financial analysis

Why not use English to Program?

- Firstly all Natural Languages like English, Kannada etc are **Context Sensitive Languages**
- **That is to say – meaning depends on the context.**
- Example: Take a English word “ Charge “
- There are many meanings for this word
- Like - Cost, -Flight, - Charge the Battery
- - Positive Charge, etc

Characteristics of Natural Languages:

- In most of the situations – meaning depends on the context.
- They are developed for communication among the Human beings.
- Human beings are capable or trained to interpret a sentence depending on the situations.
- **Where as, Machine are not in Context.**
- **Machine will not be able to interpret depending on the situation.**

Characteristics of Formal Languages:

- Meaning of a word or sentence does not depend on the context.
- Words and sentences have only one meaning irrespective of the context.
- They are simple.
- **Easy to write Compilers and Interpreters**
- **They are precise in their meaning.**
- **With this Machine do what they are instructed to do**

What is the gist of this subject?

A systematic way of depicting the problem so that its solution can be understood and analysed.

What are the properties of various types of languages.

Regular Languages(RL)

Context Free languages(CFL)

Context Sensitive Languages(CSL)

Recursively Enumerable Languages(REL)

Various types of Automata will be studied:

- **There are different types of automata for recognizing different languages**
- Deterministic Finite Automata - RL
- Pushdown Automata – CFL

- Linear Bounded Automata – CSL
- Turing Machine – REL

How to study:

- Subject is mathematical and lot of logical thinking is required.
- There are number of Theorems and proofs.
- Understand the definition – mathematically i.e. Examples are not substitute for definitions.
- Examples are only to make the definition clear.
- Work out number of problems from various other books.
- Key to understanding this subject – attempt to work harder problems even if you are not able get answers.
- **If you plan to take up - Gate examination for PG studies – you must understand it thoroughly.**

Languages and Strings(chapter-2)

Alphabet - Σ definition:

Defn: An alphabet is a non-empty, finite set of characters/symbols

Use Σ to denote an alphabet set

Examples

$$\Sigma = \{ a, b \}$$

$$\Sigma = \{ 0, 1, 2 \}$$

$$\Sigma = \{ a, b, c, \dots, z, A, B, \dots, Z \}$$

$$\Sigma = \{ \#, \$, *, @, \& \}$$

String definition: A *string* is a finite sequence, possibly empty, of characters drawn from some alphabet Σ .

ϵ is the empty string

Σ^* is the set of all possible strings over an alphabet Σ .

Examples of strings:

$$\Sigma = \{ a, b \}$$

Strings derived from Σ are.....

..... ϵ , a, b, aa, ab, ba, bb, aaa, aab, aba, ..

$$\Sigma = \{ 0, 1 \}$$

Strings derived from Σ are.....

..... ϵ , 0, 1, 00, 01, 10, 11, 000, 001, 010, ..

$$\Sigma = \{ a \}$$

Strings derived from Σ are.....

..... ϵ , a, aa, aaa, aaaa, aaaaa, aaaaaa,.....

Functions on Strings

Length – to find the length of a string Operator used $|$ $|$

Concatenation – to join two or more strings. Operator - $s|t$, or nothing i.e. st

Replication – strings raised to some power. Operator - a^3

Reversal – reverse a string

Operator - $(w)^R$

Examples of Length of a string

- $|\epsilon| = 0$
- $|101| = 3$
- $|VTU_Edusat| = 10$

Examples of Concatenation of a string

- $x = \text{good}, y = \text{student}$
- Concatenation operation $x|y$ or xy
- $xy = \text{goodstudent}$

Examples of Replication of a string

- $a^3 = \text{aaa}$
- $(\text{good})^3 = \text{goodgoodgood}$
- $a^0 b^3 = \epsilon bbb = \text{bbb}$

Examples of Reversal of a string

- $(abc)^R = cba$
- $x = ab, y = cd, (xy)^R = dcba$
- $x^R y^R = \text{badc}$

Relation on Strings

- Substring:
- aaa is substring of aaa and also $aaabbccc$
- Proper substring:

Defn: A string s is a proper substring of a string t iff s is a substring of t and $s \neq t$

Examples:

$S = \text{good}$ then proper substrings are ..

..... ϵ, g, go, goo only

Prefix and Suffix functions

- A string s is a prefix of t iff $\exists x \in \Sigma^*(t = sx)$
- ϵ, a, ab, abb are prefixes of string abb
- Proper prefix:
- $\epsilon, a, ab,$ are proper prefixes of string abb
- A string s is a suffix of t iff $\exists x \in \Sigma^*(t = xs)$
- ϵ, b, bb, abb are suffixes of string abb
- Proper suffix:
- $\epsilon, b, bb,$ are proper suffixes of string abb

Languages:

Defn: A language is (finite or infinite) set of strings over a finite alphabet Σ

Example if $\Sigma = \{ a \}$ following languages can be derived

- Language L1 = $\{ a, aaa, aaaaa, aaaaaaa, \dots \}$
- Language L2 = $\{ \epsilon, aa, aaaa, aaaaaa, \dots \}$
- Language L3 = $\{ a, aaaaa, aaaaaaaaa, \dots \}$
- Language L4 = $\{ a, aaa, a^7, a^9, a^{13}, \dots \}$

Note: number of languages that can be derived even from single alphabet set is INFINITE

Techniques for defining Languages by enumeration/defining property

Examples: (by enumeration)

- Let $L = \{ w \in \{ a, b \}^* : \text{all string begin with } a \}$
- $L = \{ a, ab, aab, abbbb, \dots \}$
- Strings not in L are:
- $\{ b, ba, \epsilon, bbbbb, baaaaa, \dots \}$
- Let $L = \{ w \in \{ a \}^* : |w| \text{ is even} \}$
- $L = \{ \epsilon, aa, aaaa, aaaaaa, aaaaaaaa, \dots \}$
- Strings not in L are:
- $\{ a, aaa, aaaaa, aaaaaa, \dots \}$ // odd no of a's

Examples: (defining property)

- Let $L = \{ w \in \{ a, b \}^* : \text{all string ending in } a \}$
- $L = \{ a, aba, aaba, bbbba, \dots \}$
- Strings not in L are:
- $\{ b, bb, \epsilon, bbbbb, aaaaaab, \dots \}$
- Let $L = \{ w \in \{ a \}^* : |w| \bmod 3 = 1 \}$
- $L = \{ a, a^4, a^7, a^{10}, \dots \}$
- Strings not in L are:
- $\{ \epsilon, a^2, a^3, a^5, a^6, a^8, a^9, \dots \}, \dots \}$

Functions on Languages.

Languages are sets. Therefore, all set operations like Union, Intersection, Difference, and Complement can be applied.

- Example if $\Sigma = \{ a \}$
- $L1 = \{ \epsilon, a^2, a^4, a^6, a^8, a^{10}, a^{12}, \dots \}$ // even no of a's
- $L2 = \{ a^1, a^3, a^5, a^7, a^9, a^{11}, \dots \}$ // odd no of a's

Set Operations on Languages

- $L1 = \{ \epsilon, a^2, a^4, a^6, a^8, a^{10}, a^{12}, \dots \}$ // even no of a's
- $L2 = \{ a^1, a^3, a^5, a^7, a^9, a^{11}, \dots \}$ // odd no of a's
- $L1 \cup L2 = \Sigma^*$ or $\{ a \}^*$ // union operation
- $L1 \cap L2 = \Phi$ or $\{ \}$ // intersection operation
- $L1 - L2 = L1$ // difference operation

- $L2 - L1 = L2$ // difference operation
- $\sim(L1 - L2) = L2$ // complement operation
- $\sim(L2 - L1) = L1$ // complement operation

Concatenation of Languages

- $L1 = \{aa, ab\}$
- $L2 = \{xx, yy\}$
- $L1L2 = \{aaxx, aayy, abxx, abyy\}$

Some important results

- $L1 = \{ \} = \Phi$
- $L2 = \{xx, yy\}$
- $L1L2 = \{ \}$
- In general for L

$$L \Phi = \Phi L = \Phi$$

Some important results

- $L1 = \{\epsilon\}$
- $L2 = \{xx, yy\}$
- $L1L2 = L2$
- In general for all L
- $L \{\epsilon\} = L \{\epsilon\} = L$
- $(L1L2)L3 = L1(L2L3)$ // associative
- $L1 = \{a^n \mid n \geq 0\}$
- $L2 = \{b^n \mid n \geq 0\}$
- $L1L2 = \{a^n b^m \mid n, m \geq 0\} = a^* b^*$ // note n & m
- Kleene star operation
- $L^* = \{ \text{set of all strings that can be formed by concatenating zero or more strings from L} \}$
- $a^* = \{\epsilon, a, aa, aaa, aaaa, aaaaa, \dots \text{infinite}\}$

What is L^+ ?

