FORMAL LANGUAGES AND AUTOMATA 22AIE302

Ayswarya R Kurup

10 July 2024

About the Course

References

1. Formal Language and Automata', Peter Linz, Fifth edition, 2012.
2. Introduction to Automata Theory, Languages and Computation', J.E.Hopcroft, R.MotwaniandandJ.D.Ullman, Pearson, 2001.
3. Elements of the Theory of Computation', H.R.Lewis and C.H.Papadimitriou, Prentice Hall, 1997/Pearson 1998.

Course Outcomes

1. Analyse formalisms and write formal proofs for properties
2. Use grammatical notations to represent sequence manipulation problems
3. Apply various formal grammars to the problem-solving avenues
4. Identify limitations of some computational models and possible methods of proving them

Evaluation Pattern

Assessment	Weightage(%)
Assignments	30
Quiz with equal credits	20
Midterm Examination	20
End Semester Examination	30

Pre-requisites

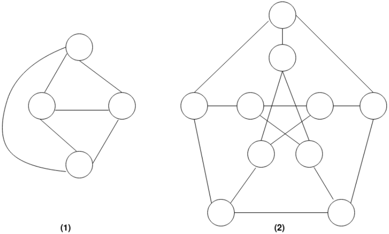
- Data Structures
- Discrete Mathematics

Syllabus I

1. Introduction to Automata and formal language: Finite State machines – Deterministic finite state machine, Non-Deterministic finite state machine, Equivalence of NFA and DFA, Minimization of Finite State Machine, Regular Expression, Regular Language, Properties of Regular Languages.
2. Context Free Grammar: Pushdown Automata, Variants of Pushdown automata, Derivations Using a Grammar, Leftmost and Rightmost Derivations, the Language of a Grammar, Sentential Forms, Parse Tress Equivalence between PDA and CFG, Context Free Language, Properties of CFL, Normal Forms.
3. Context Sensitive Language: Linear Bound Automata, Turing Machine, Variants of Turing Machine, Decidability, Post correspondence problem, Introduction to undecidable problems.

Why Formal Languages and Automata?

1. **3**-coloring problem (Graph coloring):
   Determine whether it is possible to color using three colors $\{1, 2, 3\}$provided no adjacent nodes have same color



2. A computer with input X: do the program halt on X ?
3. Shortest path/ route problem:
   " I want to drive from Ettimadai to Coimbatore", which is the shortest route?

- Every computational problem can be represented as a string input to a computer / device
- String will be over an alphabet denoted by Σ
- Problem can be viewed as a decision problem

Alphabets

- Set of symbols: a finite set, Σ
- Examples:
  - Binary: $\Sigma = \{0, 1\}$
  - Decimal: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - All lower case letters: $\Sigma = \{a, b, c, ..., x, y, z\}$
  - Alphanumeric: $\Sigma = \{a - z, A - Z, 0 - 9\}$
  - DNA molecule letters: $\Sigma = \{a, c, g, t\}$
- $N = \{0, 1, 2, 3, ...\}$ cannot be an alphabet since it is infinite

Strings

- Finite collection of symbols from chosen from alphabet set Σ
- A string of length n over an alphabet Σ is an ordered n-tuple of elements of Σ
- $\Sigma = \{a, b\}$ then $\{\epsilon, ba, bab, aab\}$ are examples of strings over Σ
- If $\Sigma = \{a\}$ then $\Sigma^* = \{\epsilon, a, aa, aaa, ...\}$
- Length of a string w: $|w|$ - number of symbols
- Example: $w = 010100$ $\quad |w| = 6$
- If $|w| = 0$, ϵ, empty string (epsilon) or λ (lambda)
- $\epsilon w = w\epsilon = w$

String Operations

1. Concatenation $(x = ab, y = bc, xy = abbc)$
2. Reversal $(x = ab, x^R = ba)$
3. ϵ denotes empty string $|\epsilon| = 0$
4. $x^K = xxx....$: x is repeated or concatenated k times
   $x^0 = \epsilon$
5. **Kleene star** of x, denoted by x^*
   $x^* =$ set of all x^k $= \{x^k | k \geq 0\}$
6. Σ^*: Set of all strings over Σ of finite length
7. Substring: V is a substring of W, if there exist strings X and Y
   $W = XVY$
   Example: "get" is a substring of "together"
8. A language over Σ is a set of strings over Σ
   $A \subseteq \Sigma^*$

String Operations..

