CS310 : Automata Theory 2019

Lecture 7: Regular expressions and NFA

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2019-01-18

Topic 7.1

RE = RL

Class of languages

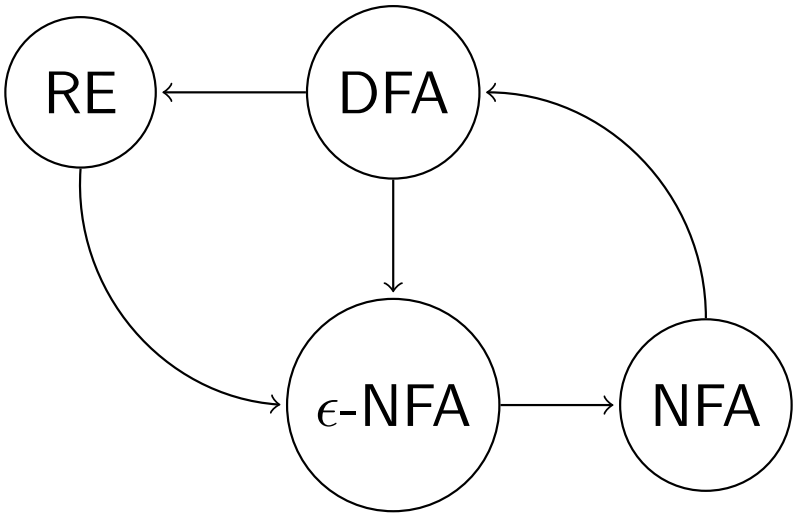
So far we have defined two classes of languages

- Regular languages
- Regular expressions

Are they same?

Commentary: As the name suggests they are same!

Language class equivalences



RE = Regular expression

Topic 7.2

DFA to regular expressions

DFA to Regular expressions

Theorem 7.1

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, then there is an RE R such that $L(R) = L(A)$.

Proof.

Let us assign states in $Q = \{q_1, .., q_n\}$ an arbitrary order, where $q_0 = q_1$ and

$$q_1 < .... < q_n.$$

Each path on a DFA corresponds to a word.

We will incrementally consider longer and longer paths. ...

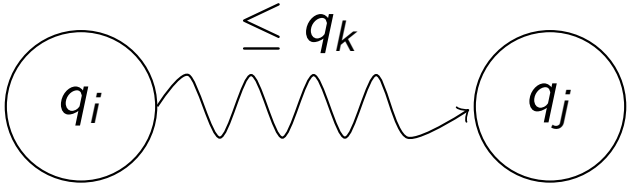
Commentary: For notational convenience, we have declared that q_0 is q_1 state.

DFA to Regular expressions

Proof(contd.)

Let us define the following set of paths.

$p(i,j,k) \triangleq$ the set of paths from q_i to q_j that do not have intermediate states that are greater than q_k.



Note that q_i and q_j need not be smaller than q_k.

Let $R(i,j,k)$ be the regular expression that defines the set of words along the paths in $p(i,j,k)$. ...

Exercise 7.1

a. What length of paths are in $p(i,j,0)$?

b. What kind of paths are in $p(i,j,1)$?

DFA to Regular expressions

Proof(contd.)

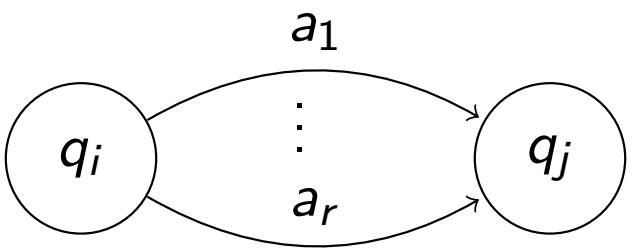
Let us define $R(i,j,k)$ by induction over k.

base case:

Let $\{a_1, ..., a_r\} = \{a | \delta(q_i, a) = q_j\}$, i. e., letters that take q_i to q_j.

$$R(i,j,0) \triangleq \begin{cases} a_1 + ... + a_r & i \neq j \\ a_1 + ... + a_r + \epsilon & \text{otherwise} \end{cases}$$

If $i = j$, empty string is also a possibility.