- $L^+ = LL^*$ // assuming L does not have ϵ
- $L^+ = L^* - \{\epsilon\}$

Example

$$a^* = \{\epsilon, a, aa, aaa, aaaa, aaaaa, \dots \text{infinite}\}$$

$$a^+ = a^* - \{\epsilon\}$$

Assigning Meaning to the strings of a Language

Following codes of C/Java have the same meaning.

- `int x=4; x++;`
- `int x=4; ++x;`
- `int x=4; x=x+1;`
- `int x=4; x=x-(-1)`

chapter-5

Finite State Machines(FSM)

Defn: A FSM(DFSM) , M is a quintuple:

$(K, \Sigma, \delta, s, A)$

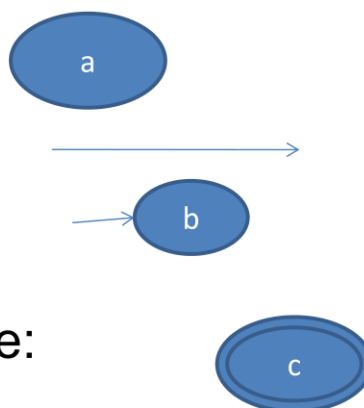
- K is a finite set of states,
- Σ is the input alphabet,
- $s \in K$ is the start state
- A subset of K is the set of accepting states and
- δ is the transition function it maps from:
 $k \times \Sigma$ to k

Finite State Machines(FSM)

- How to draw the Transition diagram of FSM

Notations used:

- Notation for state
- Notation of transition:
- Notation for start state:
- Notation for accepting state:
(note 2 concentric circles)



Finite State Machines(FSM)

On any input if FSM reaches any of the states of A, i.e. accepting states, then the input strings is accepted by FSM M.

Examples:

- Problem_1: Write a FSM to accept L, where
- $L = \{w \in \{a,b\}^* \mid w \text{ contains } a\}$
- $L = \{a, aa, aaa, baa, baaabbb, \dots\}$
- $\sim L = \{\epsilon, b, bb, bbb, bbbb, \dots\}$
- All strings in L should reach any - A state

All strings in $\sim L$ should not reach any $\rightarrow A$ state

How to write a Transition Diagram: steps are...

Find the minimum string accepted, this decides the no of states in the FSM, in most of the cases

Then, take longer strings and make them accepted, while modifying the transitions,

Check for minimum strings that are not to be accepted, are really not accepted as per the transition diagram.

See that each state has transitions equal to the no of alphabets present.

Two transition on the same alphabet do not go to different states.

Solution to the problem-1

- $L = \{a, aa, aaa, baa, baaabbb, \dots\}$
- $\sim L = \{\epsilon, b, bb, bbb, bbbb, \dots\}$
- Whenever a string from L is input, it should land in final state.
- Whenever a string from $\sim L$ is input, it should not land in final state, it can be in any other state.

Problem – 2:

Write a DFSA to accept the language

$$L = \{ w \in \{a, b\}^* \mid |w| \text{ is even length} \}$$

Step 1: Write strings accepted by L i.e.

$$L = \{ \epsilon, aa, bb, ab, ba, aaaa, bbbb, bbaa, baba, \dots \}$$

(note : ϵ is even, because its length is 0, which is even)

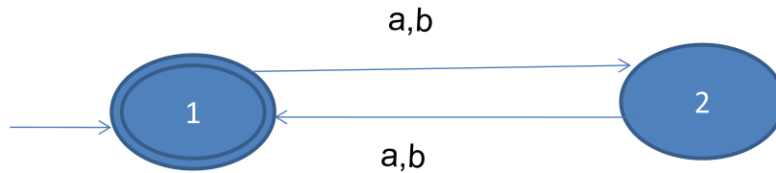
$$\sim L = \{ a, b, aaa, bbb, aba, bab, bba, aab, aabbb, \dots \}$$

Step 2: since min string are $\{\epsilon, aa\}$, 2 states are required.

Step 3: Write Transition Diagram.

Problem - 2

- **Transition Diagram:**



- $L = \{ \epsilon, aa, bb, ab, ba, abab, aabb, bbaa, baba, \dots \}$
- $\sim L = \{ a, b, aaa, bbb, aba, bab, bba, aab, aabbb, \dots \}$

Problem - 2

- **Transition table: δ is transition function**
- **maps $\delta: k \times \Sigma$ to k**

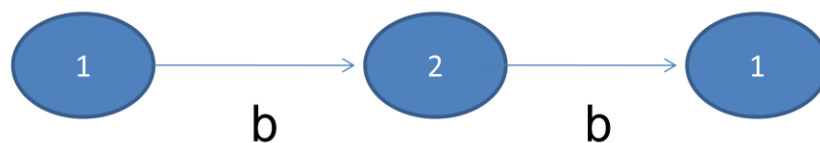
δ	a	b
-> * 1	2	2
2	1	1

Problem - 2

- **To show some strings are accepted: How**

Example : show string bb is accepted.

Draw the states reached by inputting one character at a time, as shown



Since reading all the characters state 1 , is reached , which is final state, Therefore string bb is accepted

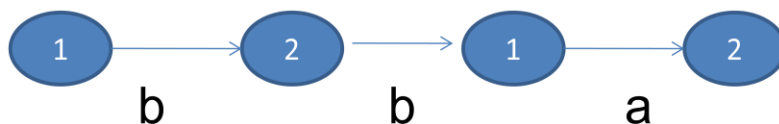
Problem – 2

Problem - 2

- **To show some strings are not accepted: How**

Example : show string bba is accepted.

Draw the states reached by inputting one character at a time, as shown



Since reading all the characters state 2 , is reached , **which is not a final state**, Therefore string bba is not accepted

Problem – 3

Write a DFSM to accept the language

$$L = \{ w \in \{a, b\}^* \mid ab \text{ is a substring of } w \}$$

Step 1: Write strings accepted by L i.e.

$$L = \{ ab, abab, aaab, abaaa, abbbb, bbababab, babb, bbab, baba, \dots \}$$

$$\sim L = \{ a, b, aa, bb, bbb, bba, bba, aaa, bbbbbb, \dots \}$$

Step 2: since min string is { ab}, 3 states are required.

Step 3: Write Transition Diagram.

Problem - 3

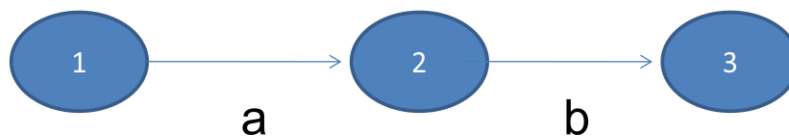


Problem - 3

• To show some strings are accepted: How

Example : show string ab is accepted.

Draw the states reached by inputting one character at a time, as shown



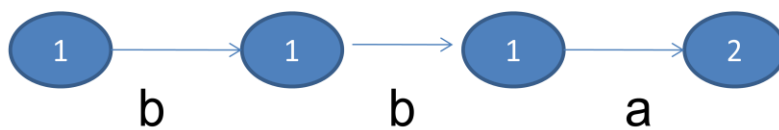
Since reading all the characters state 3, is reached, which is final state, Therefore string ab is accepted

Problem -3

- **To show some strings are not accepted: How**

Example : show string bba is not accepted.

Draw the states reached by inputting one character at a time, as shown



Since reading all the characters state 2 , is reached , which is not a final state, Therefore string bba is not accepted

Problem – 4

Write a DFSM to accept the language

$L = \{ w \in \{a, b\}^* \mid \text{every } w \text{ ends in } b \}$

$L = \{ b, ab, abab, aaab, abaab, abbbb, bbababab, babb, bbab, babb, \dots \}$

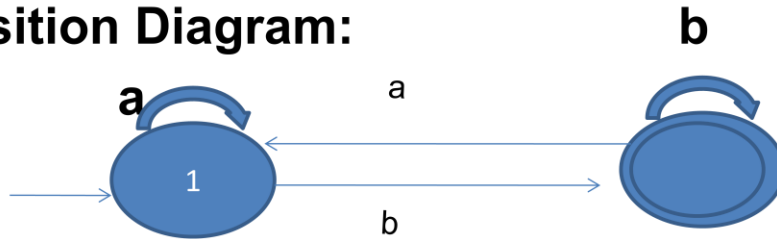
$\sim L = \{ a, aa, ba, bba, baa, baba, aaa, bbbba, \dots \}$

Step 2: since min string are { b}, 2 states are required.