" A language over Σ is a set of strings over Σ"

Examples: set of all binary strings with an odd number of 1's is a language over $\{0, 1\}$

Set of all dictionary words is a language over the English alphabet

Note: Σ^* is also a language

ϵ is also a language

Describing languages

1. Brute Force Listing: $\{a, ab, abb, ..\}$
2. Language Operations: ab^*
3. Other set theoretic operations

Refresher of Discrete Mathematics

Set Theory

- $A = \{a, b, c\}$
  - A is the set whose elements a, b, and c
- $W = \{x : x \in N\}$
  - **W** equals the set of all **x** such that **x** is a natural number
- Empty set: $\emptyset = \{\}$

Set Operations

- Union: $A \cup B$
- Intersection: $A \cap B$
- Elements that are in A but not in B: $A - B$ (Complement of B relative to A)
- Complement of A: A^c
- A is a subset of B: $A \subset B$
- Cartesian product: $A \times B$ (set of all ordered pairs in the form (a, b))
- Function: $f : A \rightarrow B$
  (* Function from A to B is a subset of $A \times B$)

Set Theory..

Power Set

- $P(X)$ is the power set of X
  - Collection of all subsets of X
- $|X|$ is the number of elements in the set X
  - $|P(X)| = 2^{|X|}$
    - $A = \{x, y, z\}$
    - $P(A) = \{\{x\}, \{y\}, \{z\}, \{x, y\}, \{y, z\}, \{x, z\}, \{x, y, z\}, \{\}\}$

AUTOMATA

- Automata Theory: Study of abstract computing devices
- Abstract devices are (simplified) models of real computations
- Computations happen everywhere: On your laptop, on your cell phone, in nature, ...
- Used in model-checking

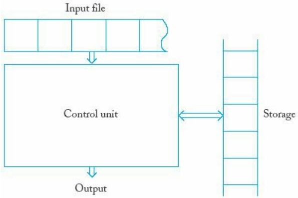
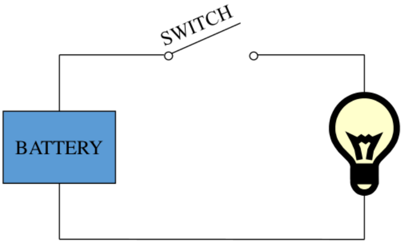


Figure: General model of an automata

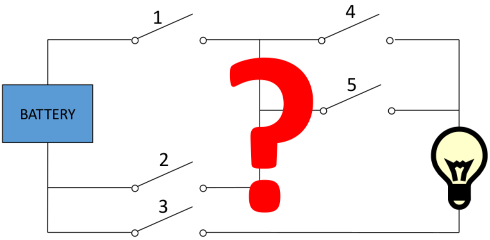
- Accepter: Automaton whose output response is limited to a simple "yes" or "no"
- Transducer: Automaton, capable of producing strings of symbols as output

A simple circuit



- **Input:** switch
- **Output:** light bulb
- **Actions / transitions :** flip switch
- **States:** on, off

A DESIGN PROBLEM



Can you design a circuit where the light is on if and only if all the switches were flipped exactly the same number of times?

A DESIGN PROBLEM...

- Such devices are difficult to reason about, because they can be designed in an infinite number of ways
- By representing them as abstract computational devices, or automata, we will learn how to answer such questions

Various types of Automata

Finite automata	Devices with a finite amount of memory. Used to model "small" computers.
Push-down automata	Devices with infinite memory that can be accessed in a restricted way. Used to model parsers, compiler for programming languges etc.
Turing Machines	Devices with infinite memory. Used to model any computer.
Time-bounded Turing Machines	Infinite memory, but bounded running time. Used to model any computer program that runs in a "reasonable" amount of time.

Finite Automata

- Computers with a limited amount of memory
- State based devices
- Examples: Timers, Door open / close, thermostat
- States acts as the memory

WHY STUDY FINITE AUTOMATA ?

- Used for design and verification of circuits and communication protocols
- Used for text-processing applications like text editors, t
- An important component of compilers - lexical analysers in programming languages
- Network Protocol Analysis
  - Ex: variable checking
- Identify simple patterns of events - DNA sequencing

Deterministic Finite Automata (DFA)

- For every combination of current state and an input symbol, there is precisely one next state
- Definition: A deterministic finite automata is defined by a 5-tuple

$$(Q, \Sigma, \delta, q_0, F)$$

  - Q is a finite set of states
  - Σ is a finite alphabet
  - $\delta : Q \times \Sigma \to Q$ is the transition function
  - $q_0 \in Q$ is the start state
  - $F \subseteq Q$ is the set of accepting states

DFA..

