FORMAL LANGUAGES AND AUTOMATA THEORY

UNIT 1

Introduction to Automata Theory

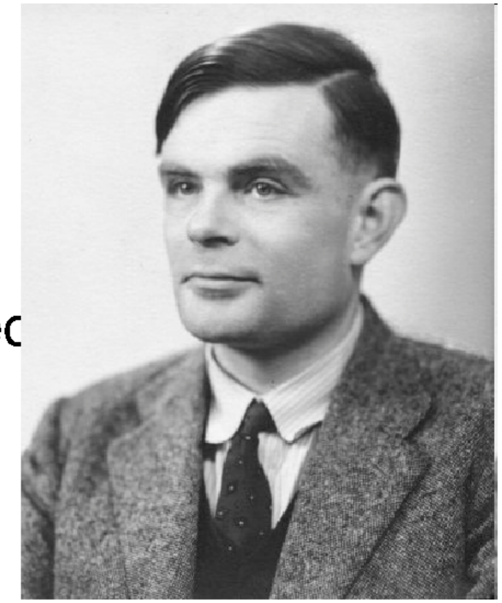
What is Automata Theory?

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

(A pioneer of automata theory)

Alan Turing (1912-1954)

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



Languages & Grammars

Or "**words**"

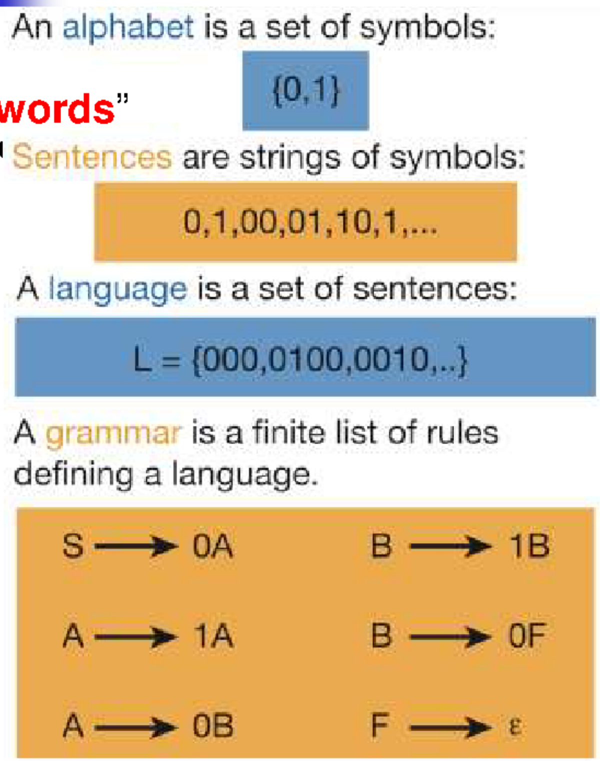
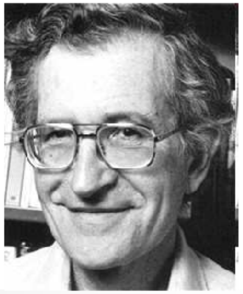