Step 3: Write Transition Diagram.

Problem - 4

- Transition Diagram:



- $L = \{ b, bbb, aaab, ababab, bbbbbbab, \dots \}$
- $\sim L = \{ a, ba, aaa, bbba, aba, baba, bba, aaba, aabbba, \dots \}$

Problem - 4

- Transition table: δ is transition function
- maps $\delta: k \times \Sigma$ to k

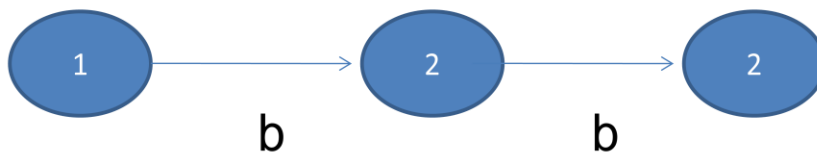
δ	a	b
1	1	2
-> * 2	1	2

Problem - 4

- **To show some strings are accepted: How**

Example : show string bb is accepted.

Draw the states reached by inputting one character at a time, as shown



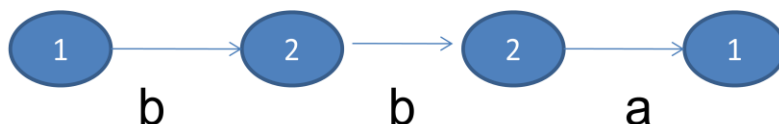
Since reading all the characters state 2 , is reached , which is final state, Therefore string bb is accepted

Problem - 4

- **To show some strings are not accepted: How**

Example : show string bba is accepted.

Draw the states reached by inputting one character at a time, as shown



Since reading all the characters state1, is reached , which is not a final state, Therefore string bba is not accepted

Write a DFSM to accept the language

$$L = \{ w \in \{a, b\}^* \mid \text{every } w \text{ ends in } ab \text{ or } ba \}$$

$$L = \{ ab, ba, abab, aaba, abaab, abbba, bbababab, baba, bbab, baba, \dots \}$$

$$\sim L = \{ a, aa, bb, abb, baa, babb, aaa, bbbbabb, \dots \}$$

Step 2: since min string are $\{ab, ba\}$, we are not able to guess no of states.

Note : this is a difficult problem, we end up in spending lot of time to find the solution

Write a DFSM to accept the language –another difficult problem

$$L = \{ w \in \{a, b\}^* \mid 3^{\text{rd}} \text{ character from right is } a \}$$

$$L = \{ abb, bbbbabb, ababb, aaba, aaaaa, ababa, bbabababb, baba, bbabb, baba, \dots \}$$

$$\sim L = \{ a, aa, bb, abbb, baabbb, babba, bbb, bbbbbb, \dots \}$$

Step 2: since min string are not there, we are not able to guess no of states.

Note : this is a difficult problem, we end up in spending lot of time to find the solution

How to solve difficult problems – study Nondeterministic finite state machines(NFSM)

Nondeterministic Finite State Machines(NFSM) –definition:

Defn: A NFSM , M is a quintuple:

$$\underline{(K, \Sigma, \Delta, s, A)}$$

- K is a finite set of states,
- Σ is the input alphabet,
- $s \in K$ is the start state

- A subset of K is the set of accepting states and
- Δ is the transition function it maps from:

$(K \times (\Sigma \cup \{\epsilon\}))$ to K

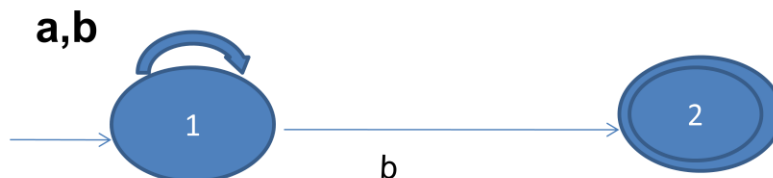
Example of NFSMs

Write a NFSM to accept the language

$L = \{ w \in \{a, b\}^* \mid |w| \text{ ends in } b \}$ //problem 3 NFSM see below

Problem - 3

- **Transition Diagram:**



- $L = \{ b, bbb, aaab, ababab, bbbbbbab, \dots \}$
- $\sim L = \{ a, ba, aaa, bbba, aba, baba, bba, aaba, aabbba, \dots \}$
- **Note: every state can have zero or one or more (equal to the no of alphabets) transitions**

Write a DFSM to accept the language

$L = \{ w \in \{a, b\}^* \mid \text{every } w \text{ ends in } ab \text{ or } ba \}$ //problem 3

$L = \{ ab, ba, abab, aaba, abaab, abbba, bbababab, baba, bbab, baba, \dots \}$

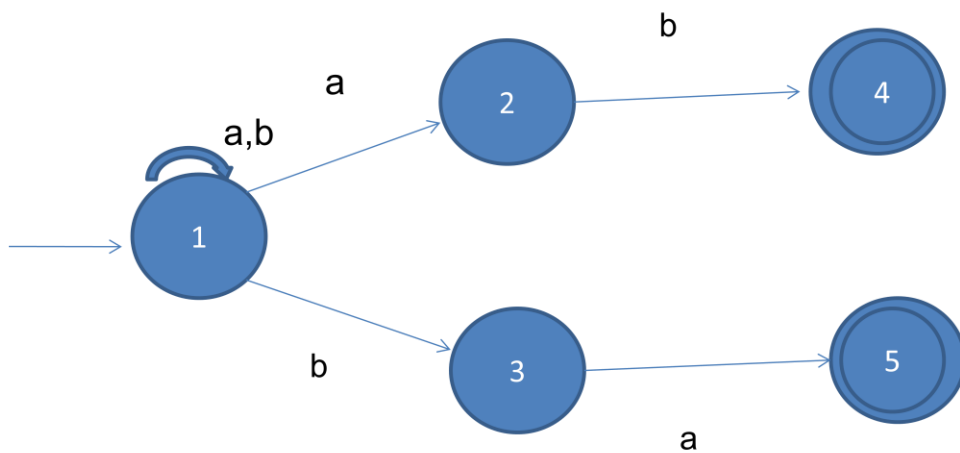
$\sim L = \{ a, aa, bb, abb, baa, babb, aaa, bbbabb, \dots \}$

Step 2: since min string are $\{ab, ba\}$, we are not able to guess no of states.

Note : this is a difficult problem, we end up in spending lot of time to find the solution

Problem – 3 – NFSM

- Solution: regular expression= $(a+b)^*(ab+ba)$



b

How to go about, with difficult problems

Write a DFSM to accept the language

$L = \{ w \in \{a, b\}^* \mid 3^{\text{rd}} \text{ character from right is } a \}$

$L = \{ abb, bbbabb, ababb, aaba, aaaaa, ababa, bbabababb, baba, bbabb, baba, \dots \}$

$\sim L = \{ a, aa, bb, abbb, baabbb, babba, bbb, bbbbbb, \dots \}$

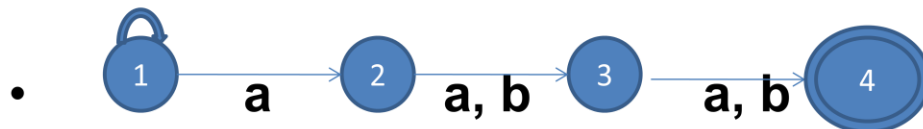
Step 2: since min string are not there, we are not able to guess no of states.

