CLASS 1 DATE MKS 15.10.20

Reading ← ① An Introduction to Formal Language and Automata – Peter Linz
② Hopkoff, Motwani
↳ studying + Video Lecture slides

Formal Languages and Automata Theory

Automata: self acting
deals with the study of various abstract and mathematical machines and also the problems that can be solved.

Computing Machines

Abstract machines, not used for energy conversion
→ you can compute as long as you want whether it getting heated up.

Finite or infinite set of strings of symbols → formal language
↓
From a finite set of symbols = alphabet

- Given a mathematical sentence, whether there is a method to validate the mathematical statement

David Hilbert • The Roots of Automata Theory
→ Work of Logicians in 1930s
→ Is a step by step solution (algorithm) possible to determine the truth or falsity of any mathematical proposition expressed in 1st order predicate calculus applied to integers
→ In 1931, Kurt Goden through his incompleteness theorem proved that no such algorithm could exist

Standard model of computation → came out/developed in 1930s

Turing Machine is the most important one.
↙
Alan Turing Turing Machine is accepted as a theoretical model of a computer as per church's hypothesis.
1930s
In 1950s, few more simpler machines (Finite Automata) were also developed – to model the human brain.

Later applied to computational problems → Lexical Analysis: if you've named the variable in program using rules.

In 1950s,

Noam Chomskey developed formal language

Specific reasons to study automata theory:

→ Automata are essential for the study of limits of computation

→ Two important issues:

1. What can a computer do at all?

This study is called "decidability" and the problems that can be solved by computers are called "decidable"

2. What can a computer do efficiently?

This study is called "intractability" and the problems that are to be solved by a computer using no more time than some slowly growing function of the size of the input are called "tractable"

Often we take all polynomial functions to be slowly growing.

What is our Ultimate Objective:

It was initiated to find the solution to a problem.
We'll study about the machines and the problems that can and cannot be solved on these machines.

Halting Problem: Is their any algorithm which when supplied with another algorithm and input can determine if the algorithm will halt with that input.

↓

cannot be solved with available computing machine.

Travelling Salesman Problem: no efficient solution

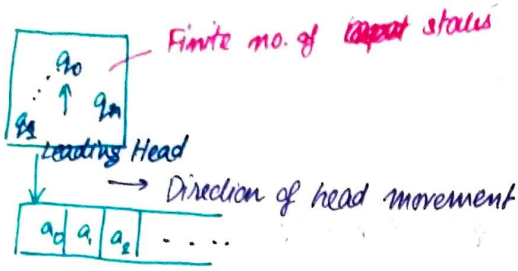
Automata (plural)

Automaton (~~signal~~ singular)

In what form, finite automata accepts input?

String of symbols

Finite Automata



→ supply as much memory as you require
→ computing power as you require

Fig: A finite Automaton

- Is it possible to have an algorithm to determine the truth of an expression?
  ↳ you have to make the machine free from all constraints

Alphabet

An alphabet is any finite set of symbols

Example:

$\{0,1\}$: binary alphabet

$\{a,b,c\}$ etc

Strings

A string is a finite sequence of symbols chosen from some alphabet.

If Σ is an alphabet, we can express the total no. of all strings of certain length from that alphabet by using an exponential notation, eg

Σ^k = the set of all strings of length k taken from Σ

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

$\Sigma^0 = \{\epsilon\}$ (sometimes λ)

$\Sigma^1 = \{1,0\}$ $\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$

= set of all possible strings possible strings which can be formed with the symbols chosen from sigma.

Countably infinite set

$= \Sigma^*$, $\Sigma^0 \cup \Sigma^+ = \{\epsilon\} \cup \Sigma^+$

① If you can enumerate all strings of size n, someone can enumerate all strings of size (n+1). ⇒ countably infinite

Σ^*: set of all strings made from alphabet excluding the empty string.