Image source: Nowak et al. Nature, vol 417, 2002

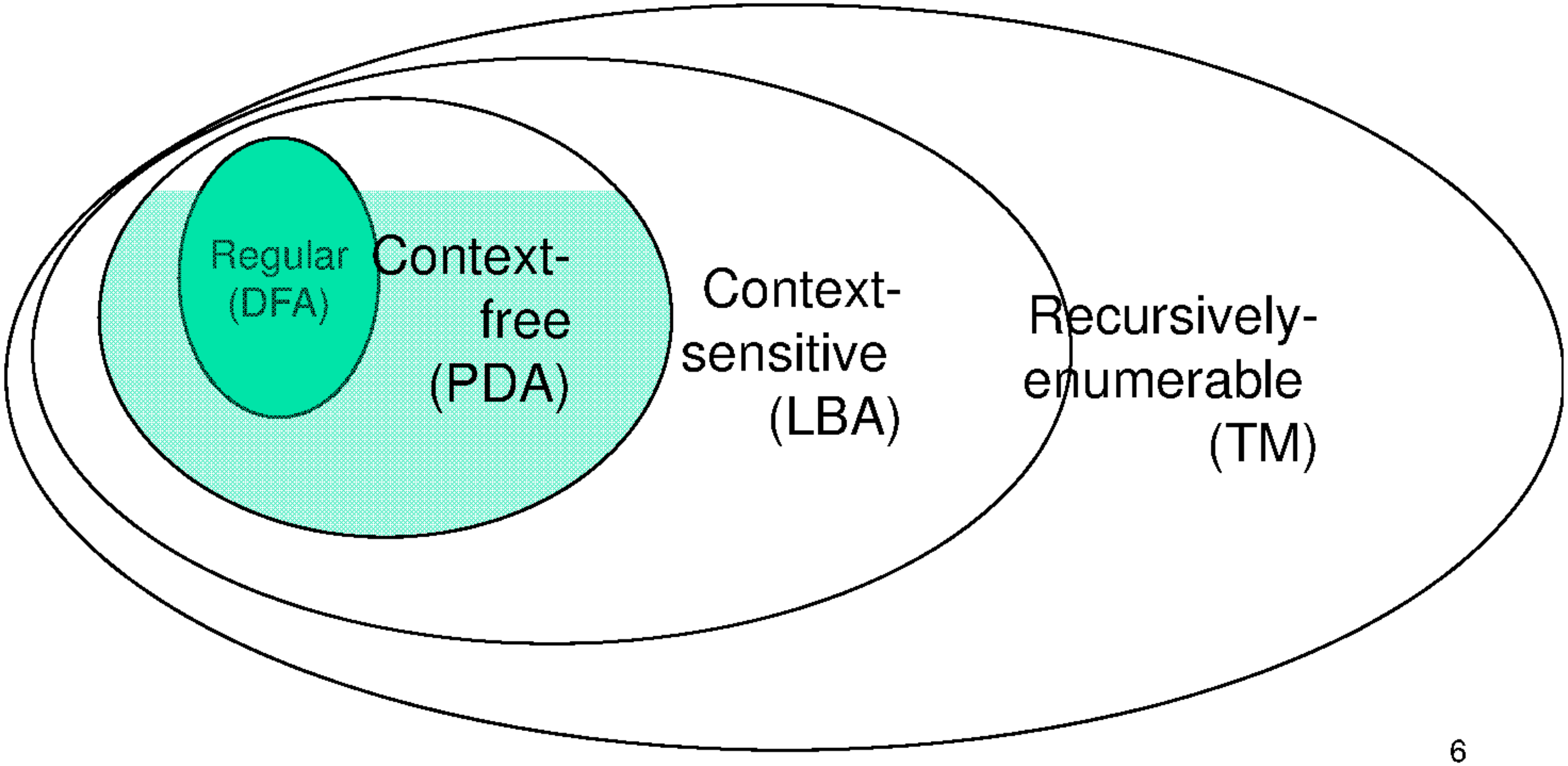
- <u>Languages</u>: "*A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols*"
- <u>Grammars</u>: "*A grammar can be regarded as a device that enumerates the sentences of a language*" - nothing more, nothing less

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

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



The Central Concepts of Automata Theory

Alphabet

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

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

Strings

A string or word is a finite sequence of symbols chosen from ∑

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

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

- *xy* = concatentation of two strings *x* and *y*

Powers of an alphabet

Let Σ be an alphabet.

- Σ^{k} = the set of all strings of length k
- $\Sigma^{*} = \Sigma^{0} \cup \Sigma^{1} \cup \Sigma^{2} \cup \dots$
- $\Sigma^{+} = \Sigma^{1} \cup \Sigma^{2} \cup \Sigma^{3} \cup \dots$

Languages

*L is a said to be a language over alphabet ∑, only if L ⊆ ∑**

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

Examples:

1. Let L be *the* language of all strings consisting of *n* 0's followed by *n* 1's:

   L = {ε,01,0011,000111,…}

2. Let L be *the* language of all strings of with equal number of 0's and 1's:

   L = {ε,01,10,0011,1100,0101,1010,1001,…}

Definition: Ø denotes the Empty language

- Let L = {ε}; Is L=Ø? NO

The Membership Problem

Given a string $w \in \Sigma^*$ and a language L over Σ, decide whether or not $w \in L$.