$M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w = w_1, w_2, ..w_n$ if there is a sequence of states $r_0, r_1, ...r_n$ iff

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}, \; \forall \; 0 \leq i \leq n-1$
3. $r_n \in F$

M recognizes the language A if

$$A = \{w| \; M \text{ accepts } w\}$$

denoted by $L(M) = A$

DFA..

$M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w = w_1, w_2, ..w_n$ if there is a sequence of states $r_0, r_1, ...r_n$ iff

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}, \forall\ 0 \leq i \leq n-1$
3. $r_n \in F$

M recognizes the language A if

$$A = \{w |\ M \text{ accepts } w\}$$

denoted by $L(M) = A$

Why it is called deterministic?

DFA..

$M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w = w_1, w_2, ..w_n$ if there is a sequence of states $r_0, r_1, ...r_n$ iff

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}, \forall\ 0 \leq i \leq n-1$
3. $r_n \in F$

M recognizes the language A if

$$A = \{w|\ M \text{ accepts } w\}$$

denoted by $L(M) = A$
Why it is called deterministic?
Unique transitions for each symbol read

Languages

- The language is a collection of appropriate strings, denoted by L
- L is a language over alphabet set Σ, only if $L \subseteq \Sigma^*$
- Examples:
  - If L takes all possible strings consisting of n 0's followed by n 1's over $\Sigma = \{0, 1\}$:
  - S $= \{\epsilon, 01, 00011, 0011, 0101, 000111, ...\}$
  - L $= \{\epsilon, 01, 0011, 000111, ...\}$ $\quad L \subseteq \Sigma^*$
- If L takes all possible strings of with equal number of 0's and 1's over $\Sigma = \{0, 1\}$:
- $L = \{\epsilon, 01, 10, 0011, 1100, 0101, 1010, 1001, ...\}$ $L = \{\}$,
- $\emptyset$ denotes the Empty language

Regular Language

- A language is regular if it is recognized by some finite automaton
- The collection of all strings that are recognized by a finite automata

Regular Operations

- Concatenation: AB $= \{xy|\ x \in A, y \in B\}$
  - x $=$ 010
  - y $=$ 1101
  - xy $=$ 010 1101
  - Language Concatenation: $L_1 = \{01, 00\}$, $L_2 = \{11, 010\}$
  - $L_1L_2 = \{01\ 11, 01\ 010, 00\ 11, 00\ 010\}$
  - L^n can be defined as L concatenated with itself n times
  - $L^0 = \{\lambda\}$
  - $L^1 = L$

- Union: A $\cup$ B $= \{x|x \in A$ or $x \in B\}$
  - $L_1 = \{01, 00\}$, $L_2 = \{01, 11, 010\}$
  - $L_1 \cup L_2 = \{01, 00, 11, 010\}$

- Star: $A^* = \{x_1, x_2, ...x_k \mid k \geq 0$ and, $x_i \in A$ for each $i\}$
- Star-closure of a language is
  - $L^* = L^0 \cup L^1 \cup L^2 ...$
- Positive Closure: $L^+ = L^1 \cup L^2 ...$
- Complement: $L^{\complement}$ or $(\overline{L}) = \Sigma^* - L$

Example: If $L = \{a^n b^n : n \geq 0\}$

$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}$

Regular languages are closed under regular operations

Grammars

- We need a mechanism to describe languages mathematically
- A grammar implies an algorithm that would generate all legal sentences of the language
- Grammar for the English language tells us whether a particular sentence is well formed or not

Example: A grammar that generates a subset of the English language

$$< sentence > \rightarrow < noun\_phrase > < predicate >$$
$$< noun\_phrase > = < article > < noun >$$
$$< predicate > = < verb >$$

Example

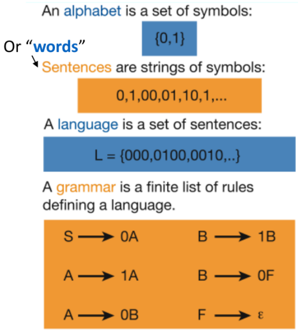
A derivation of "a dog runs"