Language

A language is a subset of Σ^* for some to alphabet Σ

Example:

The set of all strings of 0s and 1s with no two consecutive 1s

$L = \{ \epsilon, 0, 1, 00, 01, 10, 000, 001, 010, 100, 101, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010, \dots \}$

$L \subseteq \Sigma^*$

Deterministic Finite Automata (DFA)

A DFA M is a five tuple; ie; $M = (Q, \Sigma, \delta, q_0, F)$

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

~~Ex: all binary strings which do not have two consecutive~~

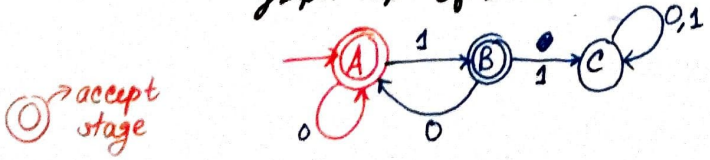
(Input Symbol
read, current state)

⇓

FA makes a movement [sequential circuit design]

- With every state FA registers a particular event

graph rep of DFA



state diagram of Deterministic FA which accepts all inputs which have no consecutive 1s

State Table

An alternative representation

State Transition Table

State	Input 0	1
A*	A	B
B*	A	C
C	C	C

[same string ⇒ gives same sequence of state transitions
↳ deterministic

Formal definition of DFA

It is a five tuple $(Q, \Sigma, \delta, q_0, F)$

where

Q = a finite set of states

Σ = an input alphabet

δ = a transition function $\delta(q, a)$ = the state that the DFA goes to when it is in state q and input a is received

q_0 in Q = the start state

$F \subseteq Q$ = the state of final states

Extended Transition Function

→ We describe the effect of a string of inputs on a DFA by extending δ to a state and a string

→ Induction on length of string

Basis: $\delta(q, \epsilon) = q$

Induction = $\delta(q, wa) = \delta(\delta(q, w), a)$

where w = a string
a = an input symbol

Example: Extended Delta

$$\delta(B, 011) = \delta(\delta(B, 01), 1) = \delta(\delta(\delta(B, 0), 1), 1)$$
$$= \delta(\delta(A, 1), 1)$$
$$= \delta(B, 1)$$
$$= C$$

Language of a DFA

for a DFA, $L(A)$ is the set of strings labelling paths from the start state to a final state

Formally,

$L(A)$ = the set of strings w such that $\delta(q_0, w)$ is in F.

$F = \{A, B\}$

Example: string in a language, 101 is in $L(M)$?

$A \to B \to A \to B$ ✓ in the language

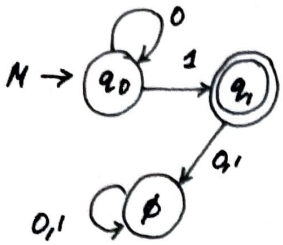
⑥

$\Sigma = \{A, B\}$

$L(M)$ = the language of DFA M

= $\{ w \mid w \text{ in } \{0,1\}^* \text{ and } w \text{ does not have two consecutive 1's} \}$

$\Sigma \hookrightarrow \Sigma^{(M)}$



$L(M)$: language accepted by this DFA $M = S$

T = the set of all binary string's beginning any number of 0s followed by a single 1

Q. Prove $S = T$

1. $S \subseteq T$
2. $T \subseteq S$

First let us prove $S \subseteq T$

IH : strings $1, 01, 001, \ldots$ in S are also in T.

So for any string x, $|x| < m$, in S is also in T.

That is, for any string x, $|x| < m$ if $\delta(q_0, x) = q_f$ then x is in T.

Let us consider a ax in S

Now $\delta(q_0, ax) = \delta(\delta(q_0, a), x) = q_f$

Since x is already accepted by M, $\delta(q_0, a) = q_0$

If $\delta(q_0, a) = q_0$ then a must be 0

So $\delta(N_0, 0x) = q_f$

Now $0x$ = a string beginning with one or more 0s followed by a 1.

So $0x$ in T

Next Let us prove $T \subseteq S$

IH : For x in T, $|x| < m$, x is in S

That is, $\delta(q_0, x) = q_f$

Consider a string ax in T

Since $|ax| = m > 1$, a must be 0

⑦ For $a=0$, $\delta(q_0, ax) = \delta(\delta(q_0,a), x)$

$= \delta(q_0, x) = q_f$

Hence ax in S

& $T \subseteq S$

Finally we prove that $S = T$

Regular Languages

A language L is regular if it is the language accepted by some DFA.