<u>Example:</u>

Let w = 100011

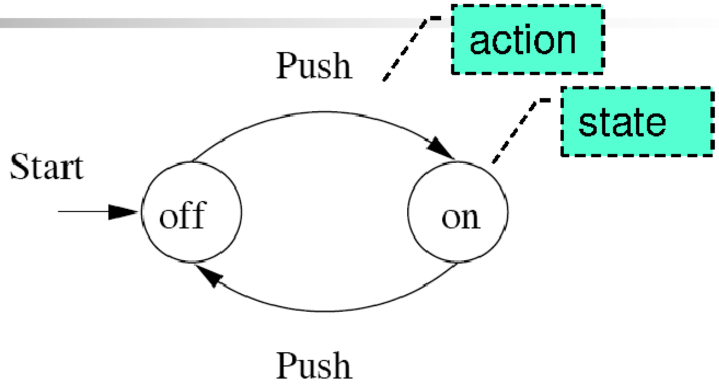
Q) Is $w \in$ the language of strings with equal number of 0s and 1s?

Finite Automata

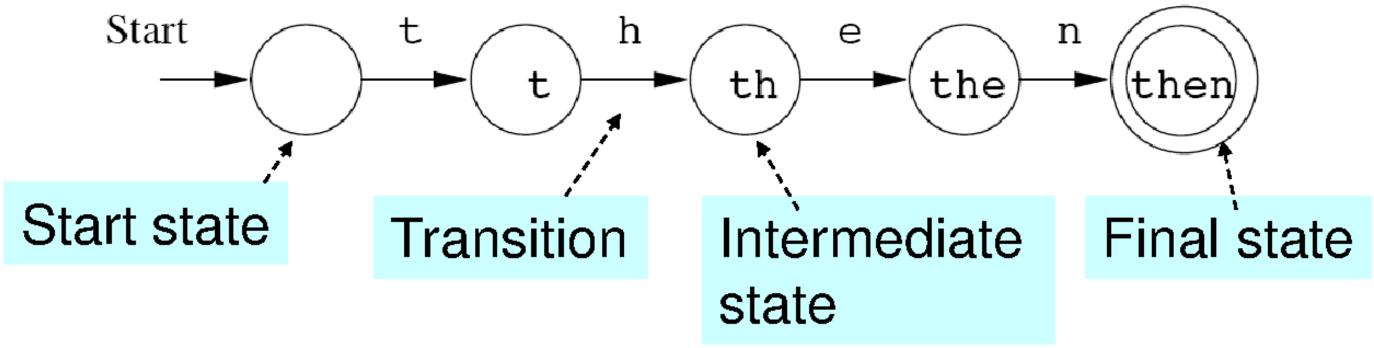
- Some Applications
  - Software for designing and checking the behavior of digital circuits
  - Lexical analyzer of a typical compiler
  - Software for scanning large bodies of text (e.g., web pages) for pattern finding
  - Software for verifying systems of all types that have a finite number of states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

Finite Automata : Examples

- On/Off switch



- Modeling recognition of the word "*then*"



Structural expressions

- Grammars
- Regular expressions
  - E.g., unix style to capture city names such as “Palo Alto CA”:
    - [A-Z][a-z]*([ ][A-Z][a-z]*)*[ ][A-Z][A-Z]

Start with a letter

A string of other letters (possibly empty)

Other space delimited words (part of city name)

Should end w/ 2-letter state code

Summary

- Automata theory & a historical perspective
- Chomsky hierarchy
- Finite automata
- Alphabets, strings/words/sentences, languages
- Membership problem

Finite Automata

Finite Automaton (FA)

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

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

Deterministic Finite Automata - Definition

- A Deterministic Finite Automaton (DFA) consists of:
  - Q ==> a finite set of states
  - Σ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of final states
  - δ ==> a transition function, which is a mapping between $Q \times \Sigma$ ==> Q
- A DFA is defined by the 5-tuple:
  - $\{Q, \Sigma, q_0, F, \delta\}$

What does a DFA do on reading an input string?