- $$\begin{aligned} < sentence > &\rightarrow < noun\_phrase > < predicate > \\ &\rightarrow < noun\_phrase > < verb > \\ &\rightarrow < article > < noun > < verb > \\ &\rightarrow \text{a} < noun > < verb > \\ &\rightarrow \text{a dog} < verb > \\ &\rightarrow \text{a dog runs} \end{aligned}$$

Language of the grammar

L = { "a boy runs", "a boy walks", "the boy runs", "the boy walks", "a dog runs", "a dog walks", "the dog runs", "the dog walks" }

Languages and 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*

Definition of the formal grammar G

A grammar G is defined as a quadruple

$$G = (V, T, S, P)$$

- V: variables, set of non-terminal symbols, uppercase letters
- T: symbols set of terminal symbols, lower case letters and some special operators
  ex: arithmetic, relational operators
- $S \in V$: start variable
- P: set of production rules

Assumption: V and T are nonempty and disjoint

Production rules

- How the grammar transforms one string into another?
- They define a language associated with the grammar

$$x \rightarrow y$$

x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$

Given a string w of the form: $w = uxv$
Replacing x with y: thereby obtaining a new string

$$z = uyv$$

$$w \Rightarrow z$$

"w derives z" or "z is derived from w"

Example

- Grammar: G=(V,T,S,P)

$$G = \{\{S\},\{\text{ a, b }\}, S, P\}$$
$$P = \{S \to aSb, S \to \lambda\}$$

- Derivation of sentence :

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Example

$$G = \{\{S\}, \{a, b\}, S, \{S \to aSb, S \to \lambda\}\}$$
$$S \to aSb \mid \lambda$$

It generates:

- $$S \to \lambda$$
- $$\begin{aligned} S &\to aSb \\ &\to ab \end{aligned}$$
- $$\begin{aligned} S &\to aSb \\ S &\to aaSbb \\ S &\to aabb \end{aligned}$$
- $$\begin{aligned} S &\to aSb \\ S &\to aaSbb \\ S &\to aaaSbbb \\ S &\to aaabbb \end{aligned}$$

$$\mathsf{L} = \{\lambda, ab, aabb, aaabbb, ...\}$$

$$\mathsf{L} = \{a^n b^n, n \geq 0\}$$

Example 2

Find a grammar that generates

$$\mathsf{L} = \{a^n b^{n+1} : n \geq 0\}$$

Example 2

Find a grammar that generates

$$L = \{a^n b^{n+1} : n \geq 0\}$$

$$S \rightarrow Ab$$

Example 2

Find a grammar that generates

$$\mathsf{L} = \{a^n b^{n+1} : n \geq 0\}$$

S $\rightarrow$ Ab

A $\rightarrow$ aAb

Example 2

Find a grammar that generates

$$\mathsf{L} = \{a^n b^{n+1} : n \geq 0\}$$

S $\rightarrow$ Ab

A $\rightarrow$ aAb

A $\rightarrow$ λ

Practice Problems..

Find grammars for $\Sigma = \{a, b\}$ that generate the sets of

1. Strings of a's followed by b's

Practice Problems..

Find grammars for $\Sigma = \{a, b\}$ that generate the sets of

1. Strings of a's followed by b's

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

Practice Problems..

Find grammars for $\Sigma = \{a, b\}$ that generate the sets of

1. Strings of a's followed by b's

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

2. all strings with exactly one 'a'.

L = { ab, ba, bab, bbab, abbb, bbbba, .. }

Grammar $G = (V, T, S, P)$

Vocabulary

Terminal symbols

Start variable

Productions of the form:

$$A \to x$$

Non-Terminal

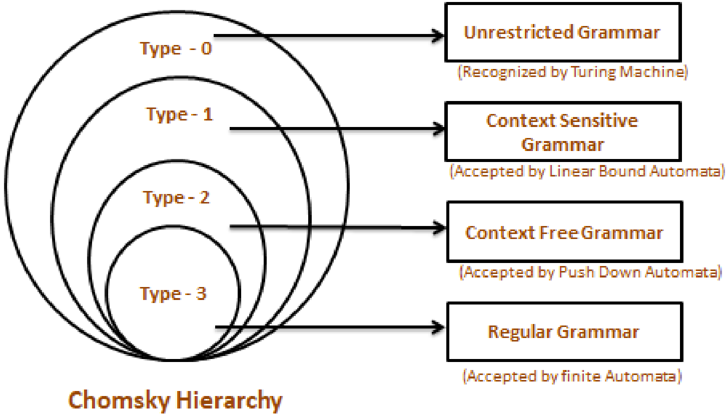
String of variables and terminals

Classification of formal grammars

Type	Name	Production rules	Recognizing automaton / Storage required / Parsing complexity
3	Regular grammars, Finite state grammars	A –> xB C –> y A, B, C – non-terminal symbols x, y – terminal symbols	Finite state automaton / Finite storage / *O (n)*
2	Context free grammars	A –> BC…D A – non-terminal symbols BC…D – any sequence of terminal or non-terminal symbols	Pushdown automaton / Pushdown stack / $O(n^3)$
1	Context sensitive grammars	aAz –> aBC…Dz A – non-terminal symbols a, z – sequences of zero or more terminal or non-terminal symbols BC…D – any sequence of terminal or non-terminal symbols	Linear bounded automaton (non-deterministic Turing machine) / Tape being a linear multiple of input length / *NP Complete*
0	Unrestricted grammars, General rewrite grammars	Allows the production rules to transform any sequence of symbols into any other sequence of symbols. To convert context-sensitive grammar into unrestricted grammar, replacement of any non-terminal symbol A with an empty sequence needs to be allowed.	Turing machine / Infinite tape / Undecidable

The Chomsky Hierachy

A containment hierarchy of classes of formal languages



Deterministic Finite Accepters (DFA)

M $= (Q, \Sigma, \delta, q_0, F)$

Transition Graphs

- To visualize and represent finite automata
- Vertices represent states
- The edges represent transitions

L $= \{a^n b : n \geq 0\}$

Transition table

	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

- Row label is the current state,
- Column label represents the current input symbol
- The entry in the table defines the next state

Transition table..

Example: Strings With No 11

Transition table..

Example: Strings With No 11

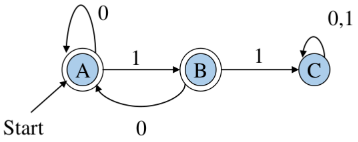
	0	1
→ * A	A	B
* B	A	C
C	C	C

Columns = input symbols

Rows = states

Each entry is δ of the row and column.

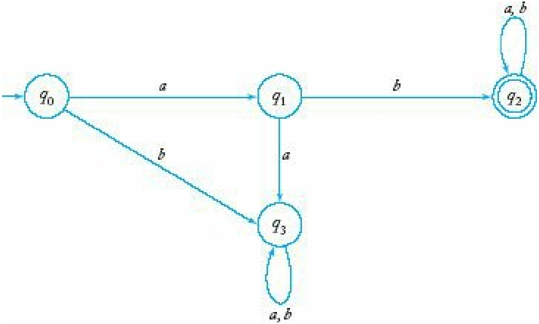
Final states - *
Start state - →



Example

Find a DFA that recognizes the set of all strings on $\Sigma = \{a, b\}$ starting with the prefix ab

Find a DFA that recognizes the set of all strings on $\Sigma = \{a, b\}$ starting with the prefix ab

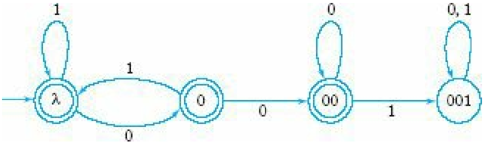


Example

Find a dfa that accepts all the strings on $\{0, 1\}$, except those containing the substring 001

Example

Find a dfa that accepts all the strings on $\{0, 1\}$, except those containing the substring 001



Regular Language: Examples

A language L is called regular iff there exists some deterministic finite accepter M such that

$$L = L(M)$$

Example: Show that the language is regular:

$$L = \{awa : w \in \{a, b\}^*\}$$

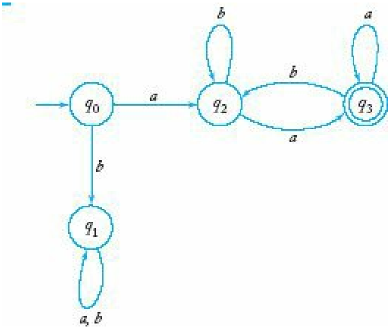
Regular Language: Examples

A language L is called regular iff there exists some deterministic finite accepter M such that

$$L = L(M)$$

Example: Show that the language is regular:

$$L = \{awa : w \in \{a,b\}^*\}$$



Regular Language: Examples..

