

# Lecture 2



# Mathematical Induction and Recursive Definitions



## The principle of mathematical induction

- Induction hypothesis [hai'poθisis]
- Basis step
- Induction step



- **THEOREM.** *All cows are the same color*
- **Proof.** Induction
- **Basis step.** Single cow ( $i=1$ ) is the same color
- **Induction hypothesis.** All sets of  $i$  cows are the same color
- **Induction step.** Divide the set of  $i+1$  cows into  $A=\{1,2,\dots,i\}$  and  $B=\{2,3,\dots,i+1\}$  sets.



We have that

- All cows in  $A$  are the same color
- All cows in  $B$  are the same color
- All cows in  $A \cap B$  are the same color

Ergo, all cows are the same color



## Recursive definitions

- $F(n)=n!$
- $F(1)=1,$
- $F(n)=F(n-1)*n$



$L^*$

$$L^* = \bigcup_{i=0}^{\infty} L^i$$



## Recursive definition of $L^*$

1.  $\Lambda \in L^*$ .
2.  $\forall x \in L^*$  and  $\forall y \in L$ ,  $xy \in L^*$
3. Every string is obtained by using only 1 and 2



## Recursion and induction

- In some situations the statements to be proved inductively is about some recursively defined construct. One can prove results about the constructed objects by induction on the number of steps used in its construction. Such thing is called *structural induction*.



## Example

- Number of leaves in a binary tree



## PART II

### Regular languages and Finite automata



## How to describe a language?

- Describe how its strings can be generated from simpler strings
- Specify an algorithmic procedure for recognizing whether a given string is in the language



## Basic languages

- $\{a\}$
- The empty language  $\emptyset$
- $\{\Lambda\}$



## Regular language over an alphabet $\Sigma$

Obtained from basic languages by

- Union
- Concatenation
- Kleene\*



## Regular Expressions ( $\cup$ to $+$ ) over $\{0,1\}$

- |   |                                  |
|---|----------------------------------|
| ■ $\{\Lambda\}$                               | $\Lambda$                        |
| ■ $\{0\}$                                     | $0$                              |
| ■ $\{001\}$                                   | $001$                            |
| ■ $\{0,10\}$                                  | $0+10$                           |
| ■ $\{1,\Lambda\}\{001\}$                      | $(1+\Lambda)001$                 |
| ■ $\{10,111,11010\}^*$                        | $(10+111+11010)^*$               |
| ■ $\{0,10\}^*\{\{11\}^*\cup\{001,\Lambda\}\}$ | $(0+10)^*(\{11\}^*+001+\Lambda)$ |



## Definition of regular languages and corresponding regular expressions over $\Sigma$

The set  $R$  of regular languages over  $\Sigma$

- |  |                  |
|--|------------------|
| 1. $\emptyset \in R$                     | 1. $\emptyset$   |
| 2. $\{\Lambda\} \in R$ ,                 | 2. $\Lambda$     |
| 3. $\forall a \in \Sigma, \{a\} \in R$ , | 3. $a$           |
| 4. If $L_1, L_2 \in R$ ,                 | 4. $(r_1, r_2)$  |
| a) $L_1 \cup L_2 \in R$                  | a) $(r_1 + r_2)$ |
| b) $L_1 L_2 \in R$                       | b) $(r_1 r_2)$   |
| c) $L_1^* \in R$                         | c) $(r_1^*)$     |



## Shortcuts

- $(r^2)$  for  $(rr)$
- $(r^+)$  for  $((r^*)r)$
- Removal parentheses



## Identity of expressions

- Example:  $(0+1)^*01(0+1)^*+1^*0^*=(0+1)^*$
- Explanation:  $(0+1)^*$  all possible strings from 0's and 1's
- Every such string is either of type *something*01*something*, or 1...110...0



## Example: strings of even length

- Is L regular? What is its corresponding regular expression?
- $L = \{00, 01, 10, 11\}^*$
- $(00+01+10+11)^* = ((0+1)(0+1))^*$



## Example: The language of Java Identifiers

- Briefly, a valid Java identifier must start with a Unicode letter, underscore (`_`), or dollar sign (`$`). The other characters, if any, can be a Unicode letter, underscore, dollar sign, or digit



