Natural Language Processing

Tutorial 1: Regular Expressions and Text Normalization

Dr. Sun Aixin

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");

  - **(\b[a-zA-Z]+\b)\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
  - **^\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 the 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 2

- **Try all your answers** on http://regexr.com/
- You may need to change the textbox to test two cases: the textbox contains one or more matched strings, and the textbox does not contain any matched string.
- What are the errors (e.g., false positive and false negative) have you observed?

Question 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**

- 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”);
  - **(\b[a-zA-Z]+\b)\s+\1**

Question 2

- 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
  - **^\d+\b.*\b[a-zA-Z]+$**

- 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.*)**

Question 3

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 1 & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

- Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of "idea" to "deal". Show your work.

Target

Source

	#	d	e	a	l
#	0	1	2	3	4
i	1				
d	2				
e	3				
a	4				

	#	d	e	a	l
#	0	1	2	3	4
i	1	1			
d	2	1			
e	3	2			
a	4	3			

	#	d	e	a	l
#	0	1	2	3	4
i	1	1	2		
d	2	1	2		
e	3	2	1		
a	4	3	2		

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\begin{cases} 1 & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

Question 3

- Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of "idea" to "deal". Show your work.

Target

Source

	#	d	e	a	l
#	**0**	1	2	3	4
i	1	1	2		
d	2	1	2		
e	3	2	1		
a	4	3	2		

	#	d	e	a	l
#	**0**	1	2	3	4
i	1	1	2	3	
d	2	1	2	3	
e	3	2	1	2	
a	4	3	2	1	

	#	d	e	a	l
#	**0**	1	2	3	4
i	1	1	2	3	4
d	2	1	2	3	4
e	3	2	1	2	3
a	4	3	2	1	2

Question 3

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\begin{cases} 1 & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

- Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of "idea" to "deal". Show your work.

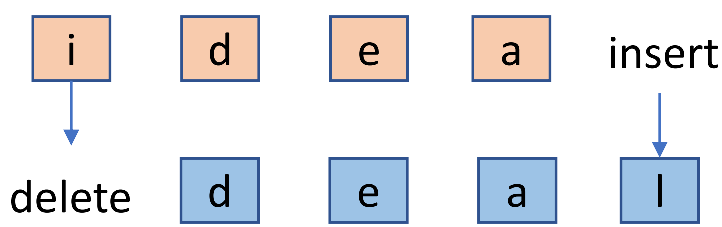
Target

Source

	#	d	e	a	l
#	**0**	1	2	3	4
i	1	1	2	3	4
d	2	1	2	3	4
e	3	2	1	2	3
a	4	3	2	1	2

	#	d	e	a	l
#	**0**	1	2	3	4
i	1	1	2	3	4
d	2	1	2	3	4
e	3	2	1	2	3
a	4	3	2	1	2

	#	d	e	a	l
#	**0**	1	2	3	4
i	1	1	2	3	4
d	2	1	2	3	4
e	3	2	1	2	3
a	4	3	2	1	2



Question 4

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\begin{cases} 1 & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

- Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of two sentences "compute the edit distance" to "the edit distance is computed". Show your work and show the alignment between the two strings.
  - If similar questions appear in exam, the requirement will be made clear, whether the edit distance is to be computed at character level or at word level.
  - We use the first character to represent each word, for simplicity and clarity.

Target

Source

	#	T	E	D	I	C
#	0	1	2	3	4	5
C	1					
T	2					
E	3					
D	4					

	#	T	E	D	I	C
#	0	1	2	3	4	5
C	1	1	2	3	4	5
T	2	1	2	3	4	5
E	3	2	1	2	3	4
D	4	3	2	1	2	3

Question 4

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 1 & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

- Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of two sentences "compute the edit distance" to "the edit distance is computed". Show your work and show the alignment between the two strings.

compute the edit distance

the edit distance is computed

Target

Source

	#	T	E	D	I	C
#	**0**	1	2	3	4	5
C	1	1	2	3	4	5
T	2	1	2	3	4	5
E	3	2	1	2	3	4
D	4	3	2	1	2	3

	#	T	E	D	I	C
#	**0**	1	2	3	4	5
C	1	1	2	3	4	5
T	2	1	2	3	4	5
E	3	2	1	2	3	4
D	4	3	2	1	2	3

Natural Language Processing

Regular Expressions and FSA

FSA is non-examinable.
For your information only.

