Introduction to Automata Theory

Reading: Chapter 1

What is Automata Theory?

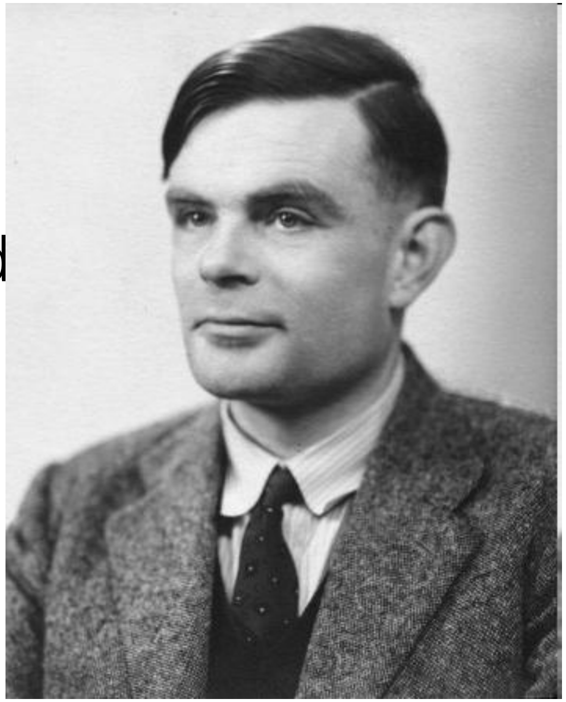
- *Study of abstract computing devices, or "machines"*
- Automaton = an abstract computing device
  - Note: A "device" need not even be a physical hardware!
- A fundamental question in computer science:
  - Find out what different models of machines can do and cannot do
  - The *theory of computation*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

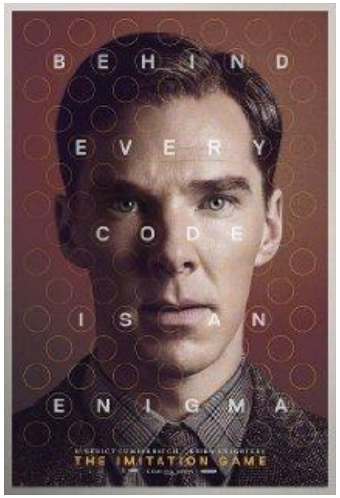
(A pioneer of automata theory)

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?





Theory of Computation: A Historical Perspective

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

Languages & Grammars

Or "**words**"

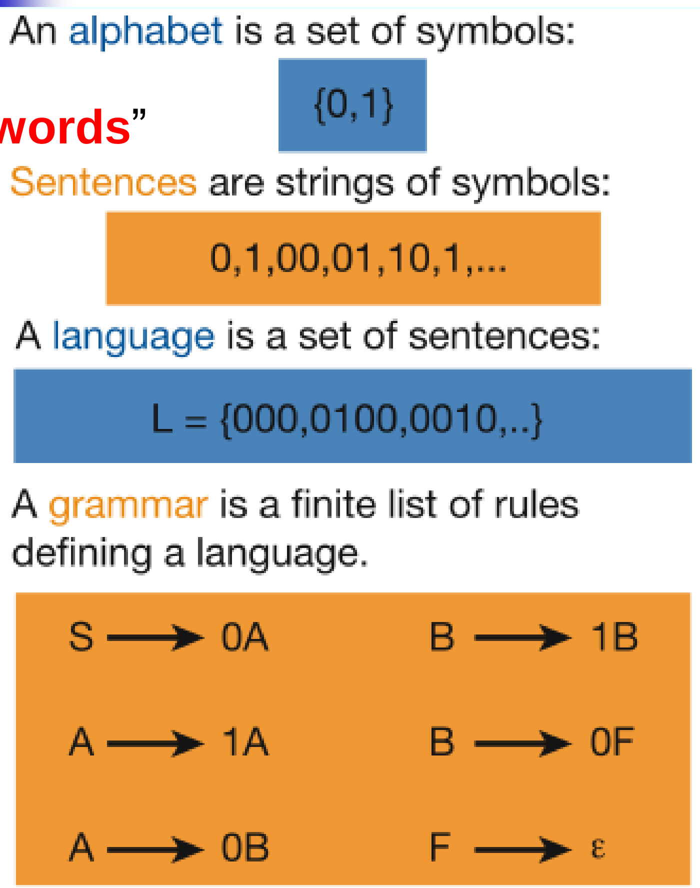
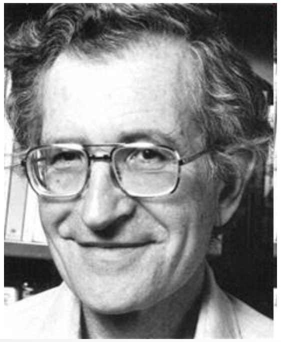