Note : this is a difficult problem, we end up in spending lot of time to find the solution

Problem – 4 – NFSM

- Solution: Regular Expression
- $(a+b)^*a(a+b)(a+b)$

- a,b



Write a NFSM to accept the language

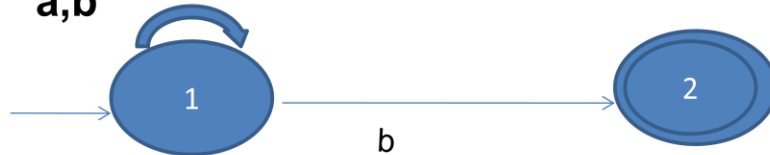
$L = \{ w \in \{a, b\}^* \mid \text{every } w \text{ ends in } b \}$ //solution problem – 1 NFSM

$L = \{ b, ab, abab, aaab, abaab, abbbb, bbababab, babb, bbab, babb,.. \}$

$\sim L = \{ a, aa, ba, bba, baa, baba, aaa, bbbba,.. \}$

Problem – 1 - NFSM

- Transition Diagram: regular exp = $(a+b)^*b$
-



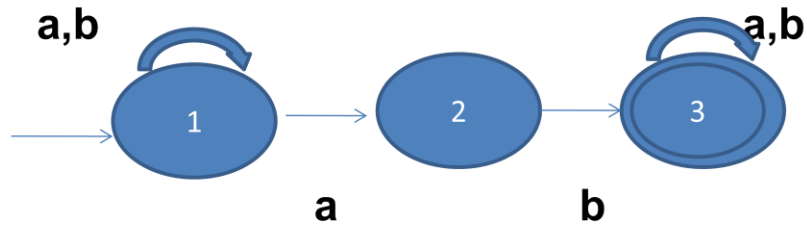
- $L = \{ b, bbb, aaab, ababab, bbbbbbab, \dots \}$
- $\sim L = \{ a, ba, aaa, bbba, aba, baba, bba, aaba, aabbba, \dots \}$
- Note: every state can have zero or one or more (equal to the no of alphabets) transitions

Write NFSM to recognize $L = \{ w \in \{a, b\}^* \mid |w| \text{ contains } ab \}$

- Solution: problem – 2 - NFSM
- regular expression $(a+b)^*ab(a+b)^*$

Problem – 2 - NFSM

- Transition Diagram:



- $L = \{ ab, bab, aab, bbab, aaab, \dots \}$
- $\sim L = \{ a, ba, aaa, bbba, bbbb, baaa, aaaaa, \dots \}$

Write a NFSM to recognize the language

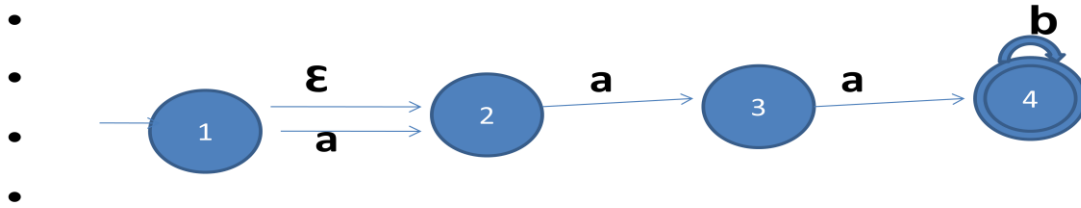
$L = \{w \in \{a, b\}^* \mid w \text{ is made up of an optional } a \text{ followed by } aa, \text{ zero or more } b\text{'s}\}$

- $L = \{ aa, aaa, aab, aaab, aabbb, aaabbb, \dots \}$
- $\sim L = \{ a, ba, baa, bbbbb, bbbbbba, \dots \}$
- Regular expression = re = $(a + \epsilon)aa(b)^*$

Solution:

Problem – 5 – NFSM

- Solution- regular expression:
- $re = (a + \epsilon)aa(b)^*$



Procedure to convert NFSM to DFSM

- Example of NFSMs
- Write a NFSM to accept the language

$$L = \{ w \in \{a, b\}^* \mid |w| \text{ ends in } b \}$$

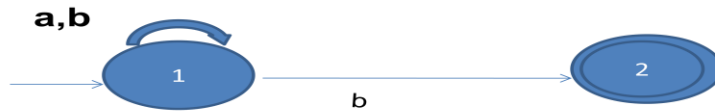
We know all the parameter related to NFSM

- K is a finite set of states
- Σ is the input alphabet
- $s \in K$ is the start state
- A subset of K is the set of accepting states and

δ is the transition function. Consider the following problem

Problem – 1 - NFSM

- **Transition Diagram: regular exp = $(a+b)^*b$**



- K is a finite set of states = $\{1,2\}$
- Σ is the input alphabet = $\{a,b\}$
- $s \in K$ is the start state = 1
- A subset of K is the set of accepting states = $\{2\}$
- δ is the transition function as indicated in above fig

Procedure of conversion

We need to calculate the following parameter of DFSM, note only three parameters, i.e. K' , A' , δ' need to be calculated

- K' is a finite set of states = ?
- Σ is the input alphabet = no change
- $s \in K$ is the start state = no change
- A' subset of K is the set of accepting states = ?
- δ' is the transition function = ?
- $s' = s = \{1\}$ // note the set notation
- Compute δ'
- Active states = $\{\{1\}\}$, consider $\{1\}$

$\delta'(\{1\}, a) = \{1\}$.. This exists

$\delta'(\{1\},b) = \{1,2\}$.. This does not exist, add

New Active states = $\{\{1\},\{1,2\}\}$, consider $\{1,2\}$

$\delta'(\{1,2\},a) = \delta(\{1\},a) \cup \delta(\{2\},a) = \{1\} \cup \Phi = \{1\}$ exists

$\delta'(\{1,2\},b) = \delta(\{1\},b) \cup \delta(\{2\},b) = \{1,2\} \cup \Phi = \{1,2\}$ exists,

No new states are added, therefore Procedure terminates

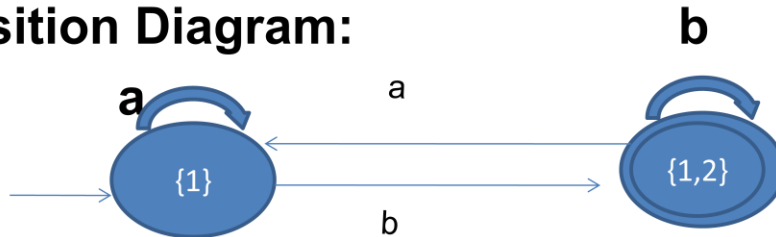
- Let us write the results in tabular form of δ' , i.e. transition function.

	δ'	a	b
->	{1}	{1}	{1,2}
*	{1,2}	{1}	{1,2}

Solution is as follows:

Problem – 1 - DFSM

- Transition Diagram:



- This is the same solution we got when we constructed the DFSM, except the state are in set notation, which we can rename them as $\{1\} = 3$, and $\{1,2\} = 4$. Now it is identical to the original solution.

Consider the following problem for which we know the NFSM. And apply the procedure to convert it to FSM

Procedure:

$s' = s = \{1\}$ // note the set notation

Compute δ'

Active states = $\{\{1\}\}$, consider $\{1\}$

$\delta'(\{1\}, a) = \{1,2\}$.. add

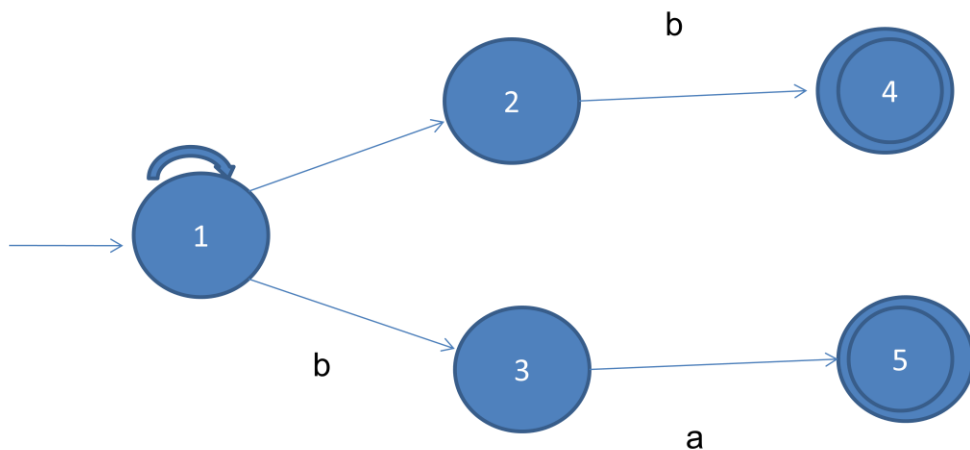
$\delta'(\{1\}, b) = \{1,3\}$.. add

Active states = $\{\{1\}, \{1,2\}, \{1,3\}\}$, consider $\{1,2\}$

$\delta'(\{1,2\}, a) = \{1,2\}, a \cup \Phi = \{1,2\}$..exists

$\delta'(\{1,2\},b) = \{1,3\} \cup \{4\} = \{1,3,4\}$ add

Problem – 3 – NFSM



b

Active states = $\{\{1\}, \{1,2\}, \{1,3\}, \{1,3,4\}\}$, consider $\{1,3\}$

$\delta'(\{1,3\},a) = \{1,2\} \cup \{5\} = \{1,2,5\}$..add

$\delta'(\{1,3\},b) = \{1,3\} \cup \{\} = \{1,3\}$ exists

Active states = $\{\{1\}, \{1,2\}, \{1,3\}, \{1,3,4\}, \{1,2,5\}\}$, consider $\{1,3,4\}$