Note: the DFA must accept only the strings in L, no other

Some languages are not regular

Intuitively, regular languages cannot count to arbitrarily high integers.

Example : A non regular language

$L_1 = \{ 0^n 1^n \mid n \geq 1 \}$

$L_2 = \{ w \mid w \text{ in } \{(,)\}^* \text{ and } w \text{ is balanced} \}$

eg, (), (()), ()(), (()()), ...

Example : A Regular Language

$L_3 = \{ w \mid w \text{ in } \{0,1\}^*$ and w, viewed as a binary integer divisible by 5 $\}$

↓

0, 101, 1010

The DFA
has 5 states named as 0, 1, 2, 3, 4
*start and only final state is named 0

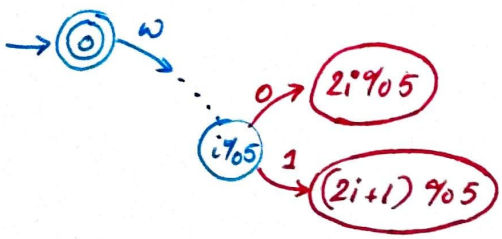
① If string w represents an integer i then assume $\delta(0,w) = i$

② $w0$ represents $2i$

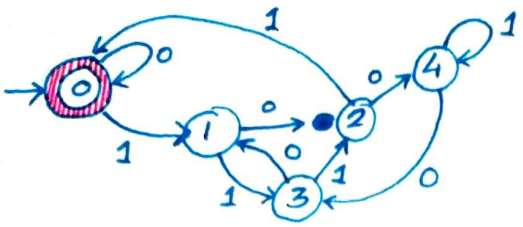
So we want $\delta(i\%5, 0) = 2i\%5$

③ $w1$ represents $(2i+1)$

So we want $\delta(i\%5, 1) = (2i+1)\%5$



(8)



Non Deterministic Finite Automata (NFA)

- An NFA has the ability to be in several states at once.
- Transition from a state on an input symbol can be to any set of states
- An NFA starts in one start state
- It accepts if any sequence of choices leads to a final state.

Example : Chessboard

Notes on a chessboard

→ States = squares

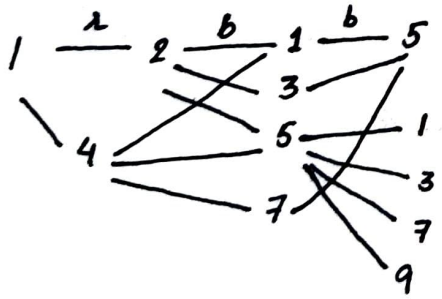
→ Inputs = r (move to an adjacent red square)
and b (move to an adjacent black square)

→ start state, final state are in opposite corners

state	Input r	Input b
→ 1 →	2,4	5
→ 2 →	4,6	1,3,5
→ 3 →	2,6	5
→ 4 →	2,8	1,5,7
→ 5 →	2,4,6,8	1,3,7,9
→ 6 →	2,8	3,5,9
→ 7 →	4,8	5
→ 8 →	4,6	7,5,A
→ 9* →	6,8	5

↳ accepted state

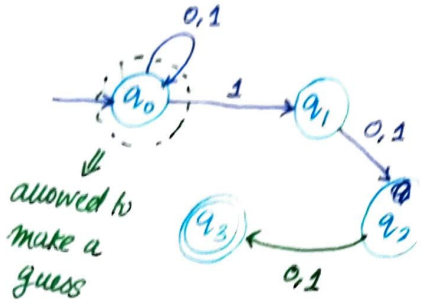
(9) states



(9)

Example Give an NFA accepting all binary strings containing a 1 in the third position from the end.

