Formal Languages, Automata and Computation

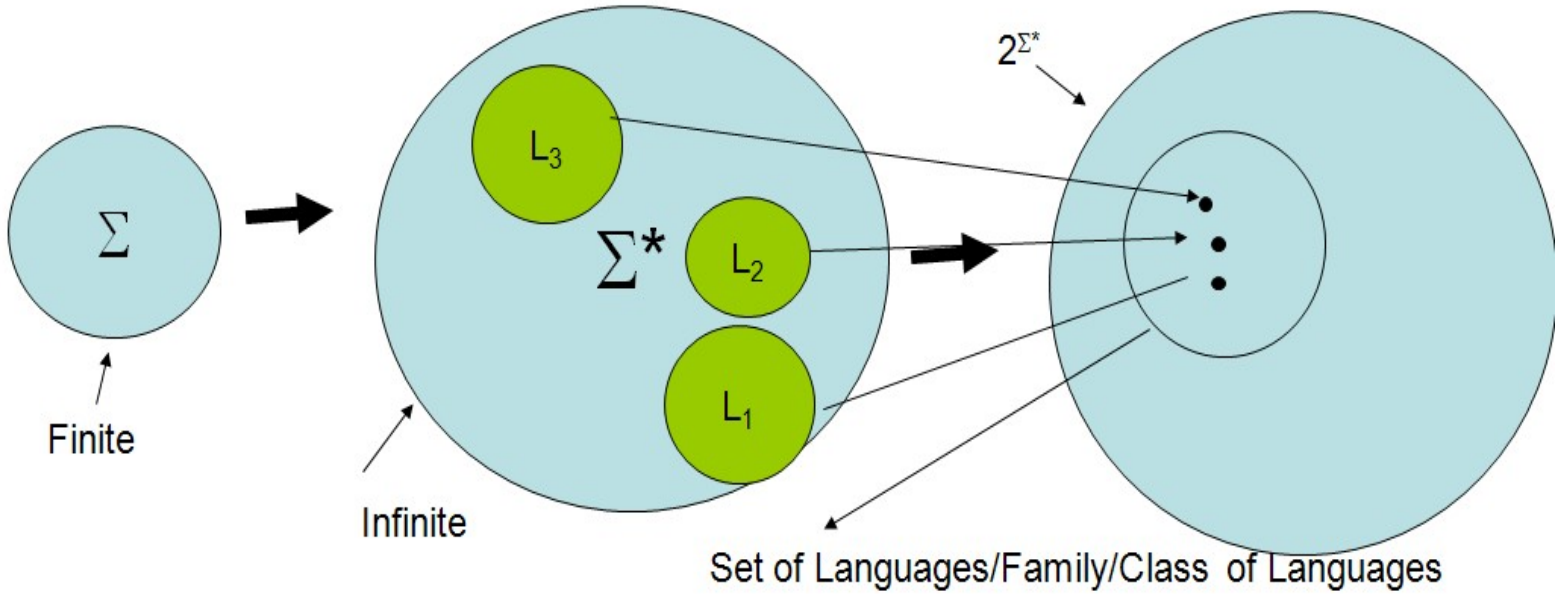
Finite State Machines

Carnegie Mellon University in Qatar

January 11, 2011

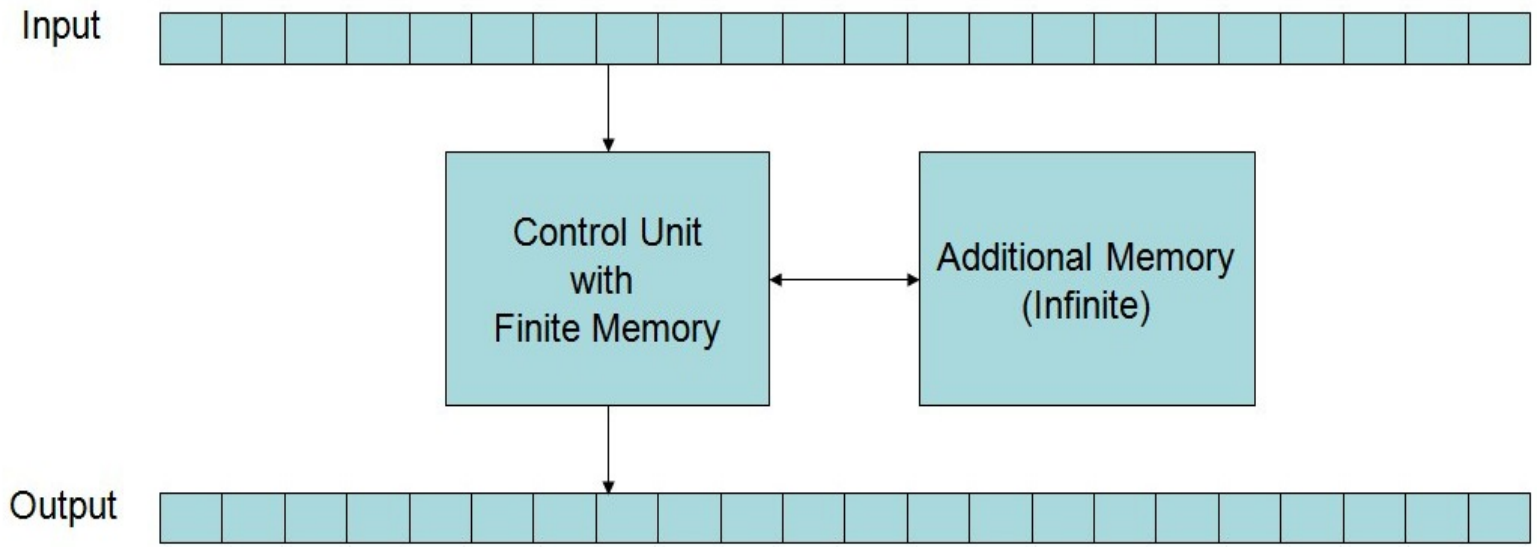
SUMMARY

- Alphabet Σ,
- Set of all Strings, Σ^*,
- Language $L \subseteq \Sigma^*$,
- Set of all languages 2^{Σ^*}



AUTOMATA

- Abstract Models of computing devices



- Each step of operation is like:
  - If the current input symbol is X then output Y, move (left/right)

AUTOMATA

- The control unit has some finite memory and it keeps track of what step to execute next.
- Additional memory (if any) is infinite - we never run out of memory!