$\delta'(\{1,3,4\},a) = \{1,2\} \cup \{5\} \cup \{\} = \{1,2,5\}$..exists

$\delta'(\{1,3,4\},b) = \{1,3\} \cup \{\} \cup \{\} = \{1,3\}$ exists

Active states = $\{\{1\}, \{1,2\}, \{1,3\}, \{1,3,4\}, \{1,2,5\}\}$, consider $\{1,2,5\}$

$\delta'(\{1,2,5\}, a) = \{1,2\} \cup \{\} \cup \{\} = \{1,2\}$..exists

$\delta'(\{1,2,5\}, b) = \{1,3\} \cup \{4\} \cup \{\} = \{1,3,4\}$ exists

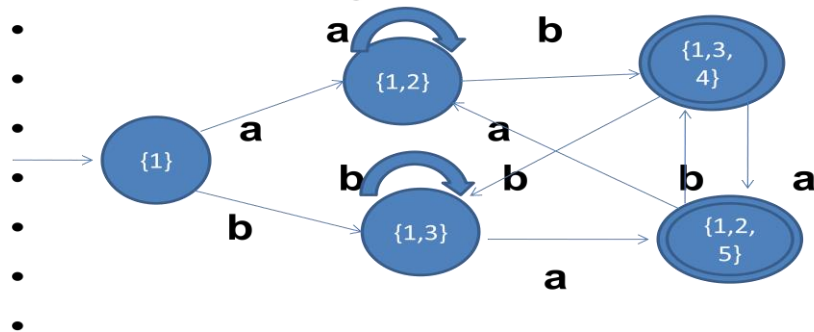
no new states are added, therefore Procedure terminates

write transition table:

	a	b
{1}	{1,2}	{1,3}
{1,2}	{1,2}	{1,3,4}
{1,3}	{1,2,5}	{1,3}
{1,3,4}	{1,2,5}	{1,3}
{1,2,5}	{1,2}	{1,3,4}

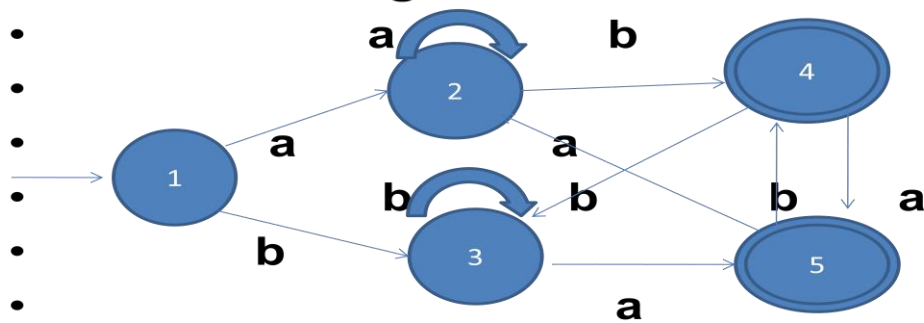
Write transition diagram.

- Transition diagram:



Rename the states and check for some representative strings in the fig given below.

- Transition diagram: DFSM



- Check for some strings for the correctness

Lecture – 5 : chapter 5

Procedure to convert NFSM to DFSM:

The problem which we are attempting to convert has ϵ transition.

We need to calculate eps for each state using the algorithm as follows:

Eps(q: state) // algorithm

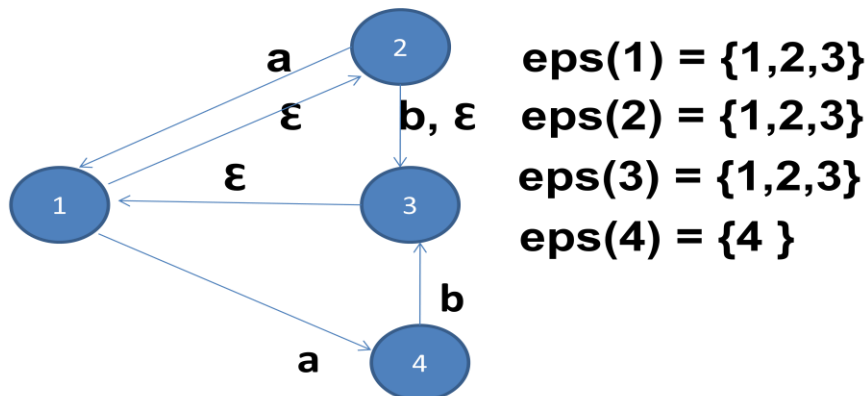
1. result = {q} and some
2. while there exists $p \in \text{result}$ $r \notin \text{result}$ and some transition $(p, \epsilon, r) \in \text{transition function}$ do:
 insert r into result.
3. return result.

Note: It means connect all states that can be reached on ϵ

Example-1 for calculation of eps:

Example-1 for calculation of eps

- Consider a diagram of NFSM



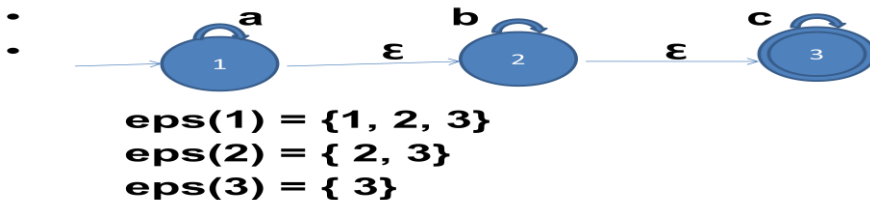
$\text{eps}(1) = \{1,2,3\}$
 $\text{eps}(2) = \{1,2,3\}$
 $\text{eps}(3) = \{1,2,3\}$
 $\text{eps}(4) = \{4\}$

Note: $\text{eps}(\text{any state}) = \{\text{that state is always included}\}$

a b

Example-2 for calculation of eps:

Example-2 for calculation of eps



$\text{eps}(1) = \{1, 2, 3\}$
 $\text{eps}(2) = \{2, 3\}$
 $\text{eps}(3) = \{3\}$

Problem – 3 – NFSM to DFSM:

Write a NFSM to recognize the language

$L = \{w \in \{a, b\}^* \mid w \text{ is made up of an optional } a \text{ followed by } aa \text{ zero or more } b\text{'s}\}$

$L = \{aa, aaa, aab, aaab, aabbb, aaabbb, \dots\}$

$\tilde{L} = \{ a, ba, baa, bbbb, bbbba, \dots \}$

Regular expression = $re = (a + \epsilon)aa(b)^*$

Write the transition diagram of NFSM

Problem – 3 – NFSM

- **Solution- regular expression:**
- $re = (a + \epsilon)aa(b)^*$

•

•

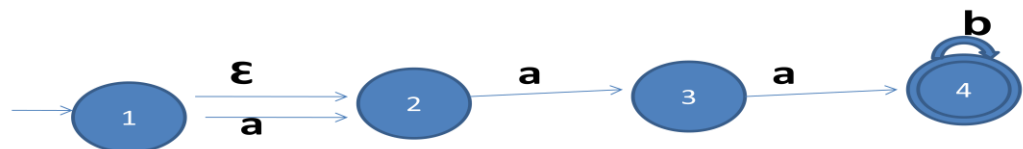
•

•

•

•

•



• $eps(1) = \{1, 2\}$

• $eps(2) = \{2\}$

• $eps(3) = \{3\}$ $eps(4) = \{4\}$

Procedure to convert NFSM to DFSM

We know all the parameter related to NFSM

- K is a finite set of states
- Σ is the input alphabet
- $s \in K$ is the start state
- A subset of K is the set of accepting states and
- δ is the transition function

Procedure of conversion:

We need to calculate the following parameter of DFSM, note only three parameters, i.e. K' , A' , δ' need to be calculated

- K' is a finite set of states = ?
- Σ is the input alphabet = no change
- $s \in K$ is the start state = no change
- A' subset of K is the set of accepting states = ?

δ' is the transition function = ?

