Course 4
Finite Automata/Finite State Machines

The structure and the content of the lecture is based on (1) http://www.eecs.wsu.edu/~ananth/CptS317/Lectures/index.htm, (2) W. Schreiner Computability and Complexity, Lecture Notes, RISC-JKU, Austria

Excursion: Previous lecture

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
  - ∑ - a finite set of input symbols (alphabet)
  - q_0 - a start state (one of the elements from Q)
  - F - set of accepting states
  - $\delta : Q \times \Sigma \rightarrow Q$ – a transition function which takes a state and an input symbol as an argument and returns a state.
- A DFA is defined by the 5-tuple: $\{Q, \sum, q_0, F, \delta\}$

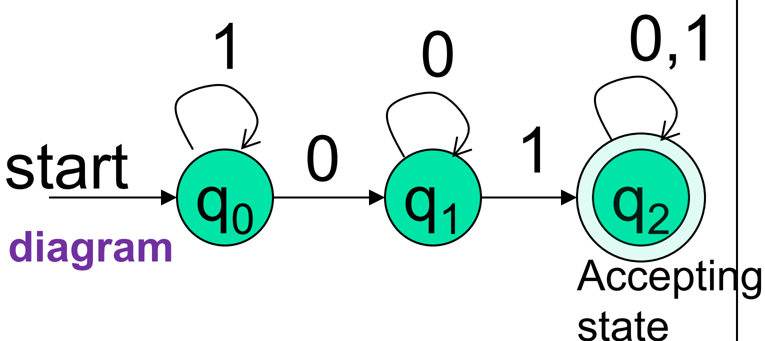
Example #1

- Build a DFA for the following language:
  - L = {w | w is a binary string that contains 01 as a substring} same as
  - L = {w | w is of the form x01y where x,y are binary strings} same as
  - L = {x01y | x,y are binary strings}
  - *Examples*: 01, 010, 011, 0011, etc.
  - *Counterexamples*: ε, 0, 1, 111000
- Steps for building a DFA to recognize L:
  - $\sum$ = {0,1}
  - Decide on the non-final (non-accepting) states: Q
  - Designate start state and final (accepting) state(s): F
  - Decide on the transitions: δ

Regular expression: (01)*01(01)*

DFA for strings containing 01

- What makes this DFA deterministic?



Transition diagram

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

Transition table

symbols

δ	**0**	**1**
→ **q_0** (Start state)	q_1	q_0
q_1	q_1	q_2
***q_2** (Accepting/final state)	q_2	q_2

states

What if the language allows empty strings?

Finite Automata (cont'd)

Example #2

- Build a DFA for the following language:
  - L = { w | w is a binary string that has exactly length 2}

See whiteboard

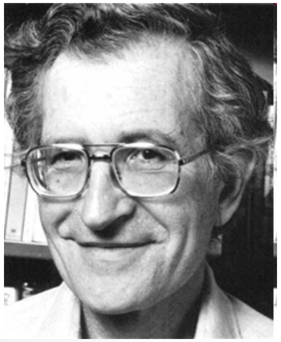
What does a DFA do on reading an input string?

- 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 accepting states (F) then *accept w;*
  - Otherwise, *reject w.*