Image source: Nowak et al. Nature, vol 417, 2002

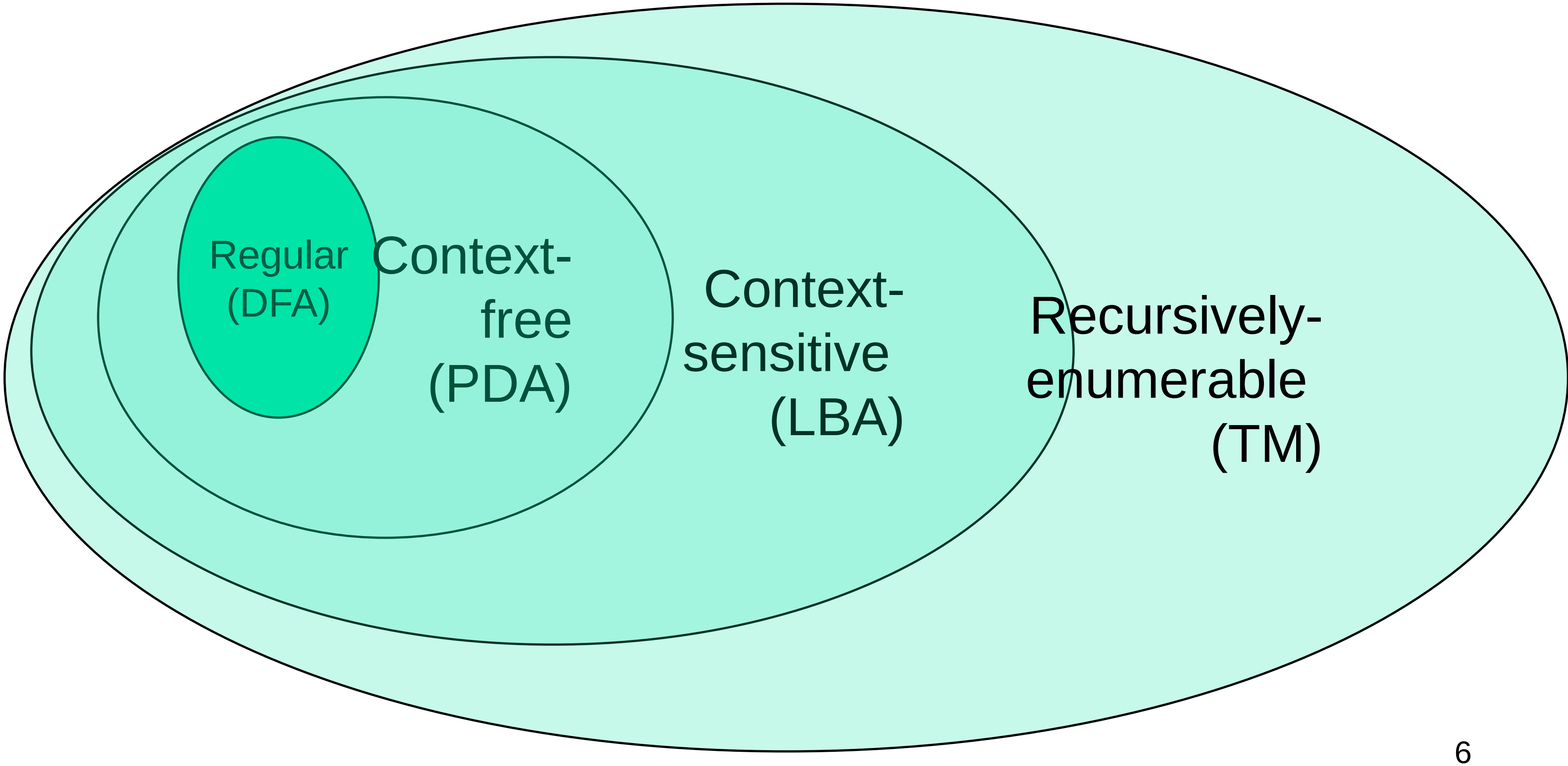
- <u>Languages</u>: "*A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols*"
- <u>Grammars</u>: "*A grammar can be regarded as a device that enumerates the sentences of a language*" - nothing more, nothing less

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

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
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

