CS3063 Theory of Computing

Semester 4 (20 Intake), Feb – Jun 2023

Lecture 2

Regular Languages & Finite Automata

Sanath Jayasena

Announcements

- We follow the **flipped-classroom model**
- Video recorded lecture and slides on Moodle; students study these privately in their own time
- Physical meeting every week from 13th March, on Mondays, by dividing the students into 2 Groups
  - Start with a short online Quiz on the previous week's lecture
  - Group 1: Monday 10.15am – 11.00am in L2 Lab
  - Group 2: Monday 11.15am – 12.00 noon in L2 Lab

Today’s Outline

Lecture 2

- Regular Languages
- Regular Expressions
- Finite Automata (FA)
- Kleene’s Theorem

PART 1

Today's Outline

Lecture 2

- Regular Languages
- Regular Expressions
- Finite Automata (FA)
- Kleene's Theorem

Introduction

- Consider the languages obtained by concatenation of simple languages of the form $\{a\}$ where $a \in \Sigma$

- If we use concatenation only, then we get single strings (languages with one string)

- Adding union and Kleene * operations, we can produce infinite languages

Basic Languages

- To the simple language of the form $\{a\}$, let us add the *empty language* $\emptyset$ and the language $\{\Lambda\}$ that has only the null string, to get the basic languages

- **Basic languages**: $\{a\}$, $\emptyset$ and $\{\Lambda\}$

Regular Languages/Expressions

- **Regular language** over an alphabet Σ
  - *The language that can be obtained from the basic languages using the **union**, **concatenation** and **Kleene** * operations*

- A regular language can be represented by a simple form called a **regular expression**

Regular Languages/Expressions

- Regular expression for a regular language is obtained by:
  1) Leaving out { and } or replacing with (and)
  2) Replacing ∪ with |

(some use "+") ; note that "0+1" and "0^+1" are different

- Example: let $\Sigma = \{0, 1\}$
  - Some regular languages over Σ and the corresponding regular expressions are: (see next slide)

Regular Language	Regular Expression
$\{\Lambda\}$	
$\{0\}$	
$\{001\}$ (i.e., $\{0\}\{0\}\{1\}$)	
$\{0, 1\}$ (i.e., $\{0\} \cup \{1\}$)	
$\{0, 10\}$ (i.e., $\{0\} \cup \{10\}$)	
$\{1, \Lambda\}\{001\}$	
$\{110\}^* \{0,1\}$	
$\{1\}^*\{10\}$	
$\{10, 111, 11010\}^*$	
$\{0, 10\}^*(\{11\}^* \cup \{001, \Lambda\})$	

Regular Language	Regular Expression
$\{\Lambda\}$	Λ
$\{0\}$	0
$\{001\}$ (i.e., $\{0\}\{0\}\{1\}$)	001
$\{0, 1\}$ (i.e., $\{0\} \cup \{1\}$)	$0 \mid 1$ or $0 + 1$
$\{0, 10\}$ (i.e., $\{0\} \cup \{10\}$)	$0 \mid 10$ or $0 + 10$
$\{1, \Lambda\}\{001\}$	$(1 \mid \Lambda)001$ or $(1 + \Lambda)001$
$\{110\}^* \{0,1\}$	$(110)^* (0 \mid 1)$
$\{1\}^*\{10\}$	1^*10
$\{10, 111, 11010\}^*$	$(10 \mid 111 \mid 11010)^*$
$\{0, 10\}^*(\{11\}^* \cup \{001, \Lambda\})$	$(0 \mid 10)^* ((11)^* \mid 001 \mid \Lambda)$

Regular Expressions

- A regular expression indicates the most typical string in a regular language

- Example

  1*10 is a string that consists of any number of 1's followed by the substring 10

Recursive Definition

- Let Σ be an alphabet; the regular expressions and the corresponding set R of regular languages over Σ are defined recursively as follows:

1. $\emptyset$ is a regular expression; denotes $\emptyset$ in R
2. Λ is a regular expression; denotes $\{\Lambda\}$ in R
3. For each a in Σ, a is a regular expression and it denotes the language $\{a\}$ in R

contd…

Recursive Definition ...contd

4. If p and q are regular expressions denoting languages P and Q, respectively, in R then:
   - $(p \mid q)$ is a regular expression; denotes $P \cup Q$ in R
   - (pq) is a regular expression; denotes PQ in R
   - (p^*) is a regular expression; denotes P^* in R

- Only those obtained from 1 - 4 above are regular expressions/languages over Σ

More on Regular Expressions

- The empty language $\emptyset$ is used in the definition mainly for consistency
  - Else, some trivial cases can be complicated

- Some notations for regular expressions
  - (x^i) means $(\underbrace{xx\cdots x}_{i \text{ times}})$
  - (x^+) means $((x^*)\ x)$

A Better Notation

- To omit some parentheses let's assume:
  - Kleene * : highest precedence
  - Concatenation: higher precedence than "|"
  - "|" has lowest precedence

- Examples
  - $(a|((b^*)c)) \rightarrow a|b^*c$ or $a+b^*c$
  - $((0(1^*))|0) \rightarrow 01^*|0$ or 01^*+0

More on Regular Expressions

- If two regular expressions p and q correspond to the same language then we write $p = q$, else $p \neq q$

- Examples

  1*(1 | Λ) = ? → 1*

  1*1* = ? → 1*

  0* | 1* = ? → 1* | 0*

  (0*1*)* = ? → (0 | 1)*

<table>
<tr><th>Regular Expression</th><th>Description</th></tr>
<tr><td>(0|1)*</td><td>?</td></tr>
<tr><td>(0|1)*00(0|1)*</td><td>?</td></tr>
<tr><td>(1|10)*</td><td>?</td></tr>
<tr><td>(0|Λ)(1|10)*</td><td>?</td></tr>
<tr><td>(0|1)*011</td><td>?</td></tr>
<tr><td>0*1*2*</td><td>?</td></tr>
<tr><td>00*11*22*</td><td>?</td></tr>
</table>

Regular Expression	Description
(0|1)*	All strings of 0’s and 1’s
(0|1)*00(0|1)*	All strings of 0’s and 1’s with at least 2 consecutive 0’s
(1|10)*	All strings of 0’s and 1’s beginning with 1 and no consecutive 0’s
(0|Λ)(1|10)*	All strings of 0’s and 1’s not having consecutive 0’s
(0|1)*011	All strings of 0’s and 1’s ending in 011
0*1*2*	Any # of 0’s followed by any # of 1’s followed by any # of 2’s
00*11*22*	0*1*2* with at least one of 0, 1, 2

More Examples

From the textbook (pp. 87-89)

- Suppose $\Sigma = \{0, 1\}$; give regular expressions for the following
  a) Strings of even length → ?
  b) Strings with an odd number of 1's → ?
  c) Strings of length 6 or less → ?
  d) Strings ending in 1, not containing 00 → ?

Solutions

- $\Sigma = \{0, 1\}$; regular expressions are:

a) Strings of even length → (00|01|10|11)*
b) Strings with an odd number of 1's → 0*10*(10*10*)* or (0*10*1)*0*10* or 0*(10*10*)*10*
c) Strings of length 6 or less → $(0|1|\Lambda)^6$
d) Strings ending in 1, not containing 00 → $(1|01)^+$

PART 2

Today's Outline

Lecture 2

- Regular Languages
- Regular Expressions
- Finite Automata (FA)
- Kleene's Theorem

Recognizing a Language

- Recognizing a language: *deciding if an arbitrary string is in the language*
- Can use the following approach
  - Use a single pass of input string, left→right
  - Rather than wait until ending symbol, make a tentative decision after each symbol
- How much memory is needed?
  - We must remember something

Finite Automata (FA)