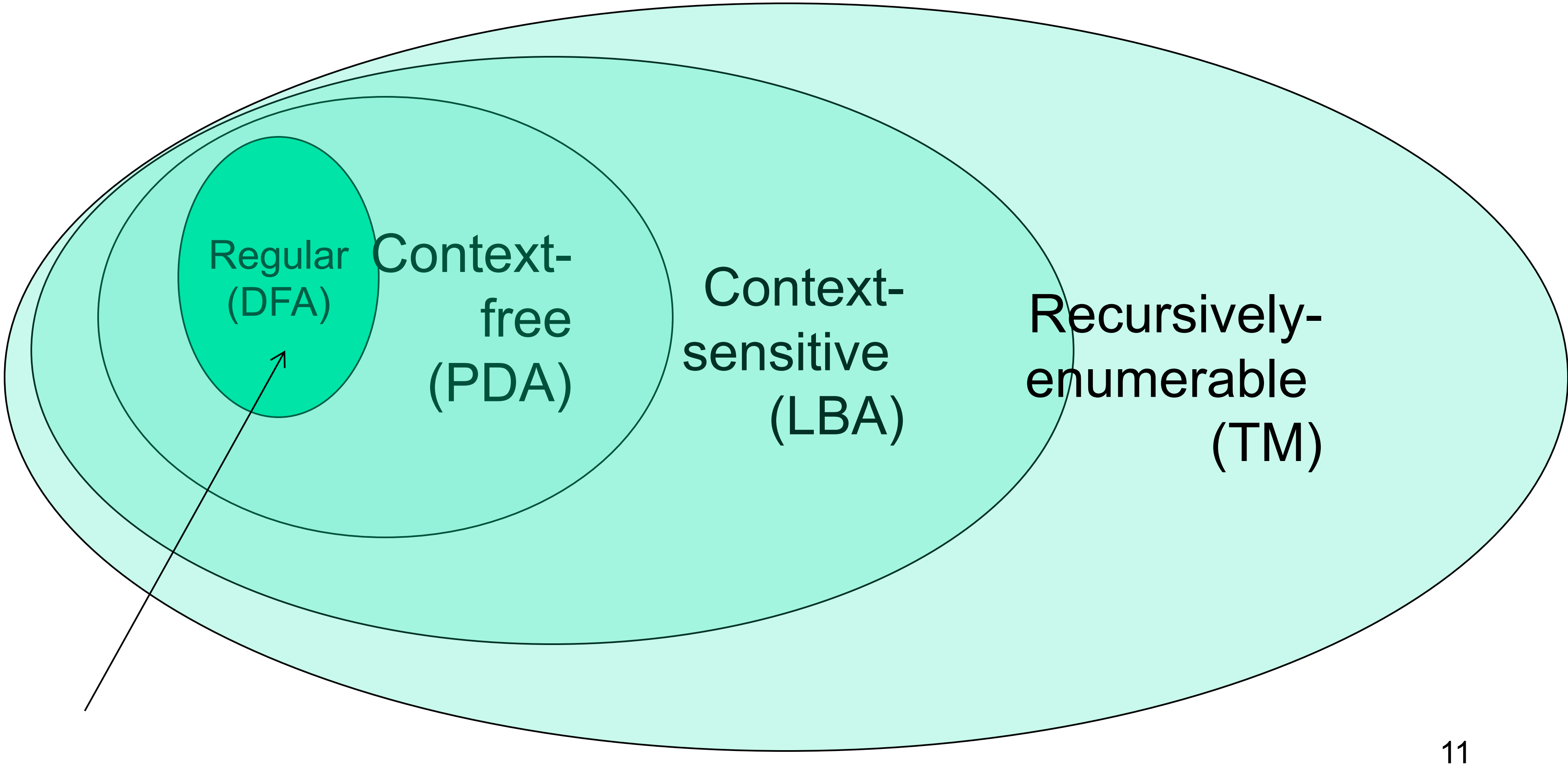
Regular Languages

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

The Chomsky Hierachy



Location regular languages in the Chomsky Hierarchy



Example #3: Clamping Logic

- *Problem*: A clamping circuit (https://en.wikipedia.org/wiki/Clamper_(electronics)) 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.
- *Solution*: 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 #4: Even Number of Digits

- Consider the program which reads symbols from an input stream and returns true if the stream contains an even number of ‘0’ and an even number of ‘1’ (and no other symbol).

```
function EvenZerosAndOnes()
    e0, e1 ← true, true
    while input stream is not empty do
        read input
        case input of
            0: e0 ← ¬e0
            1: e1 ← ¬e1
            default: return false
        end case
    end while
    return e0 ∧ e1
end function
```

	e_0	e_1
q_0	true	true
q_1	true	false
q_2	false	true
q_2	false	false

Example #3: Even Number of Digits (cont'd)

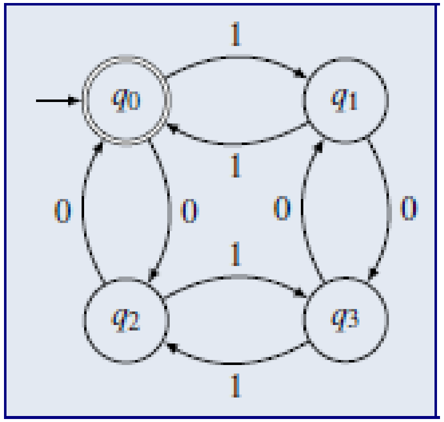
$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$

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

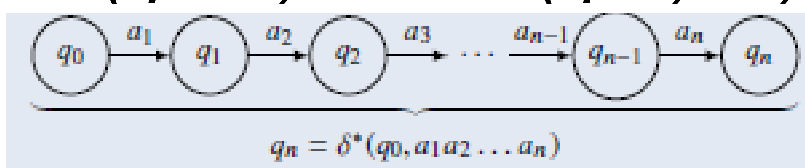
$F = \{q_0\}$

δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2



Extension of transitions (δ) to Paths (δ*)

- $\delta^*(q,w)$ = *destination state* from state *q* on input string *w*; *used for determining the language of a DFA (see next slide)*
- *Base case:* $\delta^*(q,\varepsilon) = q$
- *Induction:* $\delta^*(q,wa) = \delta(\delta^*(q,w), a)$



- Example: Work out example #3 (Clamping Logic) using the input sequence
  - *w=10:* $\delta^*(q_0,w) = ?$
  - w=1110: $\delta^*(q_0,w) = ?$