  - Infinite but like a stack - only the top item is accessible at a given time.
  - Infinite but like a tape, any cell is (sequentially) accessible.

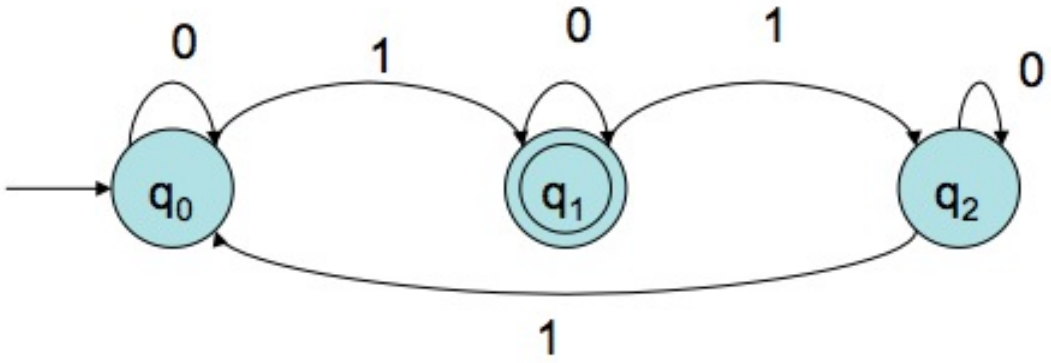
Finite State Automata

- Finite State Automata (FSA) are the simplest automata.
- Only the finite memory in the control unit is available.
- The memory can be in one of finite states at a given time – hence the name.
  - One can remember only a (fixed) finite number of properties of the past input.
  - Since input strings can be of arbitrary length, it is not possible to remember unbounded portions of the input string.
- It comes in Deterministic and Nondeterministic flavors.