Long 000100 ✓
001100 ✓
111011 ✗
010010 ✗



Formal NFA

① A finite set of states typically Q.
② An input alphabet typically Σ
③ A transition function, typically δ
④ A start state in Q, typically q_0
⑤ A set of final states $F \subseteq Q$
⇒ ⑥ Transition Function of an NFA
  ① $\delta(q,a)$ is a set of states
  ② Extended δ to strings as follows
  ③ Basis : $\delta(q, \epsilon) = \{q\}$
  ④ Induction : $\delta(q, wa\}$ = the union over all states p in $\delta(q,w)$ of $\delta(p,a)$

(wa: string + symbol)

for example:

- $\delta(1, ab)$
  $= \delta(2,b) \cup (\delta(4,b)$
  $= \{1,3,5\} \cup \{1,5,7\}$
  $= \{1,3,5,7\}$

- 

String of only b's.

Odd bs ⇒ wont be accepted
even bs ⇒ will be accepted

Language of a NFA

- A string w is accepted if $\delta(q_0, w)$ contains at least one final state

The language of an NFA is the set of strings it accepts.

⑤ ⑥ In certain it is better to design an NFA instead of a DFA because in NFA you can make a guess, if the guess fails you can guess again

However this does not mean NFA is more powerful than DFA.

(10) Equivalence of DFA's, NFA's

- ⊛ A DFA can be turned into an NFA that accepts the same language.
- ⊛ If $\delta_D(q,a) = r$, let the NFA have $\delta_N(q,a) = \{r\}$
- ⊛ Then the NFA is always in a set containing exactly one state - the state the DFA is in after reading the same input.

given

L → DFA → accept

Find $\delta_D(q,a) = r$

equivalent

L → NFA → accept

$\delta_N(q,a) = \{r\}$

Subset Construction

- given an NFA with states Q, input alphabet Σ, transition function δ_N, start state q_0, and a set of final states F, let us construct an equivalent DFA with:
  - → states 2^Q (set of subsets of Q)
  - → Input alphabet Σ
  - → start state $\{q_0\}$
  - → Final states - all those with a member of F

→ The DFA states have names that are sets of NFA states.

→ But as a DFA state an expression like $\{p,q\}$ must be read as a single symbol not a set.

δ_N known; δ_D = ?

→ The transition function δ_D is defined by $\delta_D(\{q_1, \dots q_k\}, a)$ is the union over all $i = 1, \dots k$ of $\delta_N(q_i, a)$

Example: We'll construct the DFA equivalent to our "chessboard" NFA.

state	Input a	b
$\{q_1\}$	$\{2,4\}$	$\{5\}$
$\{2,4\}$	$\{2,4,6,8\}$	$\{1,3,5,7\}$
$\{5\}$	$\{2,4,6,8\}$	$\{1,3,7,9\}$
$\{2,4,6,8\}$	$\{2,4,6,8\}$	$\{1,3,5,7,9\}$
$\{1,3,5,7\}$	$\{2,4,6,8\}$	$\{5\}$
$\{1,3,5,7,9\}$	$\{2,4,6,8\}$	$\{1,3,5,7,9\}$

$\delta(\{2,4\}, a) = \{4,6\} \cup \{2,8\}$
$= \{2,4,6,8\}$

(11) Proof of Equivalence : "Subset construction"

Let us show by induction on $|w|$ that

$$\delta_N(q_0, w) = \delta_D(\{q_0\}, w)$$

Basis : for $w = \epsilon$, $\delta_N(q_0, \epsilon) = \delta_D(\{q_0\}, \epsilon) = \{q_0\}$

Assume Induction Hypothesis for strings shorter than w

Let $w = xa$, Induction Hypothesis for x

$$\delta_N(q_0, x) = \delta_D(\{q_0\}, x) = S \;\|\; \delta_D(\{q_0\}, xa) = \ldots$$

Let T = the union over all states p in S of $\delta_N(p, a)$

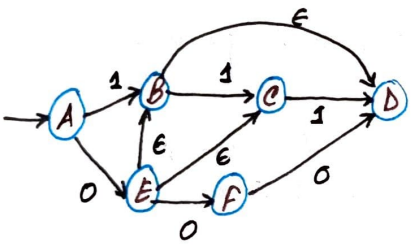
Then $\delta_N(q_0, w) = \delta_D(\{q_0\}, w) = T$

Every NFA can be converted to a DFA.