Show that L^2 is regular

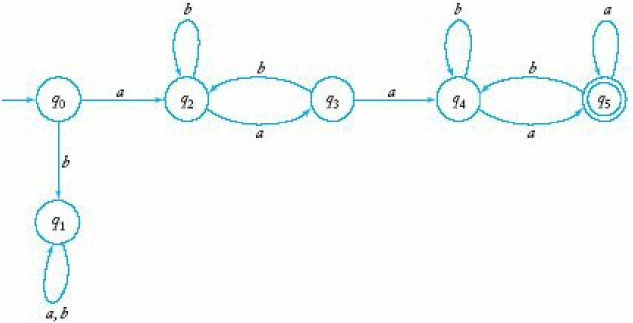
Regular Language: Examples..

Show that L^2 is regular

$$L = \{aw_1aaw_2a : w_1, w_2 \in \{a,b\}^*\}$$

Show that L^2 is regular

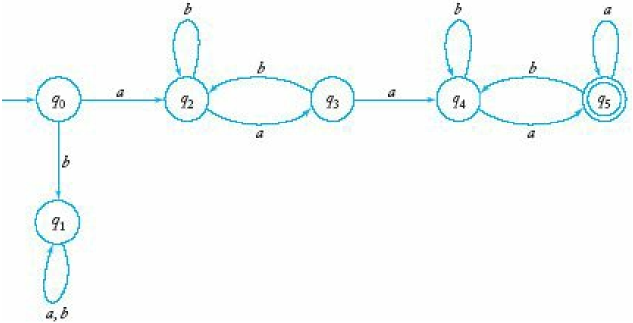
$$L = \{aw_1aaw_2a : w_1, w_2 \in \{a,b\}^*\}$$



Regular Language: Examples..

Show that L^2 is regular

$$L = \{aw_1aaw_2a : w_1, w_2 \in \{a,b\}^*\}$$



if a language L is regular, so are L^2 ,L^3 ,....

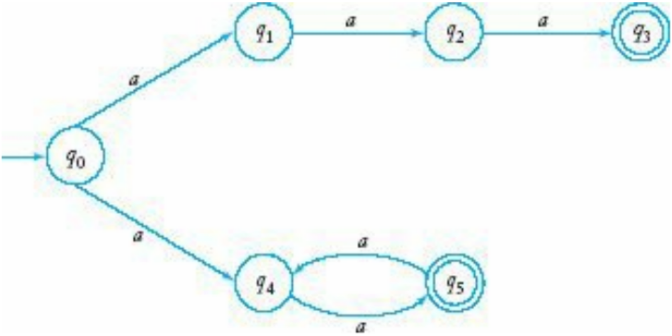
A nondeterministic finite accepter or nfa is defined by the quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

$\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$

Nondeterministic Finite Accepters (NFA)..

- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ϵ transition.
- Nondeterminism: the automaton can be in multiple states simultaneously and can follow multiple possible transitions for the same input symbol.



NFA and DFA

- DFA has exactly one transition for each state q $\in$ Q and symbol $a \in \Sigma$.
  - NFA can have 0, 1, or more than 1
- NFA it can make transitions with out reading input signal (ϵ)
- NFA can have multiple choices at each state. (It could have many possible computation paths)
- NFA accepts a string if there is at least one accepting computation path

Why NFA?

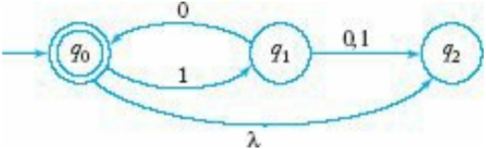
- An exhaustive search with backtracking
  - Eg: playing chess
- When several alternatives are possible, we choose one and follow it until it becomes clear whether or not it was best.
- If not, we retreat to the last decision point and explore the other choices
- NFA machines can serve as models of search-and-backtrack algorithms
- NFA is helpful in solving problems easily

Why NFA?