Deterministic Finite State Automata (DFA)

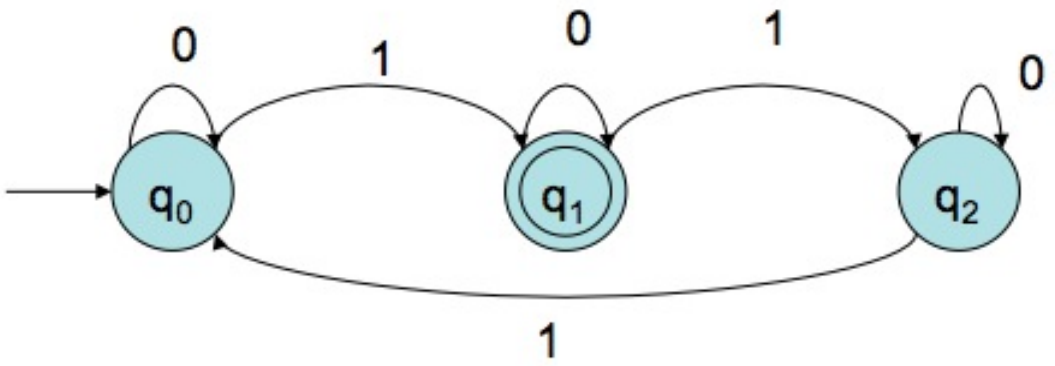
- A DFA starts in a start state and is presented with an input string.
- It moves from state to state, reading the input string one symbol at a time.
- What state the DFA moves next depends on
  - the current state,
  - current input symbol
- When the last input symbol is read, the DFA decides whether it should accept the input string

A SIMPLE DFA EXAMPLE



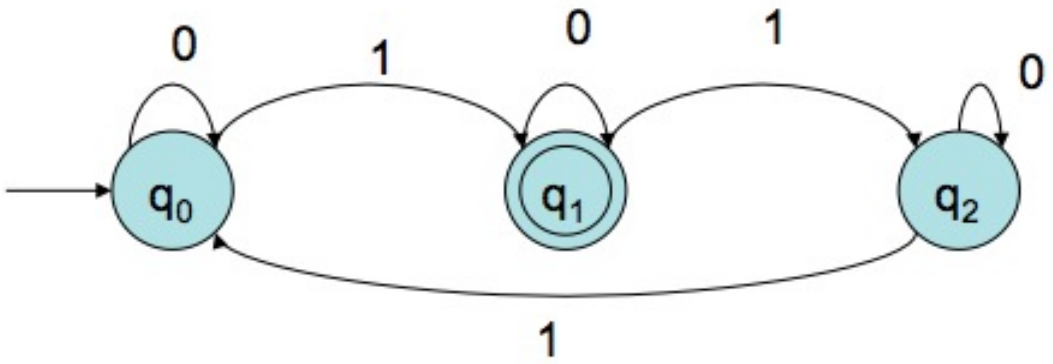
- States are shown with circles. We usually have labels on the states.
  - One designated state is the start state, (State q_0 here).
  - States with double circles denote the accepting or final states (State q_1 here)
- Directed and labeled arrows between states denote state transitions.

A SIMPLE DFA EXAMPLE



- This DFA stays in the same state when the next input symbol is a 0.
- In state q_0, an input of 1 moves the DFA to state q_1.
- In state q_1, an input of 1 moves the DFA to state q_2.
- In state q_2, an input of 1 moves the DFA back to state q_0.
- If the DFA is in state q_1 when the input is finished, the DFA accepts the input string.

A SIMPLE DFA EXAMPLE

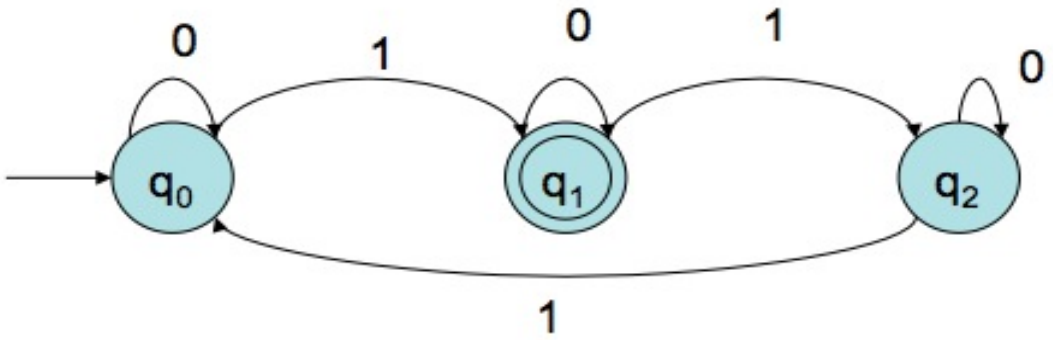


- What kinds of strings does this DFA accept?
  - It accepts $\omega = 00010000$
  - It accepts $\omega = 00010011001$
  - It accepts $\omega = 1$
  - It rejects $\omega = 1100001$
  - It rejects $\omega = 0110000$
- It accepts all strings $\omega \in \{0,1\}^*$ such that $n_1(\omega) = 1 \mod 3$