- Input: a word w in Σ^*
- Question: Is w acceptable by the DFA?
- Steps:
  - Start at the "start state" q_0
  - For every input symbol in the sequence w do
    - Compute the next state from the current state, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed, the current state is one of the final states (F) then *accept w;*
  - Otherwise, *reject w.*

Regular Languages

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

- Locate regular languages in the Chomsky Hierarchy

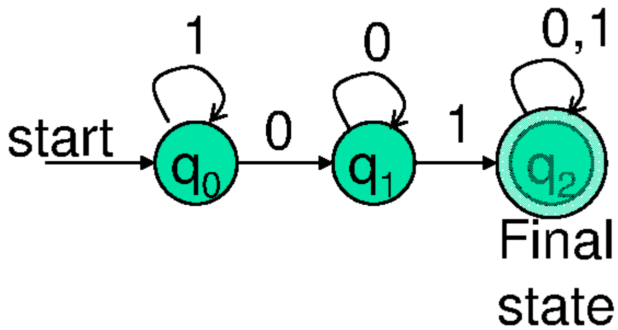
Example #1

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

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

DFA for strings containing 01

- What makes this DFA deterministic?



- What if the language allows empty strings?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

symbols

δ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
$*q_2$	q_2	q_2

states

Example #2

Clamping Logic:

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

- Build a DFA for the following language:
  L = { w | w is a bit string which contains the substring 11}
- State Design:
  - q_0 : start state (initially off), also means the most recent input was not a 1
  - q_1: has never seen 11 but the most recent input was a 1
  - q_2: has seen 11 at least once

Example #3

- Build a DFA for the following language:

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

- ?

Extension of transitions (δ) to Paths ($\hat{\delta}$)

- $\hat{\delta}$ *(q,w) = destination state* from state *q* on input string *w*

- $\hat{\delta}(q,wa) = \delta(\hat{\delta}(q,w), a)$

  - Work out example #3 using the input sequence w=10010, a=1:

    - $\hat{\delta}(q_0,wa) = ?$

Language of a DFA

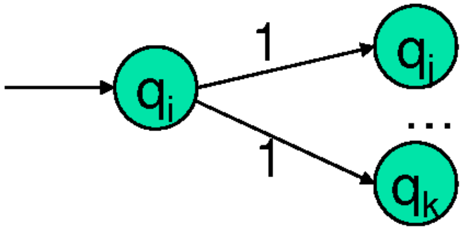
A DFA A accepts string *w* if there is a path from q_0 to an accepting (or final) state that is labeled by *w*

- *i.e.,* $L(A) = \{ w \mid \hat{\delta}(q_0,w) \in F \}$

- *I.e., L(A) = all strings that lead to a final state from* q_0

Non-deterministic Finite Automata (NFA)

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



• Each transition function therefore maps to a <u>set</u> of states

Non-deterministic Finite Automata (NFA)

- A Non-deterministic Finite Automaton (NFA) consists of:
  - Q ==> a finite set of states
  - ∑ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of final states
  - δ ==> a transition function, which is a mapping between Q x ∑ ==> subset of Q
- An NFA is also defined by the 5-tuple:
  - {Q, ∑ , q_0,F, δ }

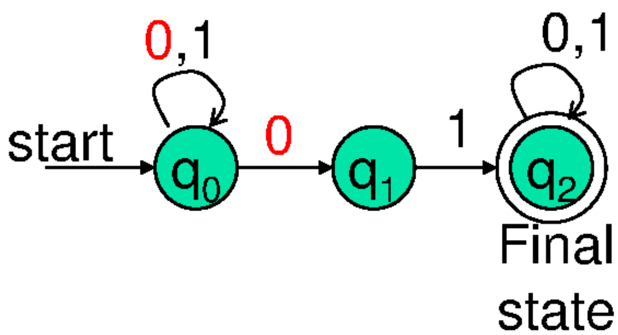
How to use an NFA?