Dr. Sun Aixin

Regular Expression and Finite state automata (FSA)

Regular expressions

- Regex is compact textual strings (e.g., "/[tT]he/"), which is perfect for specifying patterns in programs or command-lines
- Regex can be implemented as an FSA (Finite State Automata)

Finite state automata

- Graphs: **Nodes** are states and **edges** are transitions among states
- FSA has a wide range of uses

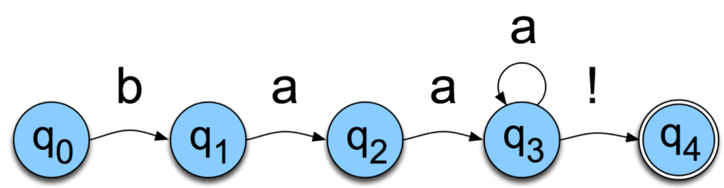
FSA vs Regex

- FSA can be described with a regular expression
- Regular expression is a textual way of specifying the structure of FSA

FSA as Graphs

FSA is non-examinable

- Example regular expression: /baa+!/
- Corresponding FSA

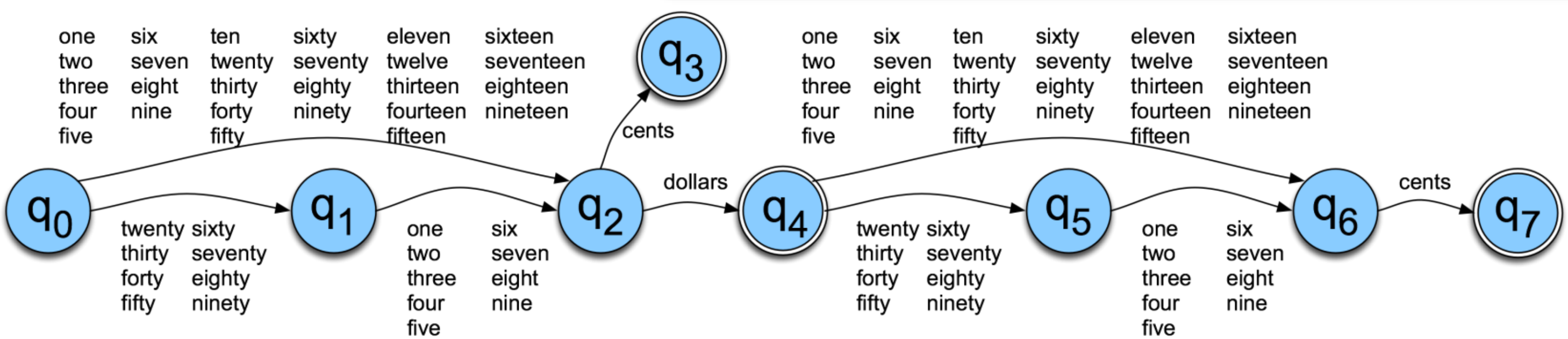


It has 5 states (q_0 to q_4)
b, **a**, and **!** are in its alphabet
q_0 is the start state
q_4 is an accept state
It has 5 transitions

FSA

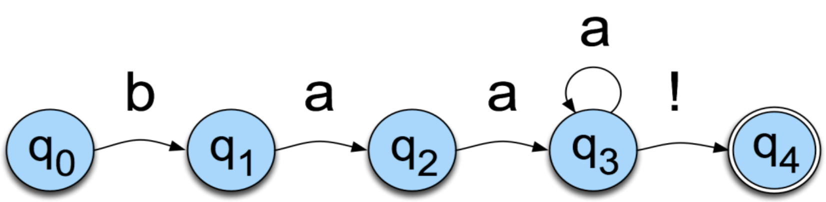
- A set of states: Q
- A finite alphabet: Σ
- A start state
- A set of accept/final states
- A transition function that maps $Q \times \Sigma \rightarrow Q$

Don't take term **alphabet** word too narrowly;
it just means we need **a finite set of symbols** in the input.