DFA – Formal Definition

- A Deterministic Finite State Acceptor (DFA) is defined as the 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states
  - Σ is a finite set of symbols – the alphabet
  - $\delta : Q \times \Sigma \rightarrow Q$ is the next-state function
  - $q_0 \in Q$ is the (label of the) start state
  - $F \subseteq Q$ is the set of final (accepting) states

Formal Description of the Example DFA



- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- δ :

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

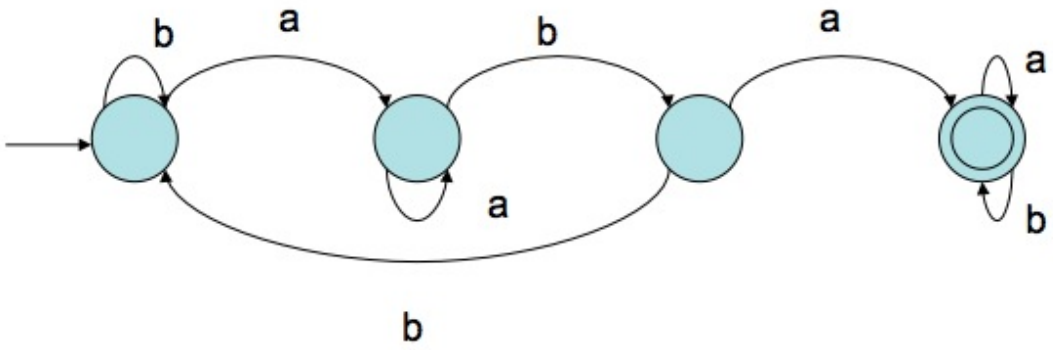
- q_0
- $F = \{q_1\}$

We will almost always use the graphical description for δ. The other components will always be implicit!

How the DFA Works

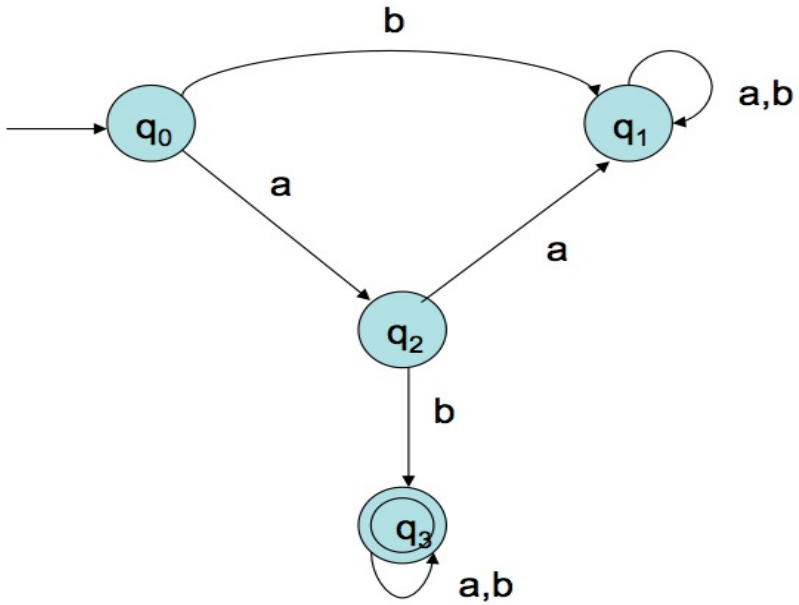
- The DFA accepts a string $\omega = x_1 x_2 \cdots x_n$ if a sequence of states $r_0 r_1 r_2 \cdots r_n$, $r_i \in Q$, exists, such that
  1. $r_0 = q_0$ (Start in the initial state)
  2. $r_i = \delta(r_{i-1}, x_i)$ for $i = 1, 2, \ldots n$
     - Move from state to state.
  3. $r_n \in F$
     - End up in a final state.
- If the DFA is NOT in an accepting state when the input string is exhausted, then the string is rejected.