Languages

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

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

<u>Examples:</u>

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

   L = {ε, 01, 0011, 000111,…}

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

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

   Canonical ordering of strings in the language

Definition: Ø denotes the Empty language

- Let L = {ε}; Is L=Ø? $\boxed{\text{NO}}$

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

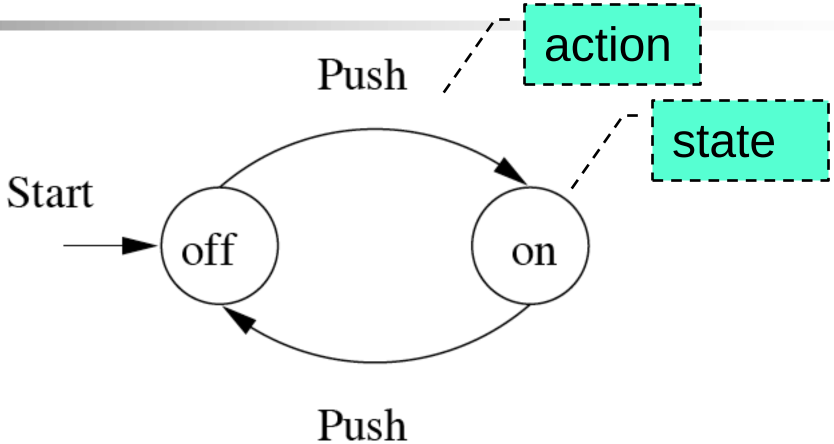
Q) Is w $\in$ the language of strings with equal number of 0s and 1s?

Finite Automata

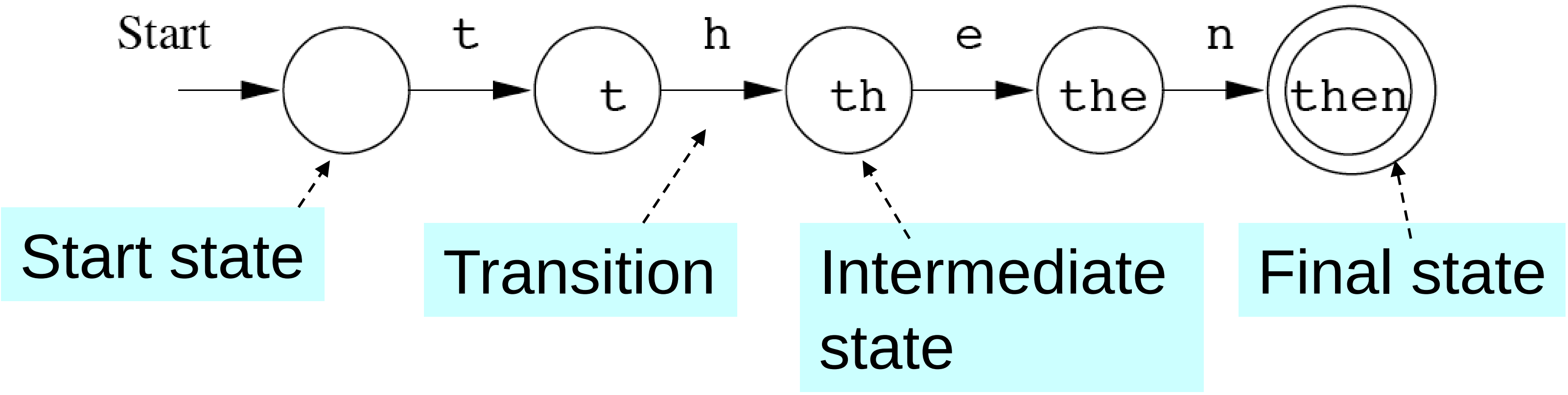
- Some Applications
  - Software for designing and checking the behavior of digital circuits
  - Lexical analyzer of a typical compiler
  - Software for scanning large bodies of text (e.g., web pages) for pattern finding
  - Software for verifying systems of all types that have a finite number of states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

Formal Proofs

Deductive Proofs

From the given statement(s) to a conclusion statement (what we want to prove)

- Logical progression by direct implications

Example for parsing a statement:

- "If $y \geq 4$, then $2^y \geq y^2$."

$\boxed{\textit{given}}$ $\boxed{\textit{conclusion}}$

(there are other ways of writing this).

Example: Deductive proof

Let <u>Claim 1:</u> If $y \geq 4$, then $2^y \geq y^2$.

Let x be any number which is obtained by adding the squares of 4 positive integers.

<u>Claim 2:</u>

Given x and assuming that Claim 1 is true, prove that $2^x \geq x^2$

- <u>Proof:</u>
  1) Given: $x = a^2 + b^2 + c^2 + d^2$
  2) Given: $a \geq 1$, $b \geq 1$, $c \geq 1$, $d \geq 1$
  3) ➔ $a^2 \geq 1$, $b^2 \geq 1$, $c^2 \geq 1$, $d^2 \geq 1$ (by 2)
  4) ➔ $x \geq 4$ (by 1 & 3)
  5) ➔ $2^x \geq x^2$ (by 4 and Claim 1)

"implies" or *"follows"*

On Theorems, Lemmas and Corollaries

We typically refer to:

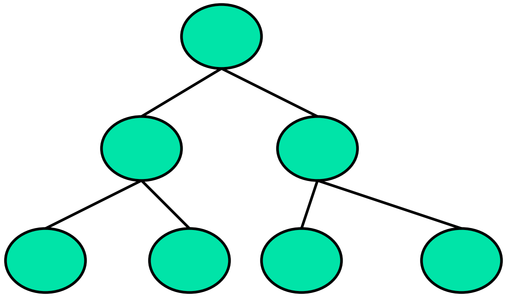
- A major result as a "**theorem**"
- An intermediate result that we show to prove a larger result as a "**lemma**"
- A result that follows from an already proven result as a "**corollary**"

An example:

Theorem: *The height of an n-node binary tree is at least floor(lg n)*

Lemma: *Level i of a perfect binary tree has 2^i nodes.*

Corollary: *A perfect binary tree of height h has 2^{h+1}-1 nodes.*



Quantifiers

- *“For all”* or “*For every”*
  - Universal proofs
  - Notation= $\forall$
- *“There exists”*
  - Used in existential proofs
  - Notation= $\exists$
- Implication is denoted by =>
  - E.g., “IF A THEN B” can also be written as “A=>B”

Proving techniques

- By contradiction
  - Start with the statement contradictory to the given statement
  - E.g., To prove (A => B), we start with:
    - (A and ~B)
    - … and then show that could never happen

    What if you want to prove that “(A and B => C or D)”?

- By induction
  - (3 steps) Basis, inductive hypothesis, inductive step
- By contrapositive statement
  - If *A* then *B* ≡ If ~*B* then ~*A*

Proving techniques…

- By counter-example
  - Show an example that disproves the claim

- Note: There is no such thing called a "proof by example"!
  - So when asked to prove a claim, an example that satisfied that claim is *not* a proof

Different ways of saying the same thing

- "*If* H *then* C":
  i. H *implies* C
  ii. *H => C*
  iii. C *if* H
  iv. H *only if* C
  v. *Whenever* H *holds*, C *follows*

“If-and-Only-If” statements

- “A if and only if B” (A <==> B)
  - *(if part)* if B then A (<=)
  - *(only if part)* A only if B (=>)
    (same as “if A then B”)
- “If and only if” is abbreviated as “iff”
  - i.e., “A iff B”
- <u>Example:</u>
  - <u>Theorem:</u> *Let x be a real number. Then floor of x = ceiling of x <u>if and only if</u> x is an integer.*
- Proofs for iff have two parts
  - One for the “if part” & another for the “only if part”

NFA to DFA conversion and regular expressions

DFAs and NFAs are equally powerful

NFA can do everything a DFA can do

How about the other way?

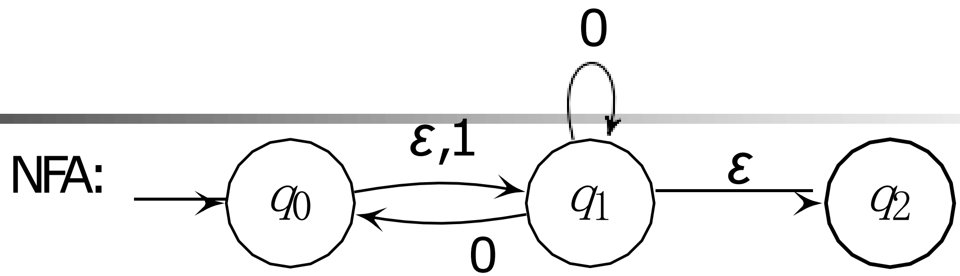
Every NFA is equivalent to some DFA for the same language

NFA → DFA algorithm

Given an NFA, figure out

1. the initial active states
2. how the set of active states changes upon reading an input symbol

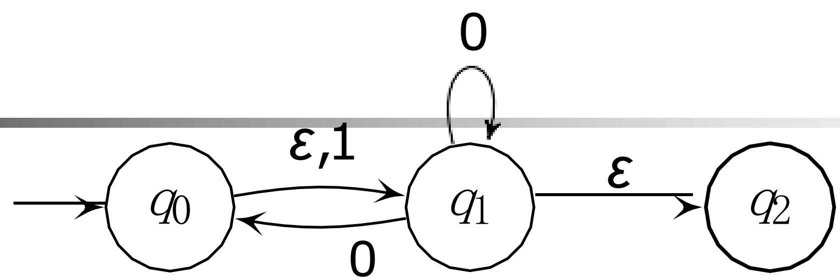
NFA → DFA example



Initial active states (before reading any input)?

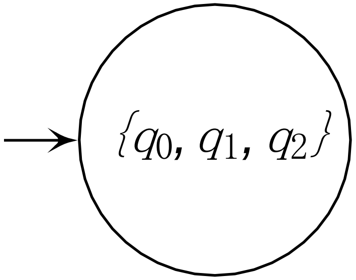
NFA → DFA example

NFA:



Initial active states (before reading any input)?