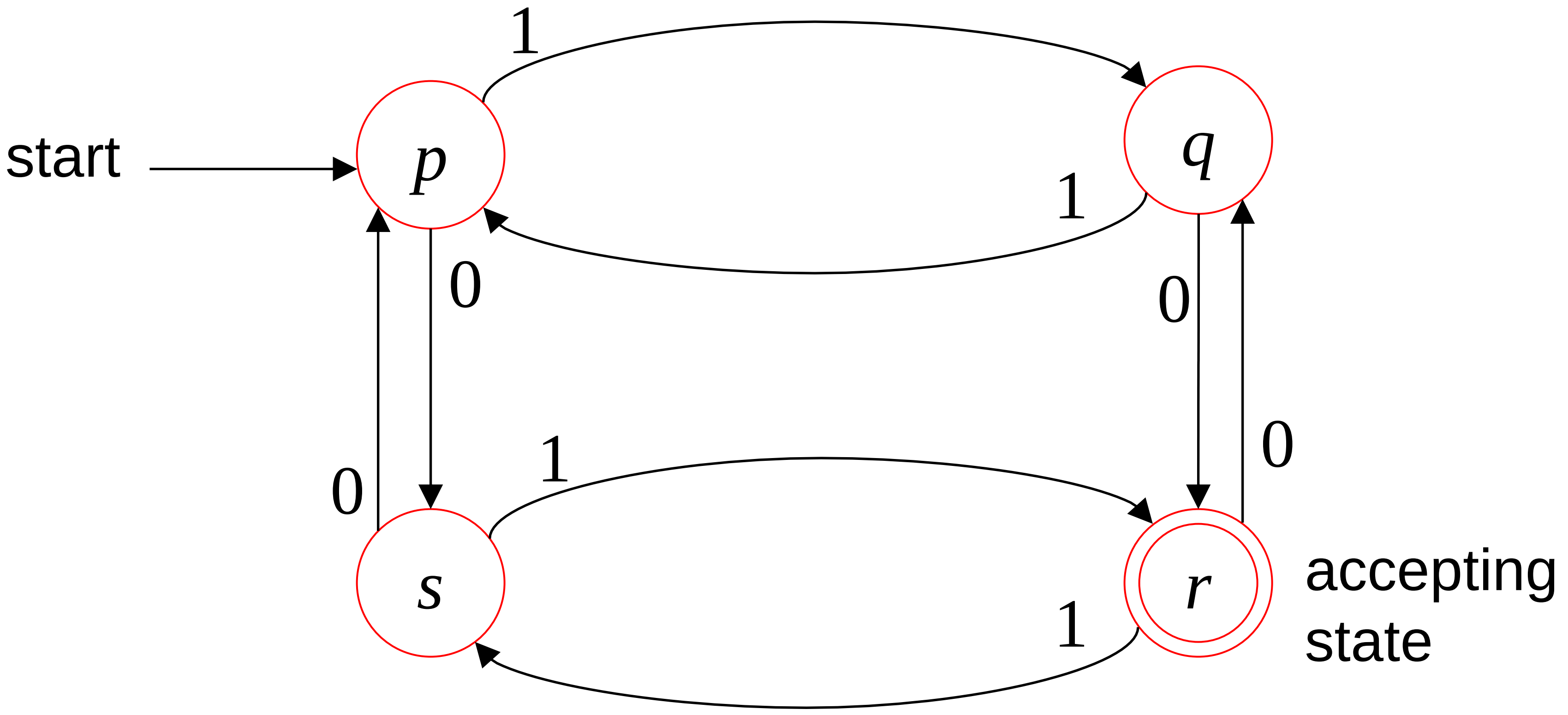
- A *finite automaton* (FA) consists of a finite set of states and a set of transitions from state to state that occur on input symbols from an alphabet

- For each input symbol, there is exactly one transition out of each state
  - Transition can be back to the state itself