DFA Example



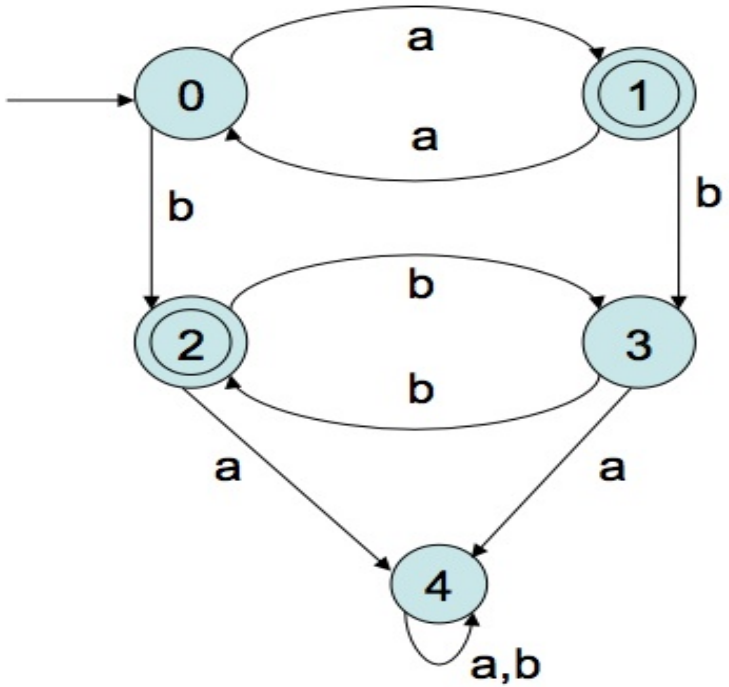
- This DFA accepts strings that have *aba* somewhere in it.
- Once the existence of *aba* is ascertained, the rest of the input is ignored!
- What do the states "remember"?
- What does it remind you of from string matching algorithms?

DFA Example



- This DFA accepts strings that start with *ab*
- Once the string starts with *ab* the rest is ignored!
- The state q_1 is known as a sink state.
  - Once a machine enters a sink state, there is no getting out! It is rejected.

DFA Example



- This DFA accepts strings of the sort $a^n b^m$ such that $n + m$ is odd.

A MORE INTERESTING DFA EXAMPLE

- Input is a string over $\Sigma = \{0, 1\}$
- We interpret the string as a binary number.
- We want to accept strings where the corresponding binary number is divisible by 3.
  - Accept e.g., 0, 11, 1001, 1100, 1111, 111100, ...
  - Reject e.g., 1, 10, 101, 10000, ...
- The most significant (leftmost) digit comes first!
- No obvious pattern at first sight!

A MORE INTERESTING DFA EXAMPLE

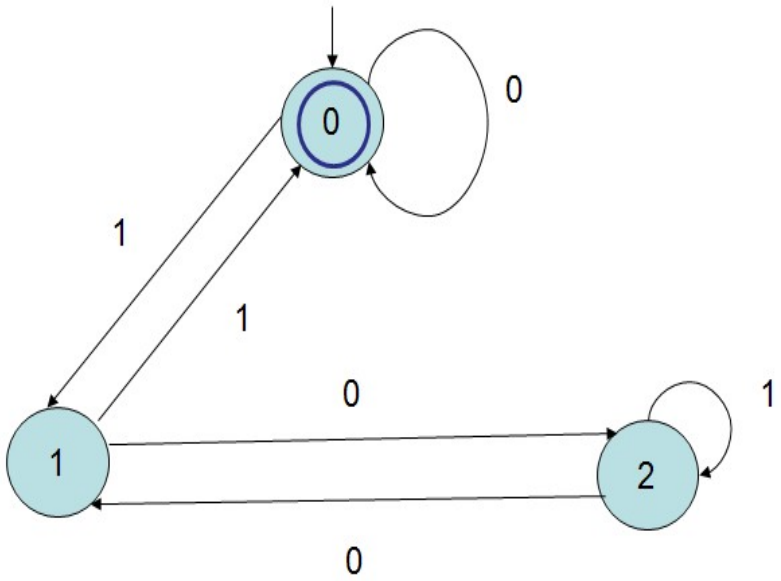
- How do we find the decimal equivalent of binary number digit-by-digit?
  1. `value=0`
  2. `repeat as long as there are more binary digits`
     - `value=value*2+input`

- $1101_2 \rightarrow 0 \cdot 2 + 1 = 1 \rightarrow 1 \cdot 2 + 1 = 3 \rightarrow 3 \cdot 2 + 0 = 6 \rightarrow 6 \cdot 2 + 1 = 13_{10}$
- We can not compute this number with a DFA, since the number can be arbitrarily large!
- However, for our problem, we can compute a running modulo 3 with a DFA!!