Yet Another View

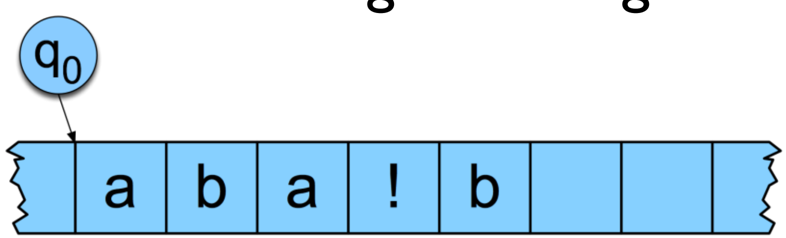
- An FSA can ultimately be represented as a **table**



If currently in state q_1 and an input is ***a***, then go to state q_2

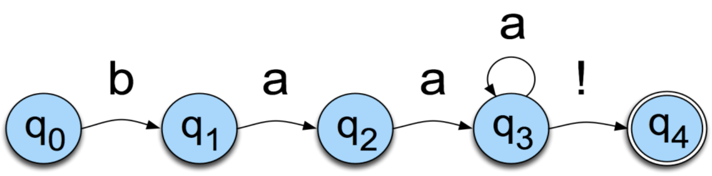
	Input		
State	b	a	!
q_0	1		
q_1		2	
q_2		3	
q_3		3	4
q_4:			

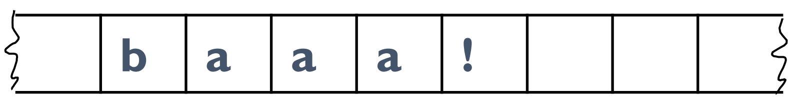
- **Recognition** is the process of determining if a string should be accepted by a machine

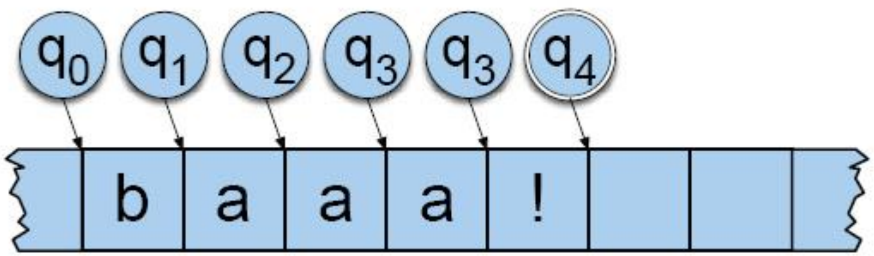


Recognition (D-Recognize)

- D-Recognize (in a tap-view)
  - Starting the process from the **start state**
  - Examining the **current input**
  - Consulting **the table**
  - Going **to a new state** and updating the tape pointer.
  - Until you run **out of tape.** Accept?



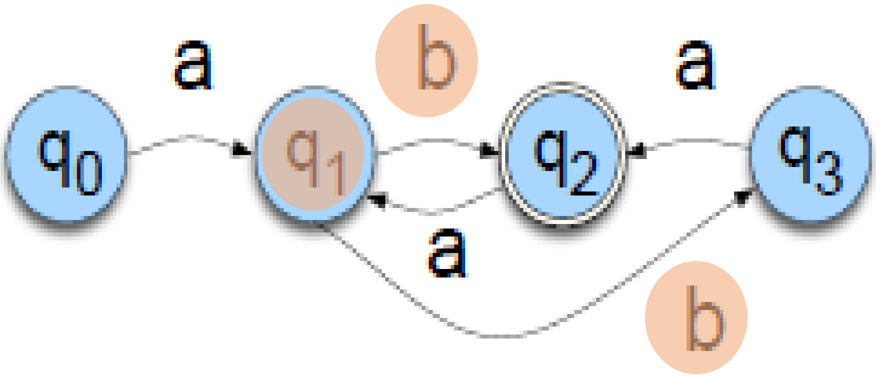




	Input		
State	b	a	!
q_0	1		
q_1		2	
q_2		3	
q_3		3	4
q_4:			

