CZ4045 Natural Language Processing

Tutorial 1: Regular Expressions and FSA

Question Q1

- Write regular expressions for the following languages. By “word”, we mean an alphabetic string separated from other words by whitespace, any relevant punctuation, line breaks, and so forth. (**HINT**: please consult the book Chapter 2.1 or some websites on regular expressions.)

1. The set of all alphabetic strings;
2. The set of all lower case alphabetic strings ending with a letter *b*;
3. The set of all strings with two consecutive repeated words (e.g., “Humbert Humbert” and “the the” but not “the bug” or “the big bug”);
4. All strings that start at the beginning of the line with an integer and that end at the end of the line with a word;
5. All strings that have both the word *grotto* and the word *raven* in them (but not, e.g., words like *grottos* that merely *contain* the word *grotto*);

Question 1.1, 1.2

- The set of all alphabetic strings;
  - **[a-zA-Z]+**

- The set of all lower case alphabetic strings ending with a letter b;
  - **[a-z]*b**

Question 1.3

- The set of all strings with two consecutive repeated words (e.g., “Humbert Humbert” and “the the” but not “the bug” or “the big bug”);
  - **([a-zA-Z]+)\s+\1**
- Explanation
  - [a-zA-Z]+ → all alphabetic strings
  - \s → whitespace (space, tab..)
  - \1 → used to refer to back to the first pattern in the expression which is put **inside a parentheses** ( )
    - We may have \2 or \3 to refer to the second and third patterns put inside parentheses.

Question 1.4

- All strings that start at the beginning of the line with an integer and that end at the end of the line with a word
  - **ˆ\d+\b.*\b[a-zA-Z]+$**

- Explanation
  - \d → a digit
  - \b → a word boundary
  - ^, $ → the **beginning** and **end** of a line
  - . → a wildcard expression that matches any single character (except a carriage return)
  - * → Kleene star, zero or more occurrences of he immediate previous character or regular expression
  - .* → any string of characters

Question 1.5

- All strings that have both the word `grotto` and the word `raven` in them (but not, e.g., words like `grottos` that merely contain the word `grotto`)
  - **(.*\bgrotto\b.*\braven\b.*)|(.*\braven\b.*\bgrotto\b.*)**

- Explanation
  - The two words grotto and raven may appear in any order.
  - There could be other strings around the two words

- http://regexr.com/

Question Q2

- Design an FSA that accepts a subset of valid web addresses.
- Be sure to accept addresses from the “.com” and “.org” domains, and handle an arbitrary amount of directory nesting.
- Accept at least “.html”, “.htm” and “.shtml” page types.

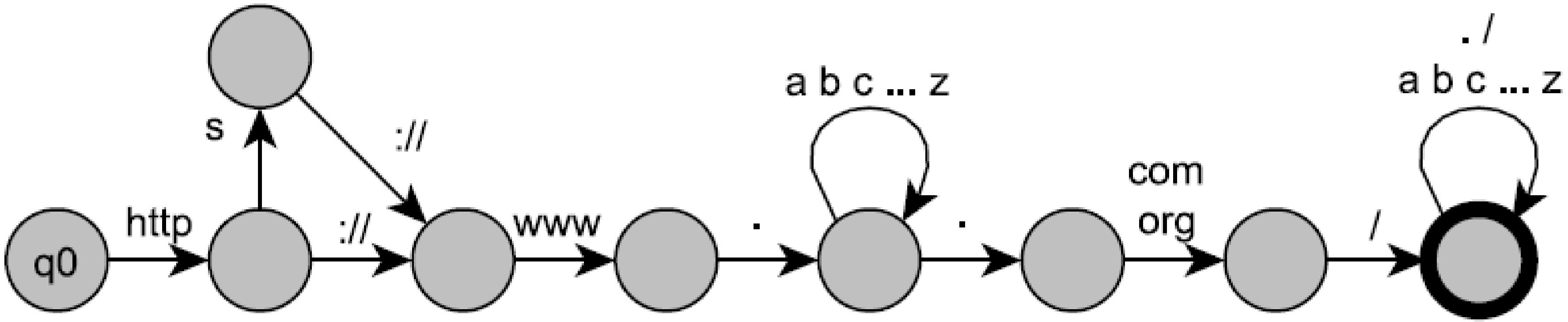
- Interested students can consult the official Web standards for other possible extensions to this recognizer.