Computing a Running Modulo 3 Remainder

- Consider any number $n = 3p + r$. It has remainder r when divided by 3
- Multiply by 2 and add 0
  - $r = 0 : 2n + 0 = 2(3p + 0) + 0 = 3(2p) + 0 \rightarrow$ New r is 0.
  - $r = 1 : 2n + 0 = 2(3p + 1) + 0 = 3(2p) + 2 \rightarrow$ New r is 2.
  - $r = 2 : 2n + 0 = 2(3p + 2) + 0 = 3(2p + 1) + 1 \rightarrow$ New r is 1.
- Multiply by 2 and add 1
  - $r = 0 : 2n + 1 = 2(3p + 0) + 0 = 3(2p) + 1 \rightarrow$ New r is 1.
  - $r = 1 : 2n + 1 = 2(3p + 1) + 0 = 3(2p + 1) + 0 \rightarrow$ New r is 0.
  - $r = 2 : 2n + 1 = 2(3p + 2) + 0 = 3(2p + 1) + 2 \rightarrow$ New r is 2.
- This information now defines the state transition function
  - We let each state denote the remainder. So δ maps each remainder and input digit combination, to a new remainder.

A DFA FOR BINARY NUMBERS DIVISIBLE BY 3



- Running some examples:
  - For $11_2 = 3_{10} \Rightarrow 0 \rightarrow 1 \rightarrow 0 \Rightarrow$ Accept
  - For $1100_2 = 12_{10} \Rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 1 \Rightarrow 0$ Accept
  - For $1111_2 = 15_{10} \Rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \Rightarrow 0$ Accept
  - For $1010_2 = 10_{10} \Rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \Rightarrow 2$ Reject

The Extended State Transition Function

- $\delta : Q \times \Sigma \rightarrow Q$ is the state transition function. The input is a symbol.
- $\delta^* : Q \times \Sigma^* \rightarrow Q$ is the extended state transition function.
  - $\delta^*(q, \epsilon) = q$
  - $\delta^*(q, \omega \cdot a) = \delta(\delta^*(q, \omega), a)$, where $a \in \Sigma$ and $\omega \in \Sigma^*$
    - First, go (sort of recursively) where ω (a string) takes you, $(\delta^*(q, \omega) = q')$
    - Then, make a single transition with symbol a $(\delta(q', a))$