...

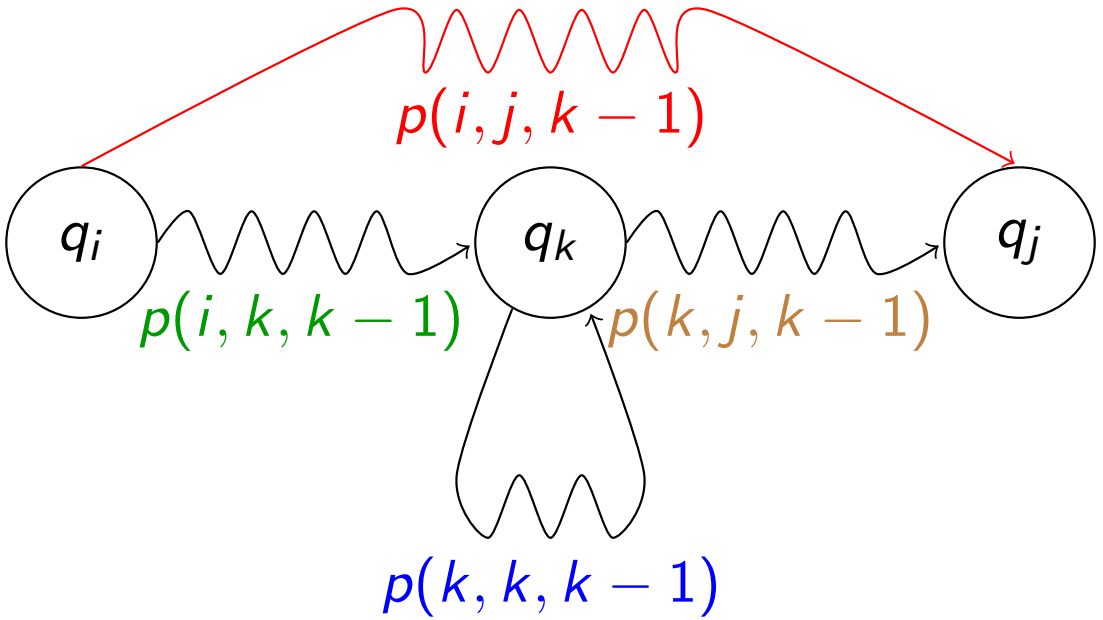
DFA to Regular expressions

Proof(contd.)

induction step:

By induction hypothesis, we have regular expressions for the paths upto $k-1$.

Let us consider the paths that also go via state q_k.



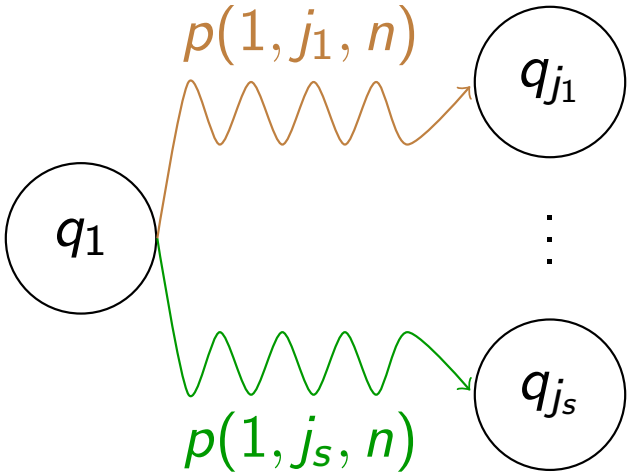
$$R(i,j,k) \triangleq R(i,j,k-1) + R(i,k,k-1)R(k,k,k-1)^*R(k,j,k-1)$$

...