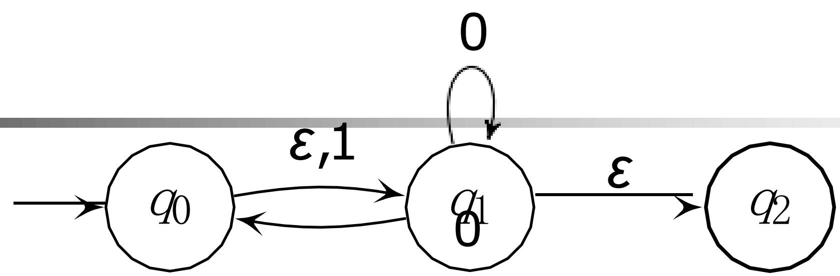
partia
l
DFA:



How does the set of active states change?

NFA → DFA example

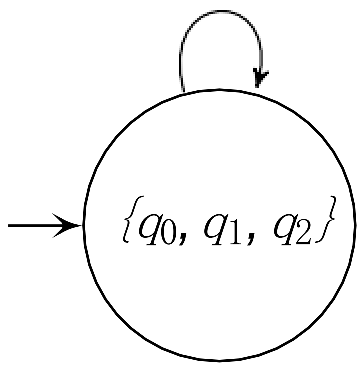
NFA:



Initial active states (before reading any input)?

0

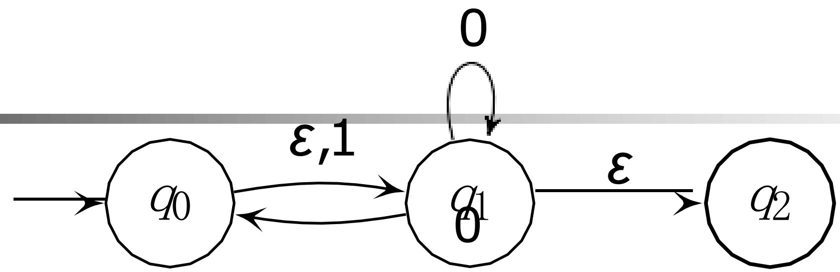
partia
l
DFA:



How does the set of active states change?

NFA → DFA example

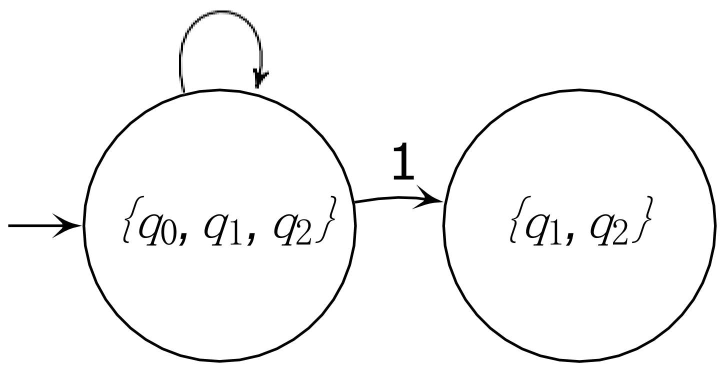
NFA:



Initial active states (before reading any input)?

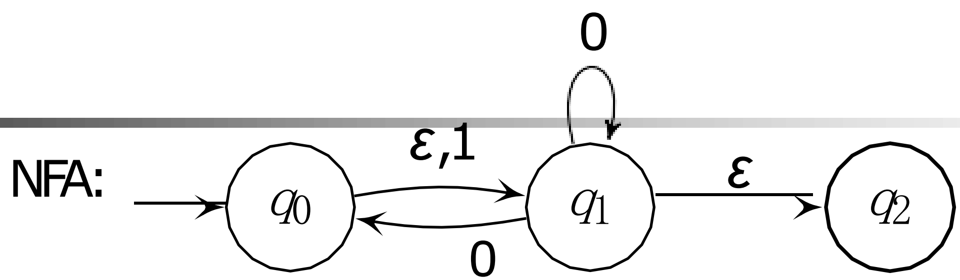
0

partia
l
DFA:

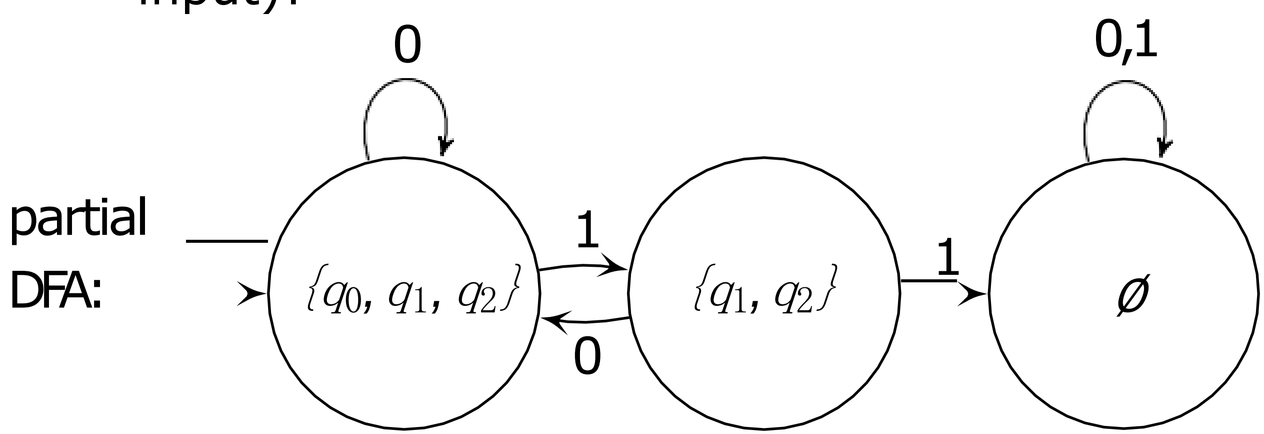


How does the set of active states change?

NFA → DFA example

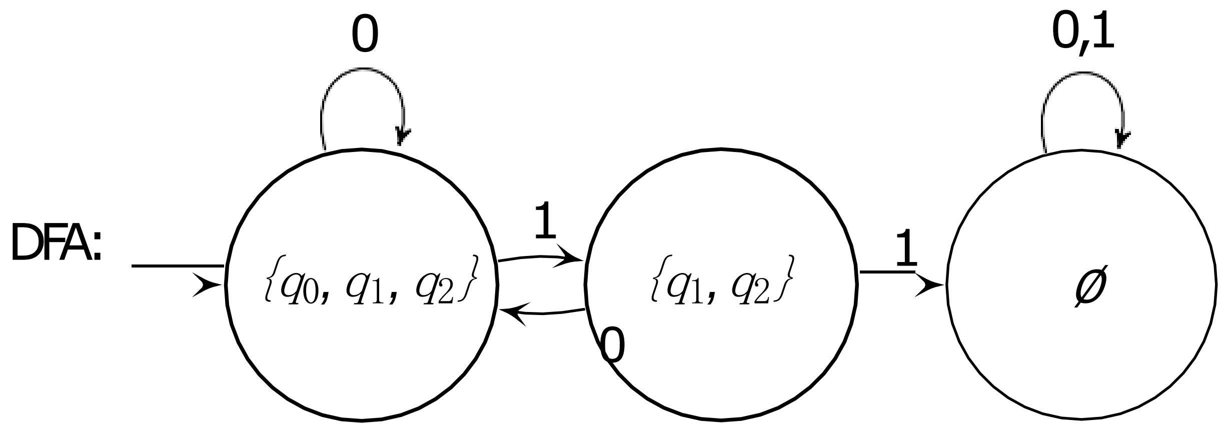


Initial active states (before reading any input)?



How does the set of active states change?

NFA → DFA summary



Every DFA state corresponds to a subset of NFA states

A DFA state is accepting if it contains an accepting NFA state

Regular expressions

S

Powerful string matching feature in advanced editors (e.g. Vim, Emacs) and modern programming languages (e.g. PERL, Python)

PERL regex examples:

`colou?r` matches "color"/"colour"

`[A-Za-z]*ing` matches any word ending in "ing"

We will learn to parse complicated regex recursively by building up from simpler ones

Also construct the language matched by the expression recursively

Will focus on regular expressions in formal language theory (notations differ from PERL/Python/POSIX regex) on

s = abb

t = bab

st = abbbab

ts = bababb

ss = abbabb

sst = abbabbbab

$$s = x_1 \ldots x_n,\quad t = y_1 \ldots y_m$$

$$\Downarrow$$

$$st = x_1 \ldots x_n y_1 \ldots y_m$$

on languages

- Concantenation of languages L_1 and L_2

$$L_1L_2 = \{st : s \in L_1, t \in L_2\}$$

- n-th power of language L

$$L^n = \{s_1 s_2 \dots s_n \,/\, s_1, s_2, \dots, s_n \in L\}$$

- Union of L_1 and L_2

$$L_1 \cup L_2 = \{s \,/\, s \in L_1 \text{ or } s \in L_2\}$$

e

$L_1 = \{0, 01\}$ $L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$

$L_1L_2 = \{0, 01, 011, 0111, \ldots\} \cup \{01, 011, 0111, 01111, \ldots\}$

$= \{0, 01, 011, 0111, \ldots\}$

0 followed by any number of 1s

$L_1^2 = \{00, 001, 010, 0101\}$

$L_2^2 = L_2$

$L_2^n = L_2$ for any n “ 1

$L_1 \cup L_2 = \{0, 01, \varepsilon, 1, 11, 111, \ldots\}$

Operations on languages

The star of L are contains strings made up of zero or more chunks from L

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$$

Example: $L_1 = \{0, 01\}$ and $L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$