Question Q2

- Example web addresses
  - http://teamsites.ntu.edu.sg/sce/default.aspx
  - http://www.ntu.edu.sg/library/facilities/Pages/LWNLearningCommons.aspx
  - http://confweb.cais.ntu.edu.sg/~cais/
  - http://www.delph-in.net/erg/
  - http://www.is.umk.pl/~duch/Wyklady/KogP/03-schem-min.htm#BA
  - http://stackoverflow.com/questions/164648/where-can-i-find-a-good-collection-of-public-domain-owl-ontologies-for-various-d
  - http://www.ascilite.org.au/ajet/ajet28/williams.html
  - http://www.cbioportal.org/public-portal/

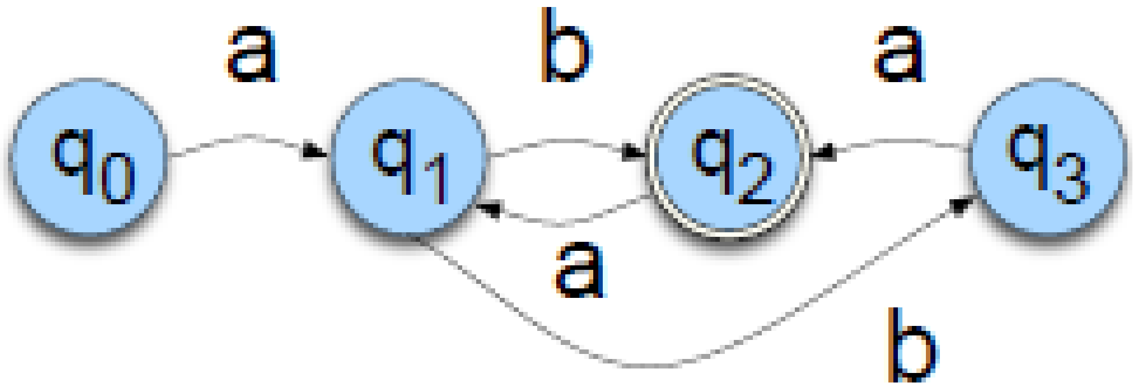
- Web addresses could be more complicated that these examples!

(simplified) Answer for Q2

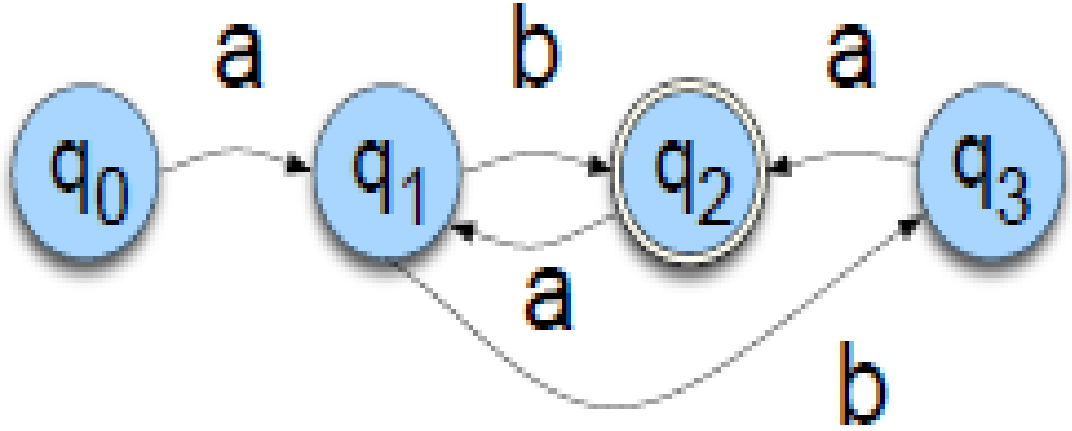


Question Q3

- Write a regular expression for the language accepted by the following NFSA.