NFA's with ϵ-Transitions

In such NFA, state to state transitions with ϵ or zero length string are allowed

Example ϵ - NFA



State	Input 0	1	ε
→ A	{E}	{B}	∅
B	∅	{C}	{D}
C	∅	{D}	{∅}
D	∅	∅	∅
E	{F}	∅	{B, C}
F	{D}	∅	∅

Closure of sets

$cl(q)$ = set of states that the m/c can reach from state q following only arcs labelled ϵ.

$cl(A) = \{A\}$

$cl(E) = \{B, C, D, E\}$

Closure of a set of states = Union of the closure of each state

Extended Delta

- Basis : $\delta(q, \epsilon) = cl(q)$
- Induction : $\delta(q, xa)$ is computed as follows
  1. Start with $\delta(q, x) = S$
  2. Take the union of $cl(\delta(r, a))$ for all r in S

Intuitively $\delta(q, w)$ is the set of states where the machine can reach from following a path labelled w.

Example: Extended Delta

$$\delta(A,\epsilon) = cl(\{A\}) = \{A\}$$

$$\hat{\delta}(A,0) = \{E\} = cl(\{E\}) = \{B,C,D,E\}$$

$$\hat{\delta}(A,1) = \{B\} = cl(\{B\}) = \{D\}$$

$$\hat{\delta}(A,01) = \hat{\delta}(\hat{\delta}(A,0),1) = cl(\{C,D\}) = \{C,D\}$$

Language of an ϵ NFA is the set of strings such that $\hat{\delta}(q_0,w)$ contains a final state

Equivalence of NFA, ϵ NFA

- Every NFA is an ϵ-NFA
  - It just has no transitions on ϵ
- Converse requires us to take an ϵ-NFA and construct an NFA that accepts the same language
- We do so by combining ϵ transitions with the next transition on a real input

NFAs with ϵ Transitions

In such NFA, state to state!

- Start with an ϵ-NFA with states Q, inputs Σ, start state q_0, final states F and transition function δ_E.
- Construct an ordinary NFA with states Q, inputs Σ, start state q_0, final states F' and transition function δ_N.

Compute $\delta_N(q,a)$ as follows:

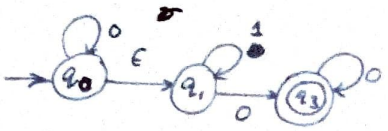
1. let $s = cl(q)$
2. $\delta_N(q,a)$ is the union over all r in s of $\delta_E(r,a)$

F' = the set of states q such that $cl(q)$ contains a state of F.

$\delta_E(E,0) = \{F\}$

$cl(E) = \{E,B,C,D\} = \{F\}$

(13) $\delta_E(E,0) \cup \delta_E(B,0) \cup \delta_E(C,0) \cup \delta_E(D,0) = \{F\}$

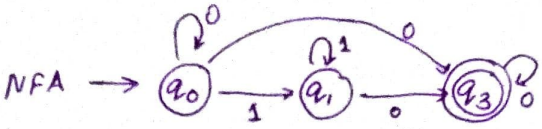


ε - NFA

Language of this NFA

~~$\{0\}^*\{\epsilon\}\{1\}^*\{0\}^*$~~

$\{0\}^*\{\epsilon\}\{1\}^*\{0\}^+$



$\delta_N(a_0,0) = ?$ $\{a_0, a_3\}$ $\delta_N(a_0,1) = ?$ $\{a_1\}$

~~$cl(a_0) = \{a_0, a_1\}$~~

~~$\delta_E(a_0,0) = a_0$ $\delta_E(a_0,1)$~~

~~$\delta_E(a_1,0) =$~~

~~$\delta_E(a_0,1) = \{a_1\}$~~

$cl(a_0) = \{a_0, a_1\}$

$\delta_E(a_0,0) \cup \delta_E(a_1,0) = \{a_0, a_3\}$

$cl(a_1) = \{a_1\}$

$\delta_N(a_1,1) = a_1$

$cl(a_3) = \{a_3\}$

$\delta_N(a_3,1) = \emptyset$ $\delta_N(a_3,0) = \{a_3\}$

$cl(F) = cl(a_3) = \{a_3\}$

a_3 will be the final state of the NFA

Regular Expressions (REs): Introduction

- Regular expressions are an algebric way to describe languages
- They describe exactly the regular languages
- If E is a regular expression then L(E) is the language it defines.