What is $L_1{}^*$? $L_2{}^*$?

e

$L_1 = \{0, 01\}$

$L_1^0 = \{\varepsilon\}$

$L_1^1 = \{0, 01\}$

$L_1^2 = \{00, 001, 010, 0101\}$

$L_1^3 = \{000, 0001, 0010, 00101, 0100, 01001, 01010, 010101\}$

Which of the following are in $L_1{}^*$?

00100001

00110001

10010001

e

$L_1 = \{0, 01\}$

$L_1^0 = \{\varepsilon\}$

$L_1^1 = \{0, 01\}$

$L_1^2 = \{00, 001, 010, 0101\}$

$L_1^3 = \{000, 0001, 0010, 00101, 0100, 01001, 01010, 010101\}$

Which of the following are in $L_1{}^*$?

00100001	00110001	10010001
Yes	No	No

e

$L_1 = \{0, 01\}$

$L_1^0 = \{\varepsilon\}$

$L_1^1 = \{0, 01\}$

$L_1^2 = \{00, 001, 010, 0101\}$

$L_1^3 = \{000, 0001, 0010, 00101, 0100, 01001, 01010, 010101\}$

Which of the following are in L_1^*?

00100001	00110001	10010001
Yes	No	No

L_1^* contains all strings such that any 1 is preceded by a 0

Example

$$L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$$

any number of 1s

$L_2^0 = \{\varepsilon\}$

$L_2^1 = L_2$

$L_2^2 = L_2$

$L_2^n = L_2 \quad (n$ “
1)

e

$$L_2 = \{\varepsilon, 1, 11, 111, \dots\}$$

any number of 1s

$L_2^0 = \{\varepsilon\}$

$L_2^1 = L_2$

$L_2^2 = L_2$

$L_2^n = L_2 \quad (n$ “ $1)$

$$L_2^* = L_2^0 \cup L_2^1 \cup L_2^2 \cup \dots$$
$$= \{\varepsilon\} \cup L_2 \cup L_2 \cup \dots$$
$$= L_2$$

$$L_2^* = L_2$$

languages

We can construct languages by starting with simple ones, like $\{0\}$ and $\{1\}$, and combining them

$\{0\}(\{0\} \cup \{1\})^*$ $\Rightarrow$ $0(0+1)^*$ all strings that start with 0

languages

We can construct languages by starting with simple ones, like $\{0\}$ and $\{1\}$, and combining them

$\{0\}(\{0\} \cup \{1\})^*$ $\Rightarrow$ $0(0+1)^*$
all strings that start with 0

$(\{0\}\{1\}^*) \cup (\{1\}\{0\}^*)$ $\Rightarrow$ $01^* + 10^*$
0 followed by any number of 1s, or
1 followed by any number of 0s

languages

We can construct languages by starting with simple ones, like $\{0\}$ and $\{1\}$, and combining them

$\{0\}(\{0\} \cup \{1\})^*$ $\Rightarrow$ $0(0+1)^*$
all strings that start with 0

$(\{0\}\{1\}^*) \cup (\{1\}\{0\}^*)$ $\Rightarrow$ $01^* + 10^*$
0 followed by any number of 1s, or
1 followed by any number of 0s

$0(0+1)^*$ and $01^* + 10^*$ are regular expressions Blueprints for combining simpler languages into complex ones

expressions

A regular expression over Σ is an expression formed by the following rules

- The symbols $\emptyset$ and ε are regular expressions
- Every symbol a in Σ is a regular expression
- If R asd S are regular expressions, so are $R + S$, RS and R^*

Examples:

$\emptyset$

$0(0 + 1)^*$

$01^* + 10^*$

ε

$1^*(\varepsilon + 0)$

$(0 + 1)^*01(0 + 1)^*$

A language is regular if it is represented by a regular expression

regular expressions

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

$01^* = 0(1)^*$ represents $\{0, 01, 011, 0111, \ldots\}$ 0 followed by any number of 1s

01^* is not $(01)^*$

regular expressions

$0 + 1$ yields $\{0, 1\}$ — strings of length 1

$(0 + 1)^*$ yields $\{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$ — any string

$(0 + 1)^*010$ — any string that ends in 010

$(0 + 1)^*01(0 + 1)^*$ — any string containing 01

Understanding regular expressions

What language does the following represent?

$$((0 + 1)(0 + 1))^* + ((0 + 1)(0 + 1)(0 + 1))^*$$

regular expressions

What language does the following represent?