Finite Automata (FA) …contd

- A directed graph called a (*state*) *transition diagram* can represent an FA
  - Vertices $\leftrightarrow$ states
  - Edge labeled a from vertex p to q $\leftrightarrow$ transition from state p to q on input a
  - The FA *accepts* a string x if the sequence of transitions for the symbols in x leads from the *start state* to an *accepting state*

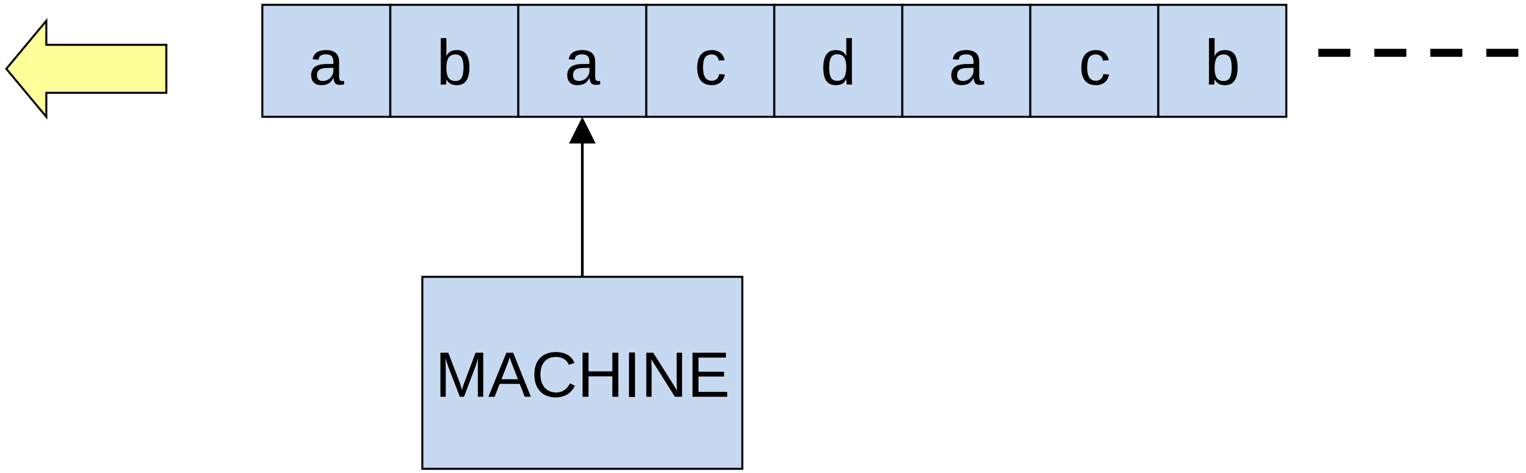
Transition Diagram



Inputs = {0, 1}

Finite Automata (FA) ...contd

- Can view an FA also as a machine in some state, reading a sequence of symbols from Σ on a tape



Finite Automata: Definition

- An FA is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where:
  - Q is a finite set of states
  - Σ is a finite alphabet of input symbols
  - $q_0 \in Q$ is the initial (or start) state
  - $A \subseteq Q$ is the set of accepting (or final) states
  - δ is the transition function; maps $Q \times \Sigma$ to Q

- $\delta(q, a)$ will be the new state of the FA, if it is now in state q and receives input a