We need to calculate the following parameter of DFSA, note only three parameters, i.e. K' , A' , δ' need to be calculated

- K' is a finite set of states = ?
- Σ is the input alphabet = no change
- $s \in K$ is the start state = no change
- A' subset of K is the set of accepting states = ?
- δ' is the transition function = ?

Procedure:

$s' = s = \text{eps}\{1\} = \{1,2\}$ // this is start state DFSA

Compute δ'

Active states = $\{\{1,2\}\}$, consider $\{1,2\}$

$\delta'(\{1,2\}, a) = \text{eps}\{\delta(1,a) \cup \delta(2,a)\} = \text{eps}(2) \cup \text{eps}(2)$

= $\{2,3\}$ This state does not exist, therefore add

$\delta'(\{1,2\}, b) = \text{eps}\{\delta(1,b) \cup \delta(2,b)\} = \text{eps}(\Phi) \cup \text{eps}(\Phi)$

= Φ , This state does not exist, therefore add

Now Active states = $\{\{1,2\},\underline{\{2,3\}},\Phi\}$, consider $\{2,3\}$

$\delta'(\{2,3\},a) = \text{eps}\{\delta(2,a) \cup \delta(3,a)\} = \text{eps}(3) \cup \text{eps}(4) = \{3,4\}$ this state does not exist, add

$\delta'(\{2,3\},b) = \text{eps}\{\delta(2,b) \cup \delta(3,b)\} = \text{eps}(\Phi) \cup \text{eps}(\Phi) = \Phi$ already exists, do not add

Now Active states = $\{\{1,2\},\{2,3\},\Phi,\underline{\{3,4\}}\}$, consider $\{3,4\}$

$\delta'(\{3,4\},a) = \text{eps}\{\delta(3,a) \cup \delta(4,a)\} = \text{eps}(4) \cup \text{eps}(\Phi) = \{4\}$ this state does not exist, add

$\delta'(\{3,4\},b) = \text{eps}\{\delta(3,b) \cup \delta(4,b)\} = \text{eps}(\Phi) \cup \text{eps}(4) = \{4\}$ does not exist, add

Now Active states = $\{\{1,2\},\{2,3\},\Phi,\{3,4\},\underline{\{4\}}\}$, consider $\{4\}$

$\delta'(\{4\},a) = \text{eps}\{\delta(4,a)\} = \text{eps}(\Phi) = \Phi$ this state exists, no need to add

$\delta'(\{4\},b) = \text{eps}\{\delta(4,b)\} = \text{eps}(4) = \{4\}$ this state exists, no need to add

Note: No new states are added, therefore algorithm terminates. Now we have all the states of DFSA and its transition functions

- Let us write the results in tabular form of δ' , i.e. transition function

δ'	a	b
$\rightarrow\{1,2\}$	$\{2,3\}$	Φ
$\{2,3\}$	$\{3,4\}$	Φ
Φ	Φ	Φ
$\ast\{3,4\}$	$\{4\}$	$\{4\}$
$\ast\{4\}$	Φ	$\{4\}$

DFSM is constructed as follows:

First find out the accepting states of NFSM In this case it is $\{4\}$

Look at all final active states of DFSM. In this case it is:

Active states = $\{\{1,2\},\{2,3\},\Phi,\{3,4\},\{4\}\}$

Find all the states containing state $\{4\}$

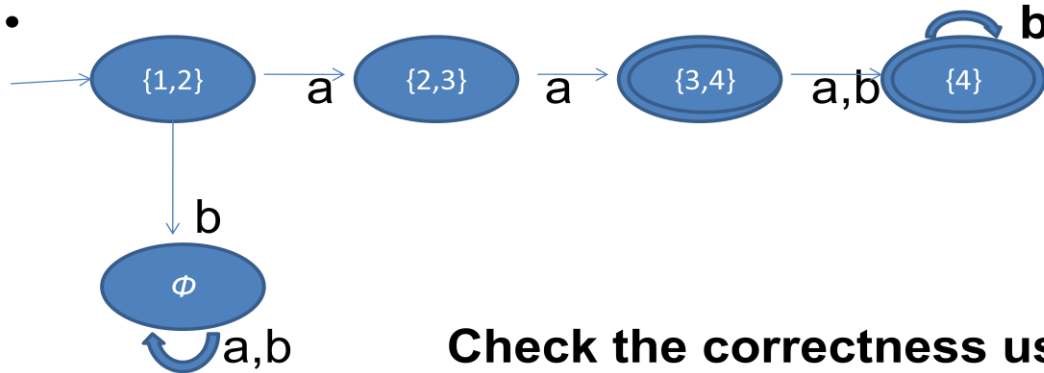
There are two state namely $\{3,4\}$ and $\{4\}$

They will accepting states of DFSM

Transitin diagram- DFSM:

Transitin diagram- DFMSM

- Write transition diagram using K' , A' and δ'



- $L = \{ aa, aaa, aab, aaab, aabbb, aaabbb, \dots \}$
- $\sim L = \{ a, ba, baa, bbbbb, bbbbbbba, \dots \}$

FSM to operational systems:

Now that we know how to design a DFSM if it is simple and if it is complex, we write NFSM and convert the same to DFSM, using the procedure discussed.

FSM can be simulated using Software or Hardware depending on the requirement.

In the next section we will discuss simulating using a pseudo code.

Simulation the deterministic FSM:

Simulation the deterministic FSM

- Transition diagram: a



- Above transition diagram for DFSM to accept the language :
 $L = \{ w \in \{a, b\}^* \mid ab \text{ is a substring of } w \}$

Hardcoding a Deterministic FSM:

Until accept or reject do:

1: s=get-next-symbol.

 If s=b go to 1.

 else if s=a then go to 2

2: s=get-next-symbol

 If s=a go to 2.

 else if s=b then go to 3

3: If s=a or b go to 3.

 else if s= end-of-file then accept.

 else reject.

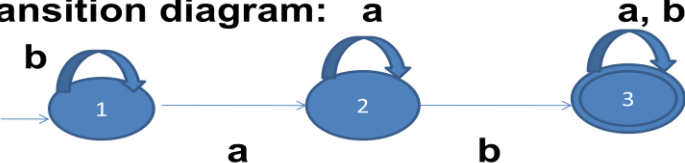
Simple interpreter for deterministicFSM:

dfsmsimulate(M:DFSM, w: string)

- 1 $st = s$
- 2 Repeat
 - 2.1 $c = \text{get-next-symbol}(w)$
 - 2.2 if $c \neq \text{end-of-file}$ then:
 - 2.2.1 $st = \delta(st, c)$
 - until $c = \text{end-of-file}$
- 3 If $st \in A$ then accept
 - else reject.

Trace dfsmsimulate:

Trace dfsmsimulate

- Transition diagram:
 

+

Note: Each state has 2 transitions.
Final $st = 3$, therefore string bbaabb is accepted (see trace in next slide)

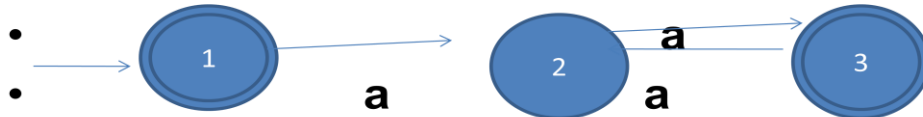
Trace table for string bbaabb

st	c	$\delta(st,c)$
1	b	1
1	b	1
1	a	2
2	a	2
2	b	3
3	b	3

What is minimization of FSMs:

What is minimization of FSMs

- Write a DFSM to accept the language
 $L = \{ w \in \{a, b\}^* \mid |w| \text{ is even length} \}$ Example

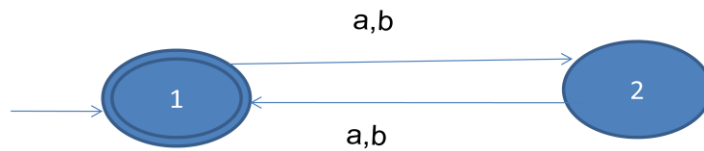


$L = \{ \epsilon, aa, bb, ab, ba, abab, aabb, bbaa, baba, \dots \}$

- $\sim L = \{ a, b, aaa, bbb, aba, bab, bba, aab, aabbb, \dots \}$
-

What is minimization of FSMs

- Transition Diagram:



- $L = \{ \epsilon, aa, bb, ab, ba, abab, aabb, bbaa, baba, \dots \}$
- $\sim L = \{ a, b, aaa, bbb, aba, bab, bba, aab, aabbb, \dots \}$

Note : The behaviour of state 1 and 3 are identical as shown below:

- $\delta(1,a) = 2$
- $\delta(3,a) = 2$

Therefore there is no need to have two separate states 1 and 3 and they can be combined as shown in the above diagram. Also note that two states are a must and it can not be further minimized.