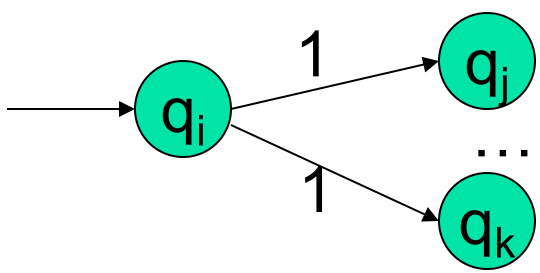
Language of a DFA

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., $L(A) = \{ w \mid \delta^*(q_0,w) \in F \}$
- *i.e., L(A) = all strings that lead to an accepting 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 set 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 accepting states
  - δ: a transition function, which is a mapping between Q x ∑ and a subset of Q
- An NFA is also defined by the 5-tuple:
  - $\{Q, \Sigma, q_0, F, \delta\}$

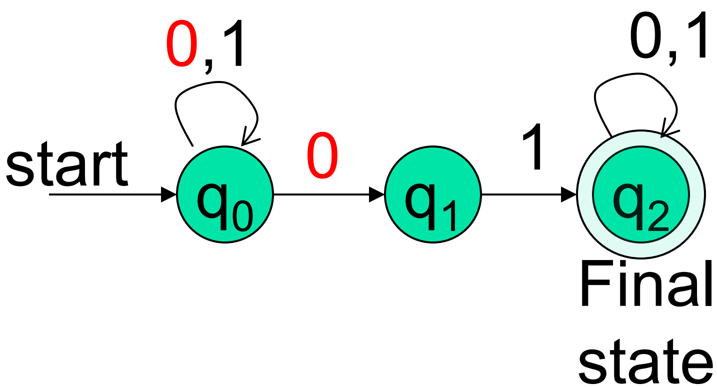
How to use an NFA?

- Input: a word w in Σ^*
- Question: Is w acceptable by the NFA?
- Steps:
  - 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 and 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

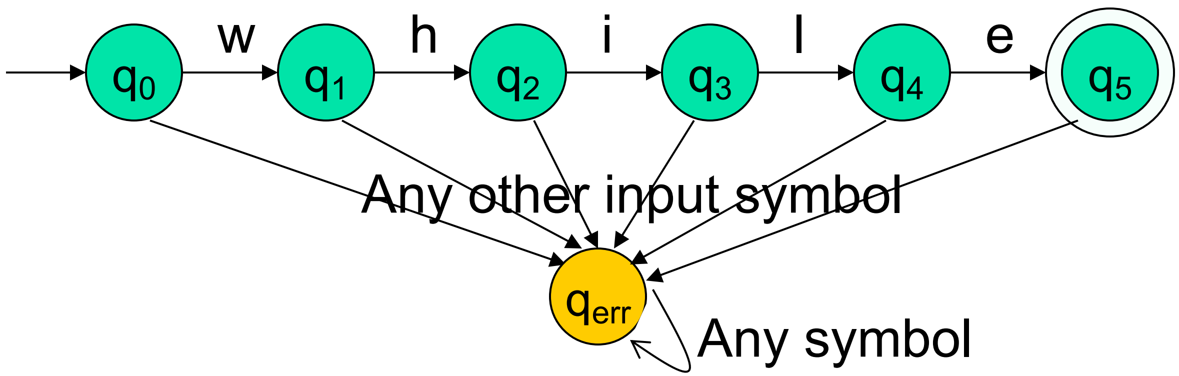
δ	**0**	**1**
→ **q_0**	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
***q_2**	$\{q_2\}$	$\{q_2\}$

states

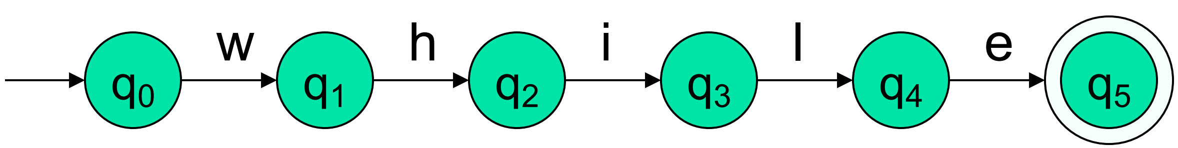
Note: Omitting to explicitly show error states is just a matter of design convenience (one that is generally followed for NFAs), and i.e., this feature should not be confused with the notion of non-determinism.

What is an "error state"?

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



- An NFA for the same purpose:



Transitions into a dead state are implicit

Further examples

- NFA for the clamping logic example (see whiteboard).
- Build an NFA for the following language:

  L = { w | w ends in 01}

  Compare them with the corresponding DFA (see whiteboard).
- Other examples
  - Keyword recognizer (e.g., if, then, else, while, for, include, etc.) @seminar
  - Strings where the first symbol is present somewhere later on at least once @seminar

Extension of δ to NFA Paths

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

- Induction:
  - Let $\delta^* (q_0, w) = \{p_1, p_2 \ldots, p_k\}$
  - $\delta (p_i, a) = S_i$ *for* $i=1,2\ldots,k$

  - *Then,* $\delta^* (q_0, wa) = S_1 \cup S_2 \cup \ldots \cup S_k$

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*
- $L(N) = \{ w \mid \delta^*(q_0,w) \cap F \neq \Phi \}$

Advantages & Caveats for NFA

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

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

Summary

- **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 visited 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 an error 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 implementations limited but emerging (e.g., Micron automata processor)