## The language of Java identifiers

- $l = a+b+L +z+A+B+L +Z+a+b+L +\text{я}+..$
- $d = 0+1+2+L +9$
- $(l+_+\$)(l+d+_+\$)^*$

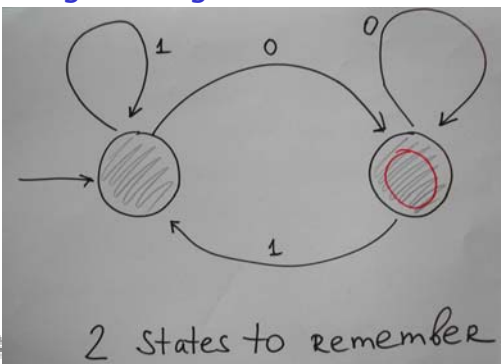


## The memory required to recognize a language

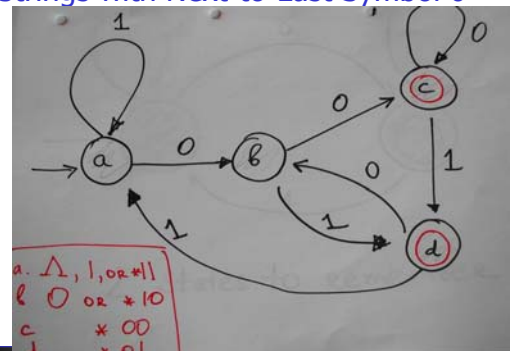
- Single pass
- End of the string is marked
- Make a decision after each input symbol



## Strings ending with 0



## Strings with Next-to-Last Symbol 0



## Recognition Algorithm via machine



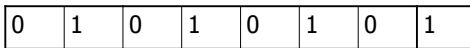
## Finite automaton (finite-state machine, FA)

A *finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$

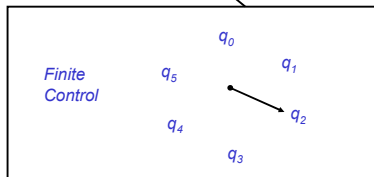
- $Q$  is a *finite* set of *states*
- $\Sigma$  is an *alphabet*
- $\delta : Q \times \Sigma \rightarrow Q$  is a *transition function*
- $q_0 \in Q$  is the *initial state*
- $A \subseteq Q$  is a set of *final* or *accepting* states



Input tape



Reading head



## Describing FA

- Transition diagram
- Transition table

		INPUT	
		0	1
STATE	A	B	A
	B	A	B



FA  $M=(Q, \Sigma, q_0, A, \delta)$

$\delta(q, a)$  is the state to which FA goes if it is in state  $q$  and receives input  $a$

$\delta^*(q, x)$  is the state where FA ends up if it begins in state  $q$  and receives string  $x$



## Recursive definition of the extended transition function $\delta^*$

Let  $M=(Q, \Sigma, q_0, A, \delta)$  be an FA.

$\delta^* : Q \times \Sigma^* \rightarrow Q$

1. For  $\forall q \in Q, \delta^*(q, \Lambda) = q$
2. For  $\forall q \in Q, \forall y \in \Sigma^*, \forall a \in \Sigma$   
 $\delta^*(q, ya) = \delta(\delta^*(q, y), a)$



## Acceptance by FA

Let  $M=(Q,\Sigma, q_0, A, \delta)$  be an FA.

A string  $x \in \Sigma^*$  is **accepted** by M if

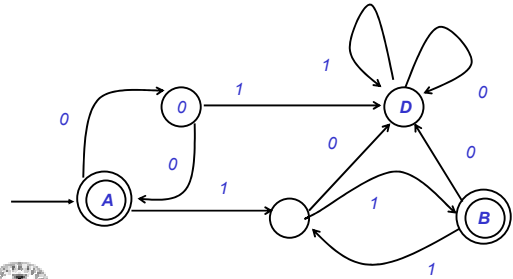
$\delta^*(q_0, x) \in A$ . Otherwise, **rejected**.

The language **accepted** by M:

$L(M)=\{x \in \Sigma^* : x \text{ is accepted by } M\}$



## How to find a regular expression corresponding to an FA?



## Answer

- $\{00\}^*\{11\}^*$