RE's Definition:

Basis 1: If a is any symbol then a is a RE, and $L(a) = \{a\}$

Basis 2: If ϵ is a RE then $L(\epsilon) = \{\epsilon\}$

Basis 3: If $\emptyset$ is a RE then $L(\emptyset) = \{\}$

Induction 1: If E_1 and E_2 are REs then E_1+E_2 is a RE and $L(E_1+E_2) = L(E_1) \cup L(E_2)$

Induction 2: If E_1 and E_2 are REs then $E_1 \cdot E_2$ is a RE and

$$L(E_1 \cdot E_2) = L(E_1)\,L(E_2)$$

↑ concatenation: the set of strings wx such that w is in $L(E_1)$ and x is in $L(E_2)$

Induction 3: If E is a RE then E^* is a RE and $L(E^*) = (L(E))^*$

↑ closure or "Kleene closure" = set of strings $w_1, w_2 \ldots w_n$ for some $n \geq 0$ where each w_i is in $L(E)$

Precedence of Operators

( )
*
·
+

Highest to lowest order of precedence

Examples : REs

$L(01) = \{01\}$

$L(01+0) = \{01, 0\}$

$L(0(1+0)) = \{01, 00\}$

$L(0^*) = \{0, 00, 000, \ldots\}$

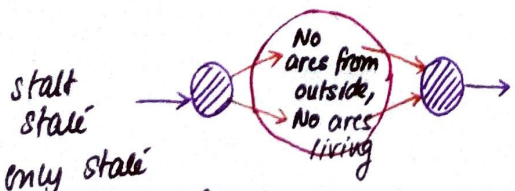
$L((0+10)^*(\varepsilon+1))$ = all binary strings without 2 consecutive 1s

⊛ Regular Expression and Automata are Equivalent

Converting a RE to an ε-NFA

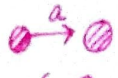
Here the proof is an induction on the number of operators (+, concatenating, *) in the RE

Assumed form of ε-NFA's constructed



(15)

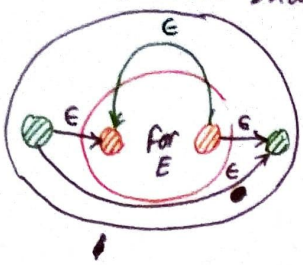
RE to ϵ NFA - Basis

Symbol a : 

ϵ : 

∅ : 

RE to ϵ NFA : Induction 3 - Closure



For E^* = ~~(L(E))*~~ $(L(E))^*$

w is in $L(E^*)$

w = xy where

x is in L(E) any

y is in L(E)

→ For every FA, there is one RA whose language is the same as FA

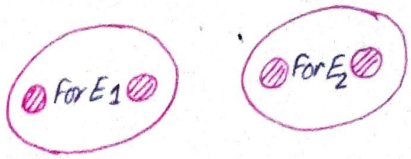
we ~~also~~ have to do the proof

Ex :

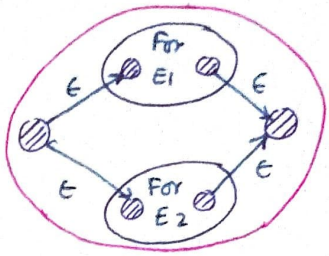
Convert $(0+1)^*1(0+1)$ to an ϵ-NFA

RE to ϵ-NFA :

Induction 1 - Union

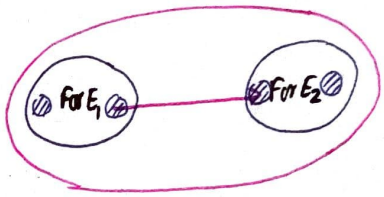


$E_1 + E_2 = L(E_1) \cup L(E_2)$



Induction 2 - Concatenation

For $E_1 \cdot E_2$



Input string ↔ x.y

x in $L(E_1)$ & y in $L(E_2)$