The Language Accepted by a DFA

- $L(M)$ denotes the language accepted by a DFA M

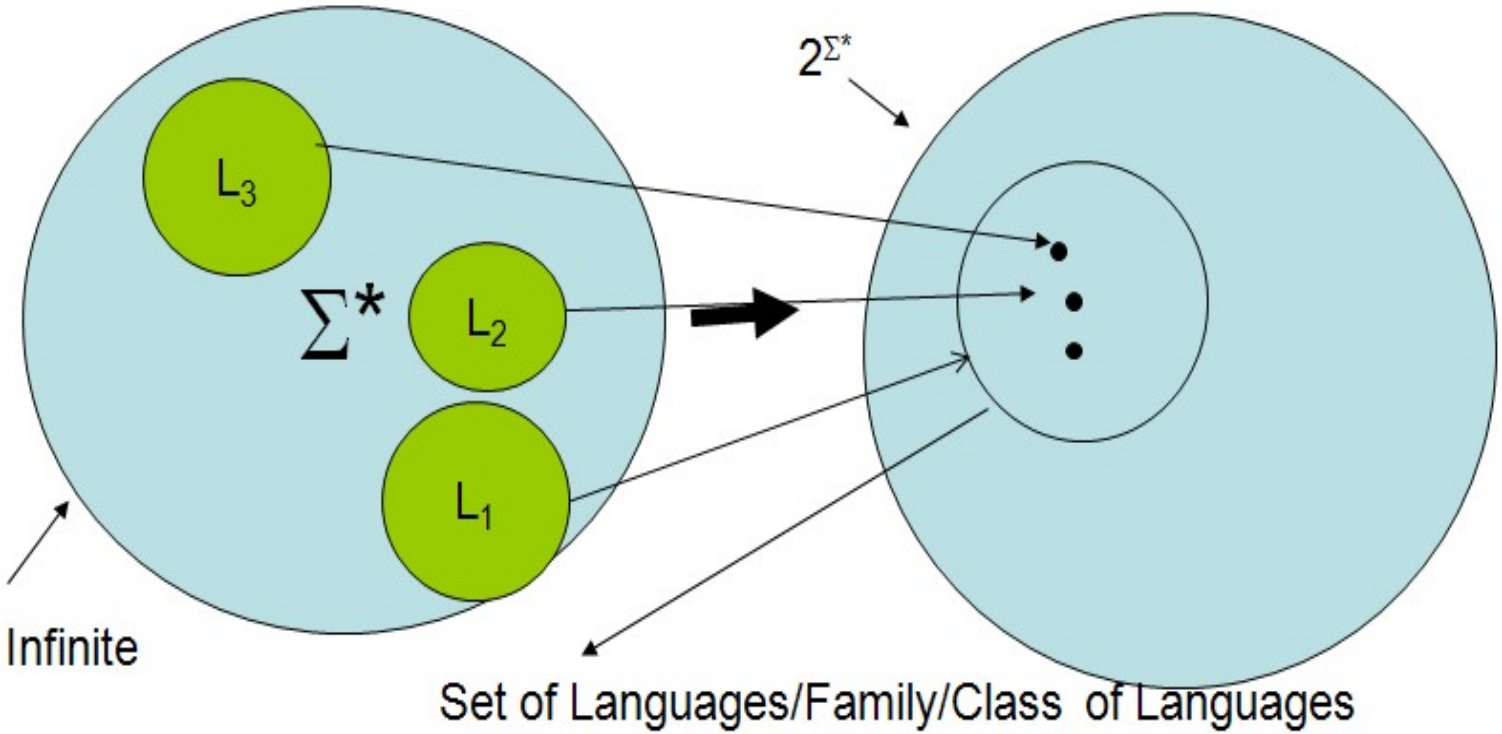
$$L(M) = \{\omega | \omega \in \Sigma^* \text{ and } \delta^*(q_0, \omega) \in F\}$$

- Similarly

$$\overline{L(M)} = \{\omega | \omega \in \Sigma^* \text{ and } \delta^*(q_0, \omega) \notin F\}$$

Regular Languages

A language L is called a regular language if and only if there exists a DFA M such that $L(M) = L$.



SAMPLE PROBLEMS

- Design a DFA for all strings over the alphabet $\Sigma = \{a, b\}$ that contain *aba* but not *abaa* as a substring.[1]
- Design a DFA for the language $L = \{w \mid w \text{ contains at least one } 0 \text{ and at most one } 1\}$
- Design a DFA for the language $L = \{w \mid w \text{ does not contain } 100 \text{ as a substring}\}$

[1] A substring is any consecutive sequence of symbols that occurs anywhere in a string. For example, *ab* and *bc* are substrings in *abc* while *cb* or *ac* are not.

Sample Problems

- Design a DFA for all strings over the alphabet $A = \{a, b, c\}$ in which no two consecutive positions are the same symbol. (5 states should be sufficient)
- Design a DFA for all strings over the alphabet $\{0, 1\}$ where the 3^{rd} symbol from the end is a 0.
- Design a DFA all strings over the alphabet $\{0, 1\}$ where the leftmost and the rightmost symbols are different.

SAMPLE PROBLEMS

- Design a DFA all strings over the alphabet $\{a, b, c\}$ where only two of the symbols occur odd number of times.
- Design a DFA all strings over the alphabet $\{a, b\}$ in which every substring of length four has at least two *b*'s. For example, *abbababbbaabbabba* is accepted, while *abbaaabbbb* is not, because the substring *aaab* does not contain two *b*'s. (At most 8 states should suffice.)
- Design a DFA all strings over $\{a, b\}$ in which every pair of adjacent 0's appears before any pair of adjacent 1's.