Lecture – 6:

Equivalence class: Definition

An equivalence relation has following properties.

- It is reflexive
- It is symmetric

- It is transitive.

Example:

-- relation- has the same birth date

-- relation- defined by =

Not an example: relation \leq

Equivalence relations can also be represented by a digraph since they are a binary relation on a set. For example the digraph of the equivalence relation congruent mod 3 on $\{0, 1, 2, 3, 4, 5, 6\}$ is as shown below. It consists of three connected components

Equivalence class –Example:

- $0 \bmod 3 = 0$ The results are $\{0,1,2\}$
- $1 \bmod 3 = 1$
- $2 \bmod 3 = 2$
- $3 \bmod 3 = 0$
- $4 \bmod 3 = 1$
- $5 \bmod 3 = 2$
- $6 \bmod 3 = 0$

$\{0, 3, 6\}$ --- have 0 as their remainder

$\{1, 4\}$ --- have 1 as their remainder

$\{2, 5\}$ --- have 2 as their remainder

Defn: Indistinguishable:

We say that x and y are indistinguishable with respect to L , which we will write as

$x \approx_L y$ iff:

all $z \in \Sigma^*$ (either xz and $yz \in L$ or neither is)

consider $x=aa$ and $y = bb$ and $z=ba$

since $aaba$ and $bbba$ are in L , therefore

$x \approx_L y$

Defn: Distinguishable:

We say that x and y are distinguishable with respect to L , iff they are not indistinguishable. If x and y are distinguishable then there exists at least one string z , such that one but not both of xz and yz is in L

consider $x=aa$ and $y = a$ and $z=ba$

since $aaba$ and aba both are in not L ,

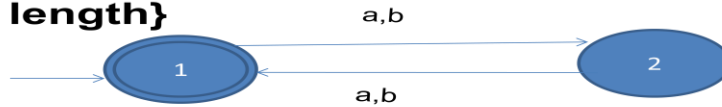
$aaba$ is in L and aba is not in L

- Note : indistinguishable is a equivalence class

Equivalence class-two partitions:

Equivalence class-two partitions.

- **Transition Diagram for $L = \{ w \in \{a, b\}^* \mid |w| \text{ is even length} \}$**



- $L = \{ \epsilon, aa, bb, ab, ba, abab, aabb, bbaa, baba, \dots \}$
- $\sim L = \{ a, b, aaa, bbb, aba, bab, bba, aab, aabbb, \dots \}$
- **Any string from L is distinguishable from any string from $\sim L$**

$L = \{ \epsilon, aa, bb, ab, ba, aaaa, bbbb, bbaa, baba, \dots \}$

All the elements are indistinguishable

$\sim L = \{ a, b, aaa, bbb, aba, bab, bba, aab, aabbb, \dots \}$

All the elements are indistinguishable.

For example aa and a are distinguishable, because

Take an element bb from the language L , $aabb$ is in L , abb

is not in L , therefore they are not in the same eq.class

For example aa and bb are indistinguishable, because

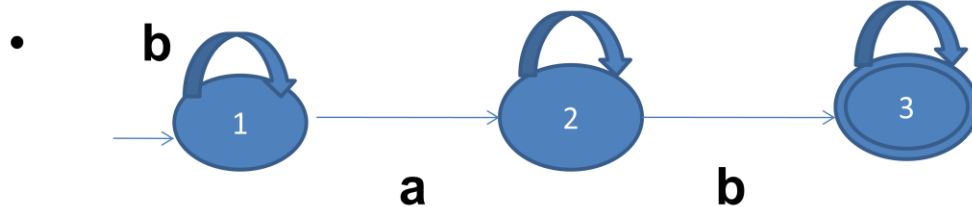
Take an element bb from the language L , $aabb$ is in L , $bbbb$

is also in L , therefore they are in the same eq.class

Equivalence class-three partitions.

Equivalence class-three partitions.

- Transition diagram: a



**L = set of all string containing string – ab
where $\Sigma = \{a, b\}$**

L = { ab, abab, aaab, abaaa, abbbb, bbababab, babb, bbab, baba,...}

$\sim L = \{ a, b, aa, bb, bbb, bba, bba, aaa, bbbbbb,...\}$

All the strings of L belong to state 3.

How to separate $\sim L$ into two separate states.

What is the basis?

- [1] = { ϵ , b, bb, bbb, bbbb,}
- [2] = {a, aa, aaa, aaaa,}

For example consider b from block-1 and a from block-2,
take b from Σ^*

bb is not in L whereas ab is in L, therefore they
are not indistinguishable, i.e. Distinguishable

- Equivalence class partitions:

In general equivalence partitions of L and $\sim L$,

Can further partitions.

In the last example we saw partition of $\sim L$

in to 2 eq classes.

We will see now how L can be partitioned into more than one block or eq. Classes.

Equivalence class-more partitions

Find the equivalence partition of L , where

$L = \{w \in \{a, b\}^* \mid w \text{ has no adjacent characters are the same}\}$
// problem 5.26(p-89)

$L = \{\epsilon, a, aba, ababa, b, ab, bab, abab, \dots\}$

$\sim L = \{aa, abaa, ababb, bbbbbb, aaaa, \dots\}$

Check any of L or $\sim L$ can be further separated,

Consider $L = \{\epsilon, a, aba, ababa, b, ab, bab, abab, \dots\}$ and select a and b

Also select a from Σ^*

Test: aa is in $\sim L$ and ba is L . Therefore L is not an eq class, it should be further refined.

- $[1] = \{a, aba, ababa, \dots \text{etc will get separated}\}$
- $[2] = \{b, ab, bab, abab, \dots\}$

- Consider $L = \{\epsilon, a, aba, ababa, b, ab, bab, abab, \dots\}$ and select ϵ and a
- Also select a from Σ^*
- $\epsilon a = a$ is in L , but
- aa is not in L , therefore ϵ and a are not in the same eq. class, finally,
- $[1] = \{\epsilon\}$
- $[2] = \{a, aba, ababa, \dots\}$
- $[3] = \{b, ab, bab, abab, \dots\}$
- $[4] = \{aa, abaa, ababb, \dots\}$

The transition diagram for above partitions is

Equivalence class-solution.

Solution to the problem : (workout on the board.)

Finally how to know that DFSA is not possible. i.e. When the partitions are infinite, no DFSA is possible.

- Example: $L = \{a^n b^n \mid n > 0\}$

This will have infinite partitions, no DFSA is possible, and L is not regular language.

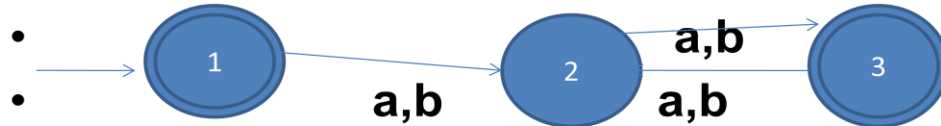
Theorem 5.6(Myhill-Nerode):

Theorem: A language is regular iff the number of eq. Classes of L is finite.

Minimization of FSMs-problem -1:

Minimization of FSMs-problem -1

- Write a DFMSM to accept the language
 $L = \{ w \in \{a, b\}^* \mid |w| \text{ is even length} \}$ Example



$$L = \{ \epsilon, aa, bb, ab, ba, abab, aabb, bbaa, baba, \dots \}$$

- $\sim L = \{ a, b, aaa, bbb, aba, bab, bba, aab, aabbb, \dots \}$

Procedure for Minimization of DFMSM:

Partition the states in to non-accepting and accepting states.

{2} and {1,3}

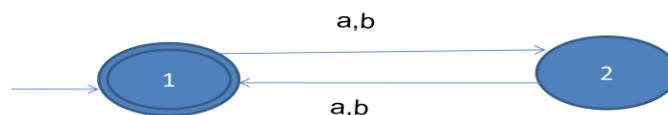
Check whether they are distinguishable?