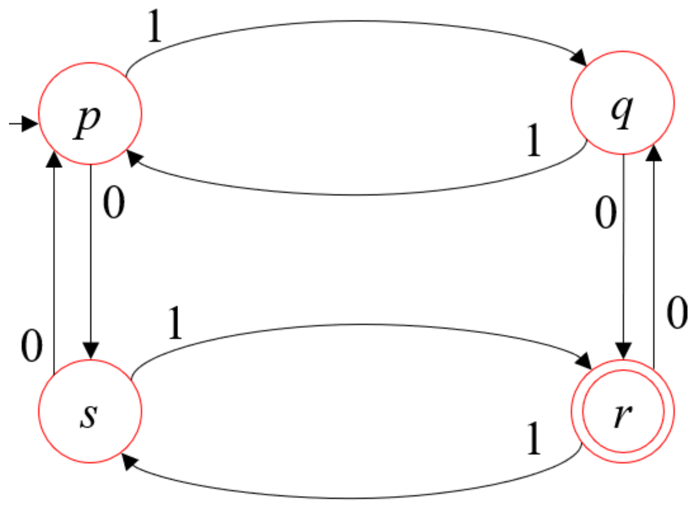
Finite Automata (FA) ...contd

- In our machine on slide 26, after reading a symbol a while in state q, the machine enters state $\delta(q, a)$ and moves its head one symbol to the right

- If $\delta(q, a)$ is an accepting state, then the FA accepts the string up to a on the tape

(State) Transition Table

- Alternative representation for an FA
  - E.g., transition table corresponding to the transition diagram on slide 25



Current State	Input	
	0	1
p	s	q
q	r	p
r	q	s
s	p	r

next state

Extended Transition Function δ*

- We extend δ to describe concisely what happens to an FA on an input string x

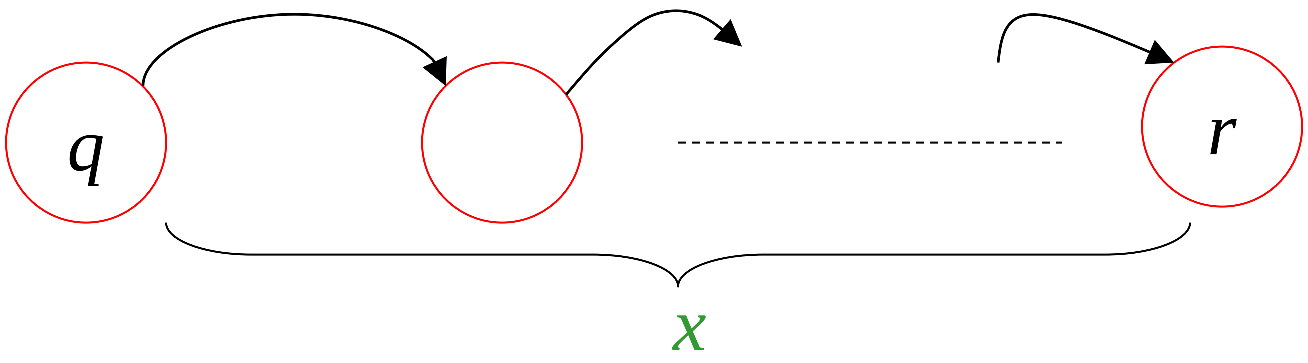
- Definition: The function $\delta^*: Q \times \Sigma^* \rightarrow Q$ is such that:
  - For any $q \in Q$, $\delta^*(q, \Lambda) = q$
  - For any $q \in Q$, $y \in \Sigma^*$ and $a \in \Sigma$,

$$\delta^*(q, ya) = \delta\,(\delta^*(q, y), a)$$

Extended Transition Function δ*

- $\delta^*(q, x)$ is the state the FA will be in after reading the string x starting in state q

- In the transition diagram, there is a path labeled x from q to some unique state r



Acceptance by an FA

- Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA
- A string $x \in \Sigma^*$ is accepted by M if $\delta^*(q_0, x)$ is in A
- If a string is not accepted, then it is rejected by M

- The language accepted (or recognized) by M is the set

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

Regular Languages and FA

- **Kleene's Theorem**
  - *A language $L \subseteq \Sigma^*$ is regular if and only if there is an FA with alphabet Σ that accepts L*
- This means:
  - If M is an FA, there is a regular expression corresponding to the language $L(M)$
  - Given a regular expression, there is an FA that accepts the corresponding language

Conclusion

- Summary of discussion today
  - Regular languages
  - Regular expressions
  - Finite automata (FA)
  - Acceptance by FA
  - Kleene’s Theorem