DFA to Regular expressions

Proof(contd.)

Let $F = \{q_{j_1}, ..., q_{j_s}\}$.



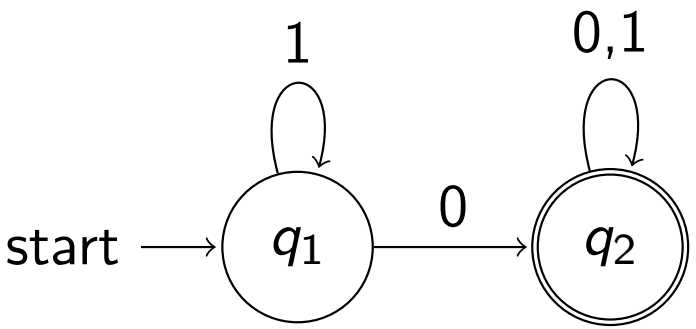
The following regular expression will recognize $L(A)$

$$R(1, j_1, n) + ... + R(1, j_s, n)$$

Example : DFA to RE

Example 7.1

Consider the following DFA



$$\left.\begin{array}{rl} R(1,1,0) = & \epsilon + 1 \\ R(2,2,0) = & \epsilon + 0 + 1 \\ R(1,2,0) = & 0 \\ R(2,1,0) = & \emptyset \end{array}\right\} \text{Base cases}$$

$$\begin{array}{rcccl} R(1,2,1) = & R(1,2,0) & + & R(1,1,0)R(1,1,0)^*R(1,2,0) & \\ = & 0 & + & (\epsilon+1)(\epsilon+1)^*0 & = \quad 1^*0 \end{array}$$

$$\begin{array}{rcccl} R(2,2,1) = & R(2,2,0) & + & R(2,1,0)R(1,1,0)^*R(1,2,0) & \\ = & \epsilon + 0 + 1 & + & \emptyset(\epsilon+1)^*0 & = \quad \epsilon + 0 + 1 \end{array}$$

Example : DFA to RE(contd.)

$$
\begin{aligned}
L(A) &= R(1,2,2) \\
&= R(1,2,1) + R(1,2,1)R(2,2,1)^*R(2,2,1) \\
&= 1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1) \\
&= 1^*0(0+1)^*
\end{aligned}
$$

Exercise 7.2

Give the following expressions $R(1,1,1) =$

$R(2,1,1) =$

Topic 7.3

Regular expressions to ϵ-NFA

Regular expressions to ϵ-NFA

Theorem 7.2

For a given RE R, there is an ϵ-NFA A such that $L(R) = L(A)$.

Proof.

We have six constructors for RE.

Three of them are base cases

- ϵ
- $\emptyset$
- a

The other three are inductive

- $R_1 + R_2$
- $R_1 R_2$
- R^*

...

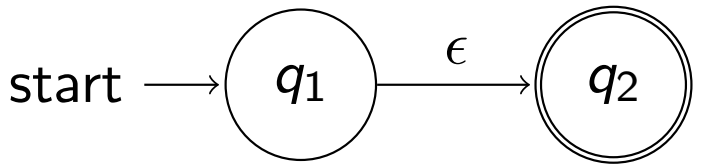
Regular expressions to NFA II

Proof(contd.)

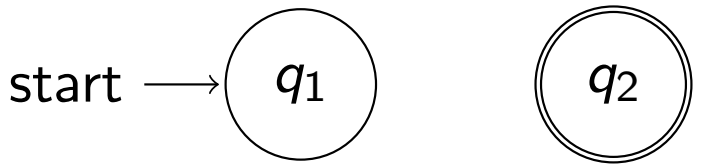
base case:

ϵ-NFAs for the base cases

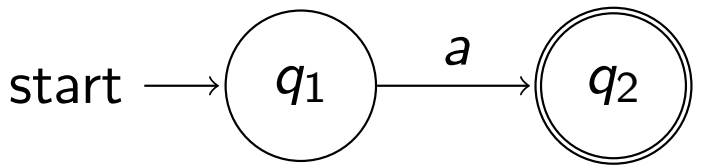
- ϵ



- $\emptyset$



- a



...