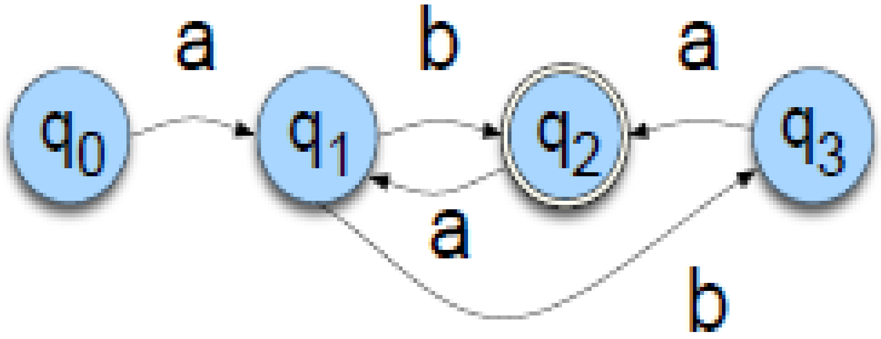
Hint: List the strings can be generated from the NFSA



- **ab**
- **aba**
- **abab**
- **ababa**
- **ababab**
- **abaab**
- **...**

Answer for Q3

- **(aba?)+**



q_0 ::= ε

q_1 ::= q_0a | q_2a

q_1 ::= a | q_2a

q_2 ::= q_1b | q_3a

q_3 ::= q_1b

q_2 ::= q_1b | $q_1$ba

q_2 ::= ab | $q_2$ab | aba | $q_2$aba

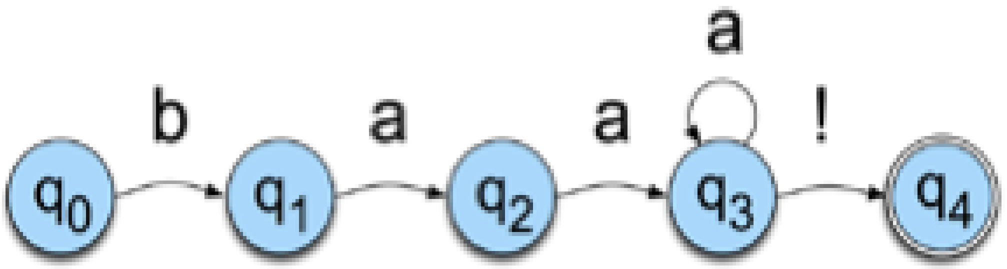
q_2 ::= ab | aba | $q_2$ab | $q_2$aba

q_2 ::= (aba?)|($q_2$aba?)