Workout(on the board)

Minimized FSM-problem-1

Minimized FSM-problem-1

- Transition Diagram:

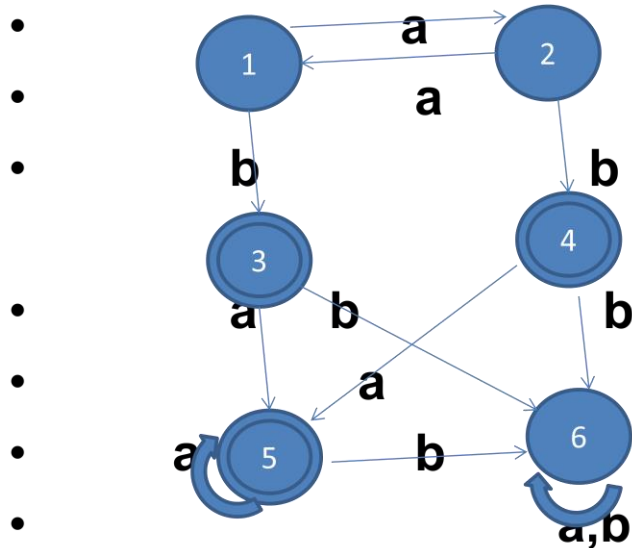


- $L = \{ \epsilon, aa, bb, ab, ba, abab, aabb, bbaa, baba, \dots \}$
- $\sim L = \{ a, b, aaa, bbb, aba, bab, bba, aab, aabbb, \dots \}$

Minimization of FSMs -Problem-2:

Minimization of FSMs -Problem-2

- Minimize the DFMSM



Start with what is clearly distinguishable i.e.

Non-Accepting states and accepting states

- $\{1, 2, 6\}$ and $\{3, 4, 5\}$ check further if they are
- Distinguishable input

Continue subdividing state, until all distinguishable states are separated.

Lecture:7

Moore Machine(transducer)

Transducer: A device that converts variations in a physical quantity, such as pressure or brightness, into an electrical signal, or vice versa.

Defn: A Moore machine, M is a seven tuple:

$(K, \Sigma, O, \delta, D, s, A)$, where

- K is a finite set of states,
- Σ is the input alphabet,
- O is the output alphabet,
- $s \in K$ is the start state
- A subset of K is the set of accepting states(not imp)
- δ is the transition function it maps from $K \times \Sigma$ to K
- D is the display or output function from K to $(O)^*$

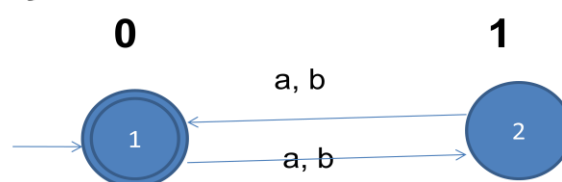
Example of Moore Machine:

Example of Moore Machine

- In Moore machine each state is associated with a output. Suppose we want Moore machine to output '0' when the length of input string is even, otherwise output '1'

- $\Sigma = \{a, b\}$

•



Transition table for the Moore Machine(see fig a bove):

Transition table

Transition function	Input = a	Input = b	output
1	2	2	0
2	1	1	1

Mealy machine(transducer):

Defn: A Mealy machine, M is a six tuple:

$(K, \Sigma, O, \delta, s, A)$, where

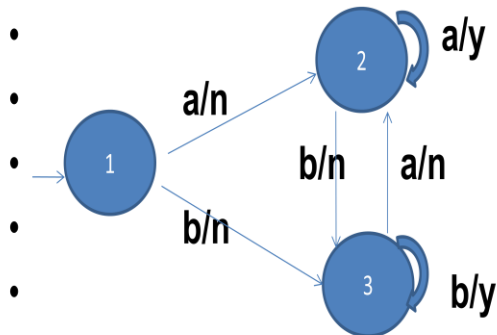
- K is a finite set of states,
- Σ is the input alphabet,
- O is the output alphabet,
- $s \in K$ is the start state
- A subset of K is the set of accepting states and
- δ is the transition function it maps from $(K \times \Sigma)$ to $(K \times O^*)$

Note: output is associated with each input.

Example of Mealy machine:

Example of Mealy machine

- $L = \{ w \in \{a, b\}^* \mid w \text{ ends in } aa \text{ or } bb \}$
- Requirements are **when input ends aa it should have output 'y'** and **when input ends bb it should have output 'n'**



Transition table:

Transition table

Trans fun	a	output	b	output
1	2	n	3	n
2	2	y	3	n
3	2	n	3	y

Computation(chapter 4):

In this chapter, effort is made to:

Define problems as languages to be decided

Define programs as state machines whose input is a string and whose output is *accept or reject*

- Key ideas :
1. Decision procedures
 2. Non determinism
 3. Function on languages.

Decision Procedures

Defn: A decision problem is one for which we must make a *yes/no* decision.

A decision procedure is an algorithm to solve a decision problem.

It a program whose result is a Boolean value.

In order to return a Boolean value, a decision procedure must be guarantee to halt on all inputs

Decision procedures are to answer question such as:

- Is string s in Language L ?
- Given a machine, does it accept any string?
- Given two machines, do they accept the same strings?
- Given a machine, is it the smallest m/c that does its job?

Three imp things about Procedures:

1. Does there exist a decision procedure(algorithm)

2.If any decision procedures exist, find one

3. If exists, find the most efficient one, and how efficient it is?

Decision procedures are programs, and they must have two correctness properties:

1. Program must be guaranteed to halt.

2. The answer must be correct.

Example – 1:

Checking for even numbers:

even(x: integer)=

 If $(x/2)*2=x$ then return *True*

 else return *False*.

If $x=3$ then $x/2=1$ and $1*2 \neq 3$ therefore “false”

If $x=8$ then $x/2=4$ and $4*2 = 8$ therefore “true”

Example – 2:

Checking for Prime numbers:

prime(x: positive integer) =

 For $i = 2$ to $\text{ceiling}(\text{sqrt}(x))$ do:

 If $(x/i)*i = x$ then return *False*

 return *True*

Assume $x = 7$ then $\text{ceiling}(\text{sqrt}(x)) = \text{ceiling}(2.65) = 3$

$i = 2; 7/2 * 2 \neq 7$ next iteration

$i = 3; 7/3 * 2 \neq 7$ next iteration (no more iterations)

Returns True

Example-3:

Checking for Programs that halt on a particular input.

$\text{haltOnw}(p: \text{program}, w: \text{string}) =$

1. Simulate the execution of p on w
2. If the simulation halts return True
else return False.

note: 1. this is not a procedure, because it never returns False.

2. No decision procedure exists for this.

Determinism and non determinism:

Consider a program:

Choose(action 1;;

 action 2;;

 action n;;)

Observation on choose:

Returns some successful value, if there is one

If there is no successful value, the choose will:

- Halt and return False if all the actions halt and return False
- Fail to halt if any of the actions fails to halt.

(note that this has a potential to return successful value, it may be taking more time)

Deterministic and non Deterministic:

If a program does not use choose then it is deterministic.

If a program includes choose then it is non deterministic.

Functions on Languages and Programs:

The function chop:

chop(L): is all the odd length strings in L with their middle character chopped out.

The function firstchars:

firstchars(L): determines the first characters by looking at all strings in L

Examples of chop(L):

Examples of chop(L)

n	in $A^n B^n C^n$	in chop $A^n B^n C^n$
0	ϵ	
1	abc	ac
2	aabbcc	
3	aaabbbccc	aaabbbccc
4	aaaabbbbcccc	
5	aaaaabbbbbccccc	aaaaabbbbbccccc

Examples of firstchars(L)

Examples of firstchars(L)

L	Firstchars(L)
\emptyset	\emptyset
$\{\epsilon\}$	\emptyset
$\{a\}^*$	$\{a\}^*$
$A^n B^n$	$\{a\}^*$
$\{a, b\}^*$	$\{a\}^* \cup \{b\}^*$

Closure of Languages:

Closure of Languages

	finite	infinite
union	yes(1)	yes(5)
intersection	yes(2)	no(6)
chop()	yes(3)	no(7)
firstchars()	no(4)	yes(8)

Language Hierarchy:

Hierarchy of Languages and corresponding automata

Regular languages: FSMs

Context-free languages: PDAs

D(decidable) Languages: Turing machine

SD(semi decidable) languages: Turing machine

Importance of classification:

The factors are:

1.Computational efficiency: As function of input length

FSMs - Linear with respect to input string

PDAs - cube of the length of input string

TM - exponentially with respect to input

String

2.Decidability: Answer to the questions

FSM - accepts some string?

FSM - is it minimal?

FSMs- are two FSMs identical?

**PDA- only some of the above can be
answered**

TM - none of the above can be answered

3.Clarity: tools that enable analysis- exist?

FSM - yes

PDA - yes

TM - none

..... End of Module – 1