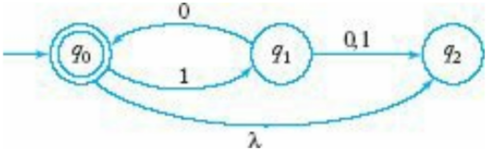
- An exhaustive search with backtracking
  - Eg: playing chess
- When several alternatives are possible, we choose one and follow it until it becomes clear whether or not it was best.
- If not, we retreat to the last decision point and explore the other choices
- NFA machines can serve as models of search-and-backtrack algorithms
- NFA is helpful in solving problems easily

Every DFA is NFA

Examples

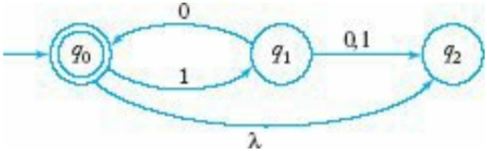


Examples



$L = \{\lambda, 1010, 101010, ...\}$

Examples



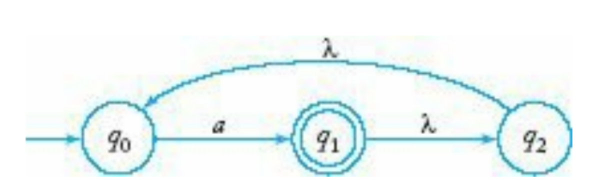
$L = \{\lambda, 1010, 101010, ...\}$

$L = \{(10)^n : n \geq 0\}$

- Extended transition function $\delta^*(q_i, w) = Q_j$
- Q_j is the set of all possible states the automaton may be in, having started in state q_i and having read w.

- Extended transition function $\delta^*(q_i, w) = Q_j$
- Q_j is the set of all possible states the automaton may be in, having started in state q_i and having read w.

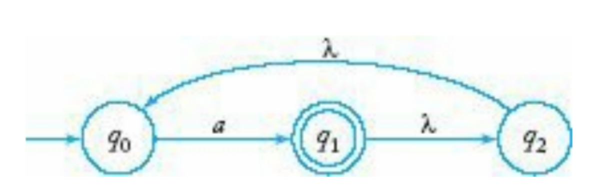
Example



Find $\delta^*(q_0, a)$, $\delta^*(q_2, \lambda)$.

- Extended transition function $\delta^*(q_i, w) = Q_j$
- Q_j is the set of all possible states the automaton may be in, having started in state q_i and having read w.

Example

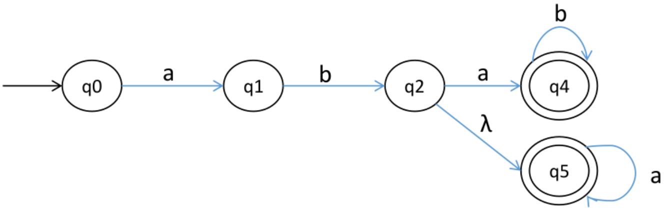


Find $\delta^*(q_0, a)$, $\delta^*(q_2, \lambda)$.
$\delta^*(q_0, a) = \{q_0, q_1, q_2\}$
$\delta^*(q_2, \lambda) = \{q_0, q_2\}$

Examples

- Draw an NFA with no more than 5 states for

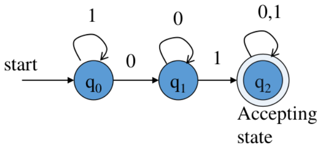
$$L = \{abab^n : n \geq 0\} \cup \{aba^n : n \geq 0\}$$



DFA for strings containing 01

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

- 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

symbols

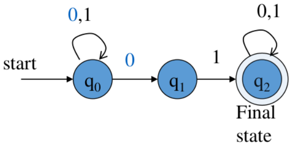
δ (states)	0	1
→ q_0	q_1	q_0
q_1	q_1	q_2
*q_2	q_2	q_2

Examples

NFA for strings containing 01

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

Non-deterministic - multiple states



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

symbols

δ	**0**	**1**
$\rightarrow$ **q_0**	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
$*q_2$	$\{q_2\}$	$\{q_2\}$

states

The language L accepted by an nfa $M = (Q, \Sigma, \delta, q_0, F)$

$$\text{L(M)} = \{w \in \Sigma^* : \delta^*(q_0, w) \bigcap F \neq \Phi\}$$

The language consists of all strings w for which there is a walk labeled w from the initial vertex of the transition graph to some final vertex

Examples..

Construct an NFA that accepts the language $\{ab, abc\}*$

Examples..

Construct an NFA that accepts the language $\{ab, abc\}*$