- <u>Input:</u> a word w in $\sum^*$
- <u>Question:</u> Is w acceptable by the NFA?
- <u>Steps:</u>
  - Start at the “start state” q_0
  - For every input symbol in the sequence w do
    - Determine all possible next states from all current states, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed <u>and</u> if at least one of the current states is a final state then *accept w;*
  - Otherwise, *reject w.*

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

NFA for strings containing 01

Why is this non-deterministic?



What will happen if at state q_1 an input of 0 is received?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

symbols

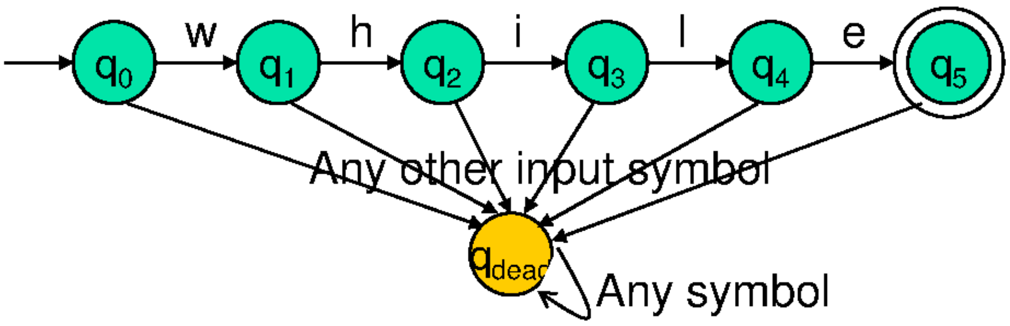
δ	**0**	**1**
→q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
***q_2**	$\{q_2\}$	$\{q_2\}$

states

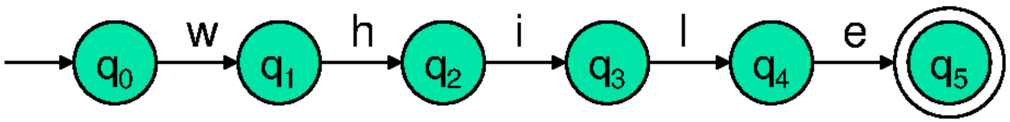
Note: Explicitly specifying dead states is just a matter of design convenience
(one that is generally followed in NFAs), and
this feature does not make a machine deterministic or non-deterministic.

What is a "dead state"?

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



- An NFA for the same purpose:



Transitions into a dead state are implicit

Example #2

- Build an NFA for the following language:
  L = { w | w ends in 01}
- ?
- Other examples
  - Keyword recognizer (e.g., if, then, else, while, for, include, etc.)
  - Strings where the first symbol is present somewhere later on at least once

Extension of δ to NFA Paths

- Basis: $\hat{\delta}(q,\varepsilon) = \{q\}$

- Induction:
  - Let $\hat{\delta}(q_0,w) = \{p_1,p_2\ldots,p_k\}$
  - $\delta(p_i,a) = S_i$ for $i=1,2\ldots,k$
  - Then, $\hat{\delta}(q_0,wa) = S_1 \cup S_2 \cup \ldots \cup S_k$

Language of an NFA

- An NFA accepts *w* if *there exists at least one* path from the start state to an accepting (or final) state that is labeled by *w*
- $L(N) = \{ w \mid \hat{\delta}(q_0,w) \cap F \neq \Phi \}$

Advantages & Caveats for NFA

- Great for modeling regular expressions
  - String processing - e.g., grep, lexical analyzer

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

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

Differences: DFA vs. NFA

- **DFA**

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

- **NFA**

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