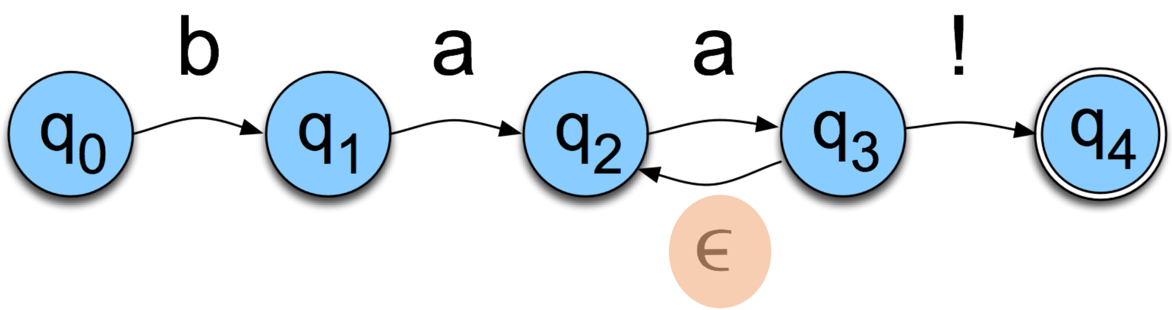
Deterministic FSA vs Non-Deterministic FSA

- If currently at a state, given an input
  - Deterministic FSA: There is only one next state to move to
  - Non-deterministic FSA: There are more than one possible states to move to



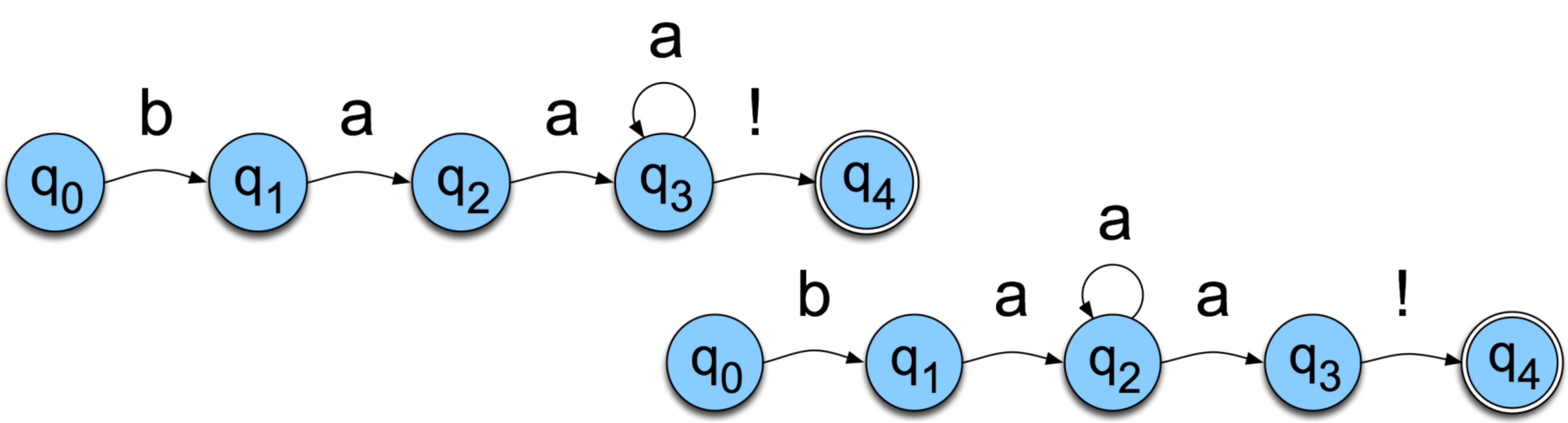
Deterministic FSA vs Non-Deterministic FSA

- There exists other form of Non-deterministic FSA
  - Epsilon transitions
  - These transitions do not examine or advance the tape during recognition