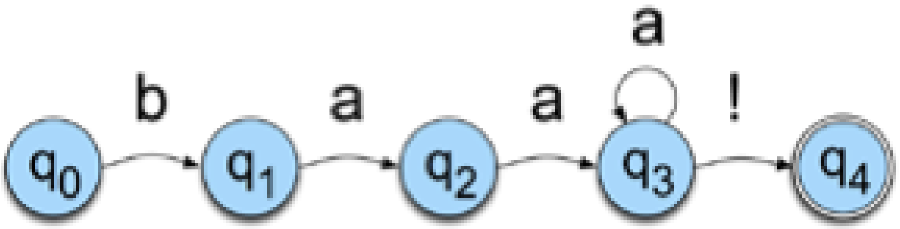
q_2 ::= (aba?)+

Question Q4

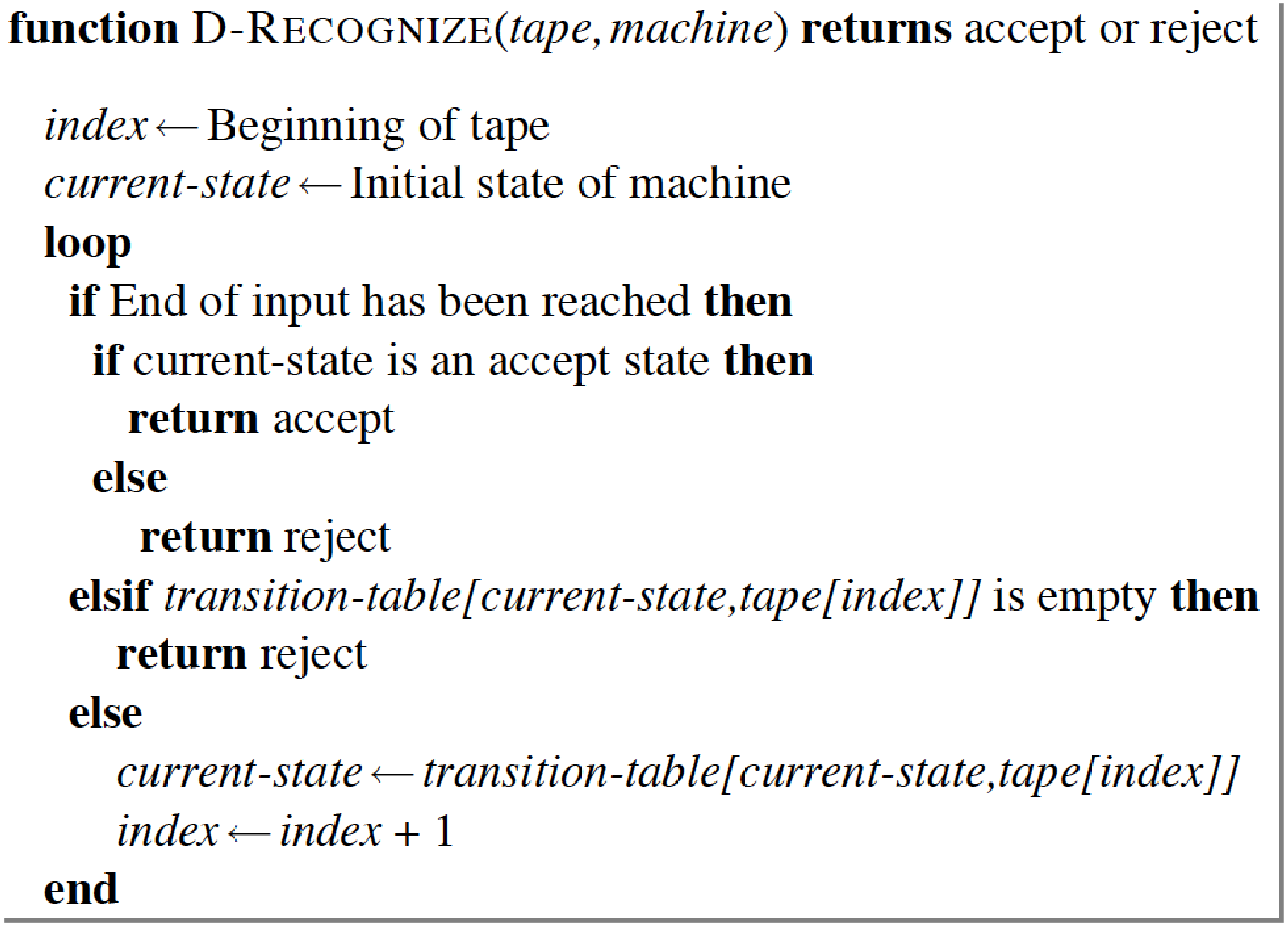
- Given the FSA in the following figure, please walk through the D-RECOGNIZE algorithm on strings *baaaa!* and *baac*, respectively.



Q4 D-Recognize



- *baaaa!*
- *baac*



```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
    if End of input has been reached then
      if current-state is an accept state then
        return accept
      else
        return reject
    elsif transition-table[current-state,tape[index]] is empty then
      return reject
    else
      current-state ← transition-table[current-state,tape[index]]
      index ← index + 1
  end
```