$((0+1)(0+1))^* + ((0+1)(0+1)(0+1))^*$

$((0+1)(0+1))^*$

$((0+1)(0+1)(0+1))^*$

regular expressions

What language does the following represent?

$((0+1)(0+1))^* + ((0+1)(0+1)(0+1))^*$

$((0+1)(0+1))^*$

$((0+1)(0+1)(0+1))^*$

$(0+1)(0+1)$

$(0+1)(0+1)(0+1)$

regular expressions

What language does the following represent?

$((0+1)(0+1))^* + ((0+1)(0+1)(0+1))^*$

$((0+1)(0+1))^*$

$((0+1)(0+1)(0+1))^*$

$(0+1)(0+1)$

strings of length 2

$(0+1)(0+1)(0+1)$

strings of length 3

regular expressions

What language does the following represent?

$((0+1)(0+1))^* + ((0+1)(0+1)(0+1))^*$

$((0+1)(0+1))^*$

strings of even length

$(0+1)(0+1)$

strings of length 2

$((0+1)(0+1)(0+1))^*$

strings whose length is a multiple of 3

$(0+1)(0+1)(0+1)$

strings of length 3

regular expressions

What language does the following represent?

$((0+1)(0+1))^* + ((0+1)(0+1)(0+1))^*$

strings whose length is even or a multiple of 3

$=$ strings of length $0, 2, 3, 4, 6, 8, 9, 10, 12, \ldots$

$((0+1)(0+1))^*$

strings of even length

$(0+1)(0+1)$

strings of length 2

$((0+1)(0+1)(0+1))^*$

strings whose length is a multiple of 3

$(0+1)(0+1)(0+1)$

strings of length 3

Understanding regular expressions

What language does the following represent?

$((0+1)(0+1)+(0+1)(0+1)(0+1))^*$

Understanding regular expressions

What language does the following represent?

$$((0+1)(0+1)+(0+1)(0+1)(0+1))^*$$

$$(0+1)(0+1)+(0+1)(0+1)(0+1)$$

regular expressions

What language does the following represent?

$((0+1)(0+1)+(0+1)(0+1)(0+1))^*$

$(0+1)(0+1)+(0+1)(0+1)(0+1)$

$(0+1)(0+1)$

$(0+1)(0+1)(0+1)$

regular expressions

What language does the following represent?

$((0+1)(0+1) + (0+1)(0+1)(0+1))^*$

$(0+1)(0+1) + (0+1)(0+1)(0+1)$

$(0+1)(0+1)$

strings of length 2

$(0+1)(0+1)(0+1)$

strings of length 3

regular expressions

What language does the following represent?

$$((0+1)(0+1)+(0+1)(0+1)(0+1))^*$$

$$(0+1)(0+1)+(0+1)(0+1)(0+1)$$

$(0+1)(0+1)$

strings of length 2

strings of length 2 or 3

$(0+1)(0+1)(0+1)$

strings of length 3

regular expressions

What language does the following represent?

$$((0+1)(0+1) + (0+1)(0+1)(0+1))^*$$

strings that can be broken into blocks, where each block has length 2 or 3

$$(0+1)(0+1) + (0+1)(0+1)(0+1)$$

strings of length 2 or 3

$$(0+1)(0+1)$$

strings of length 2

$$(0+1)(0+1)(0+1)$$

strings of length 3

regular expressions

What language does the following represent?

$((0+1)(0+1) + (0+1)(0+1)(0+1))^*$

strings that can be broken into blocks, where each block has length 2 or 3

Which are in the language?

ε 1 01 011 00110 011010110

regular expressions

What language does the following represent?

$((0+1)(0+1) + (0+1)(0+1)(0+1))^*$

strings that can be broken into blocks, where each block has length 2 or 3

Which are in the language?

ε	1	01	011	00110	011010110
✓	(	✓	✓	✓	✓

The regular expression represents all strings except 0 and 1

Understanding regular expressions

What language does the following represent?

$$(1 + 01 + 001)^* (\varepsilon + 0 + 00)$$

Understanding regular expressions

What language does the following represent?

ends in at most two 0s

$$(1 + 01 + 001)^* (\varepsilon + 0 + 00)$$

regular expressions

What language does the following represent?

ends in at most two 0s

$$(1 + 01 + 001)^*(\varepsilon + 0 + 00)$$

at most two 0s between two consecutive 1s

Never three consecutive 0s

The regular expression represents strings not containing 000

Examples:

ε 00 0110010110 0010010

expressions

Write a regular expression for all strings with two consecutive 0s

Writing regular expressions

Write a regular expression for all strings with two consecutive 0s

(anything)00(anything)

$$(0+1)^*00(0+1)^*$$