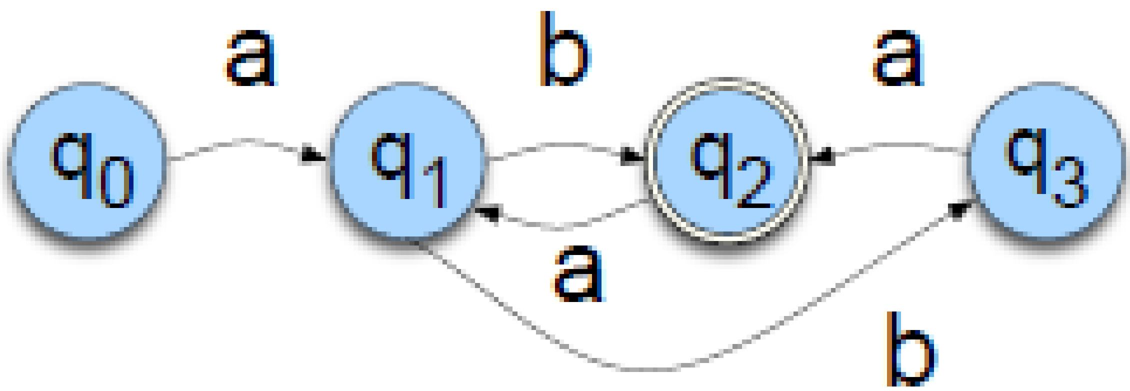
Deterministic FSA vs Non-Deterministic FSA

- Non-deterministic machines can be **converted** to deterministic through algorithm
- Non-deterministic machines **are not more powerful** than deterministic ones in terms of the languages they can and can't characterize
  - Not always obvious to users whether the regex that they've produced is deterministic or non-deterministic
  - Sometimes, non-determinism may look more natural (understandable)

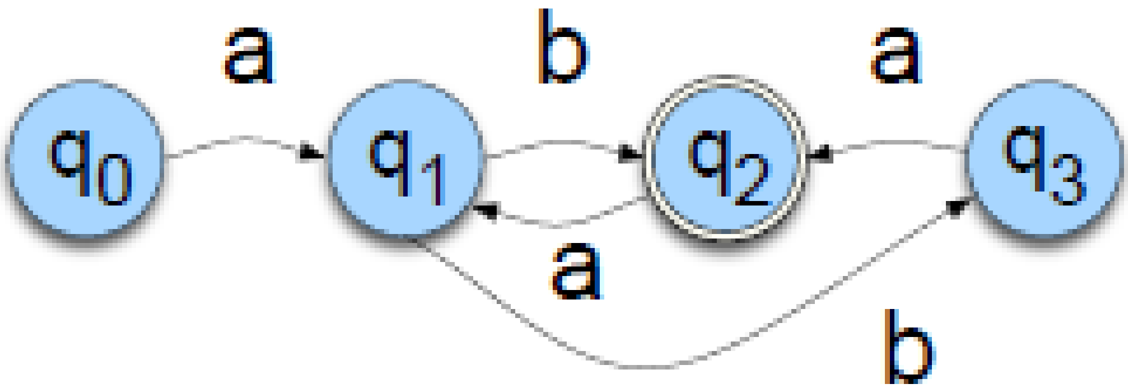


Additional Question on FSA

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



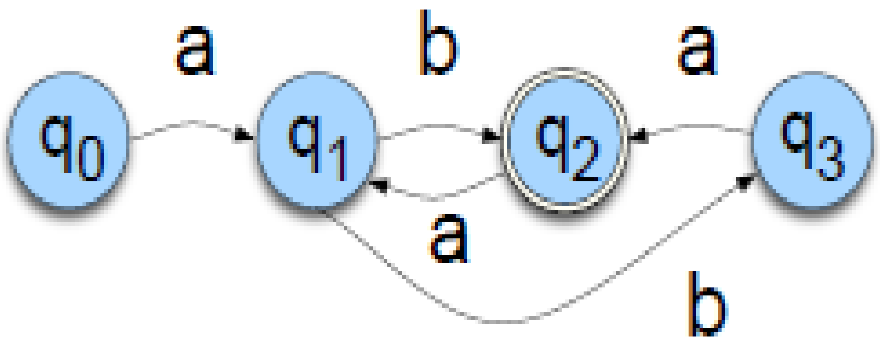
Hint: List the strings can be generated from the NFSA



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

Answer

- (aba?)+



$q_0 ::= \varepsilon$

$q_1 ::= q_0a \mid q_2a$

$q_1 ::= a \mid q_2a$

$q_2 ::= q_1b \mid q_3a$

$q_3 ::= q_1b$

$q_2 ::= q_1b \mid q_1ba$

$q_2 ::= ab \mid q_2ab \mid aba \mid q_2aba$

$q_2 ::= ab \mid aba \mid q_2ab \mid q_2aba$

$q_2 ::= (aba?) | (q_2aba?)$

$q_2 ::= (aba?)+$