Question Q5

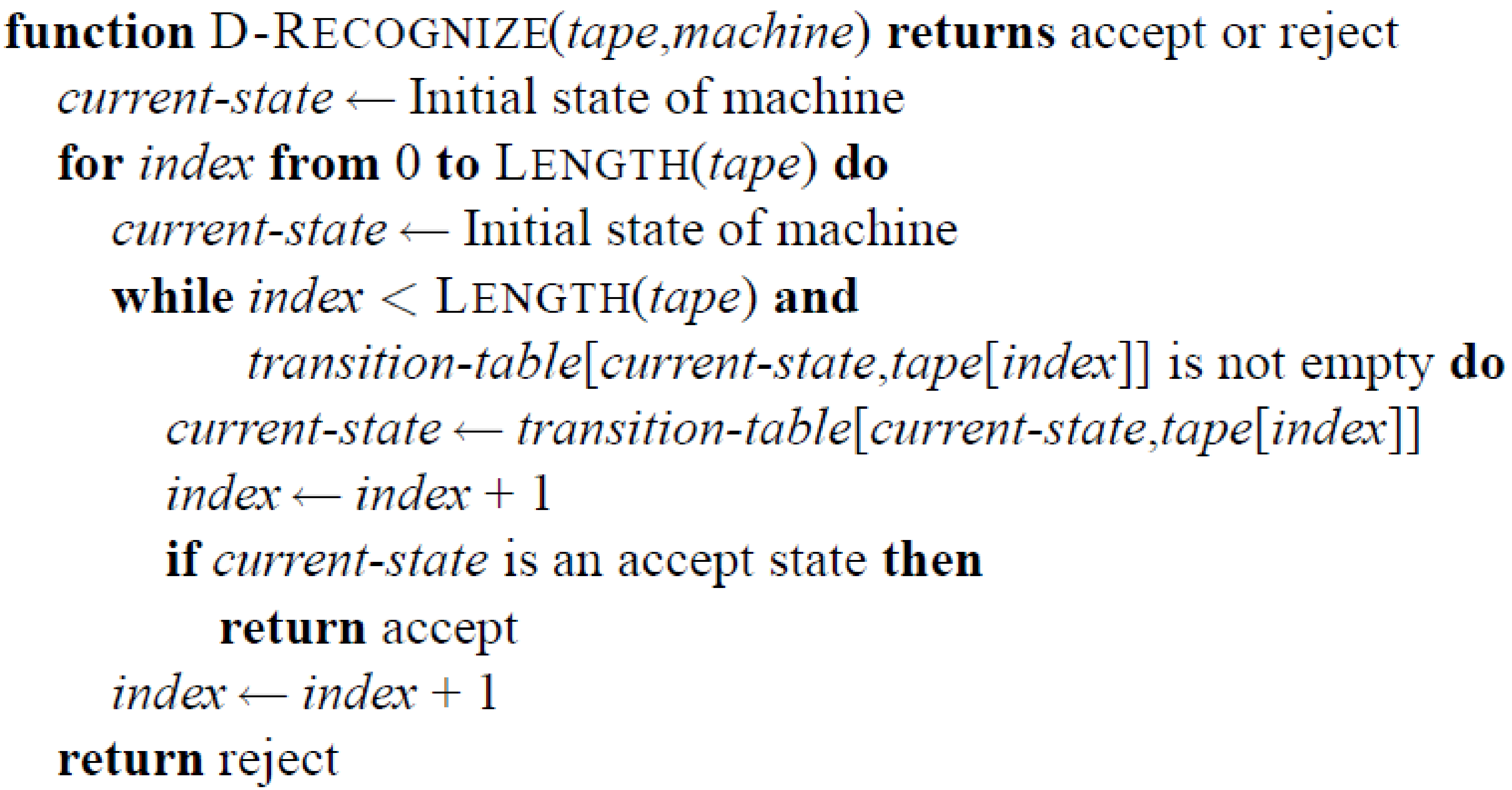
- Currently the function D-RECOGNIZE (as in Fig. 2.12 in the textbook) solves only a subpart of the important problem of finding a string in some text. Extend the algorithm to solve the following two deficiencies:
  - D-RECOGNIZE assumes that it is already pointing at the string to be checked, and
  - D-RECOGNIZE fails if the string it is pointing to includes as a proper substring of a legal string for the FSA. That is, D-RECOGNIZE fails if there is an extra character at the end of the string.

Question Q5: Analysis

- To address these problems, we will have to try to match our FSA at each point in the tape,
  - requires an additional outer loop,
- We will have to accept (the current substring) any time we reach an accept state.
  - requires a slightly different structure for our case statements

Question Q5: An improved version of D-Recognize

- – An additional outer loop
- – Accept (the current substring) when we reach an accept state



```
function D-RECOGNIZE(tape,machine) returns accept or reject
  current-state ← Initial state of machine
  for index from 0 to LENGTH(tape) do
    current-state ← Initial state of machine
    while index < LENGTH(tape) and
          transition-table[current-state,tape[index]] is not empty do
      current-state ← transition-table[current-state,tape[index]]
      index ← index + 1
      if current-state is an accept state then
        return accept
    index ← index + 1
  return reject
```