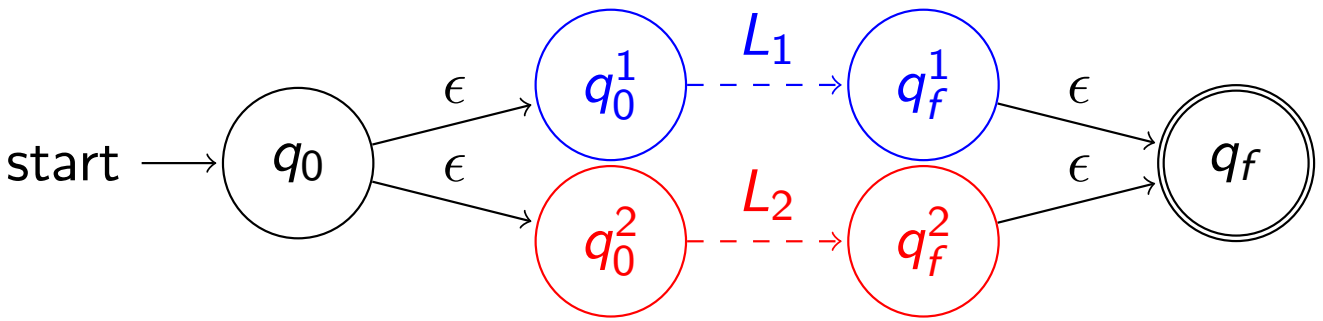
Regular expressions to NFA III

Proof(contd.)

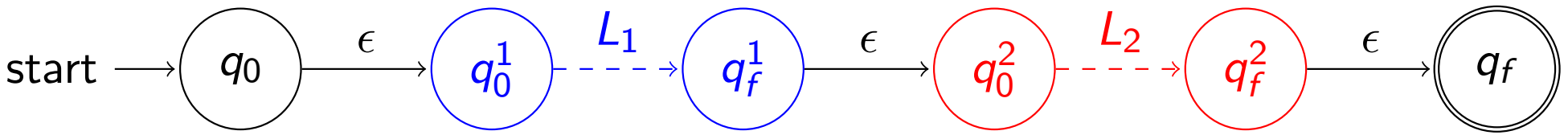
induction step:

ϵ-NFAs for the inductive constructors

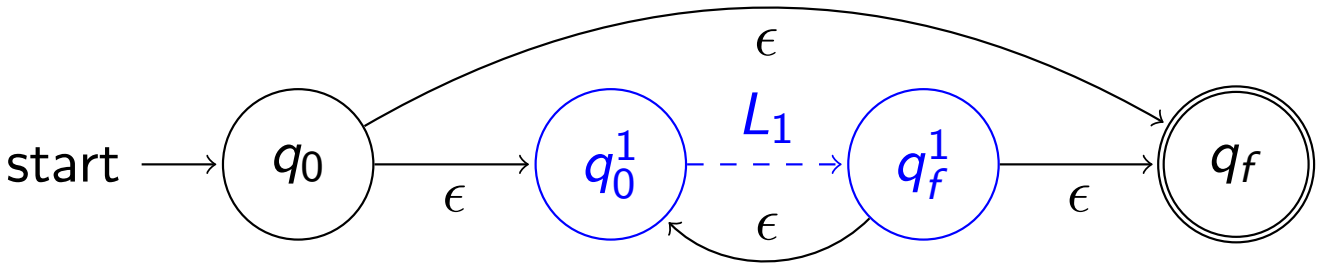
- $L_1 + L_2$



- $L_1 L_2$



- ${L_1}^*$



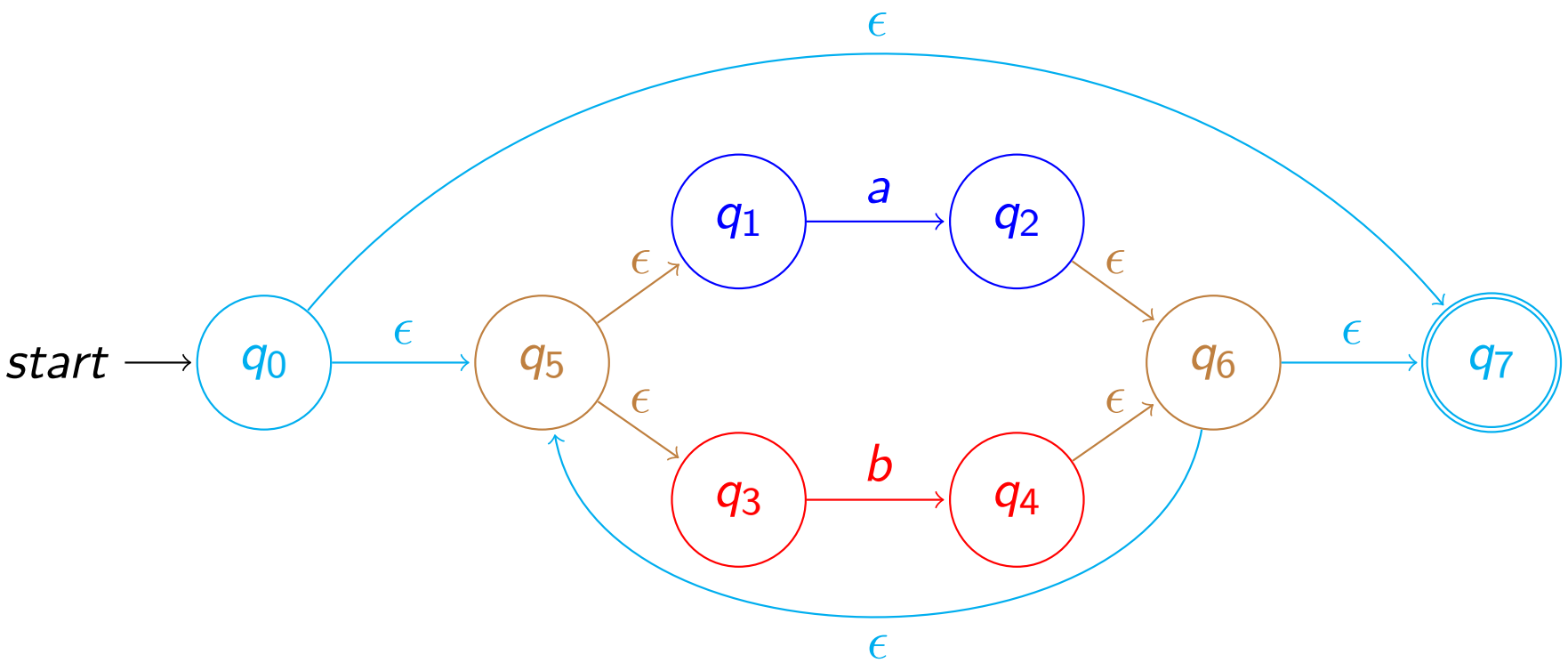
Example: RE to ϵ-NFA

Example 7.2

Consider regular expression

$$(a+b)^*$$

The following is the equivalent ϵ-NFA.



Complexity of the translation!

Exercise 7.3

What is the worst case size of ϵ-NFA for an RE with n inductive operators?

Too many ϵ-transitions are introduced!

We can easily remove them using the earlier algorithms.

Matching algorithm for regular expressions

Let us simplify the problem.

Let us match the longest prefix of a string against a given RE.

We can obtain NFA from RE without blowup.

Various implementations proceed different ways

1. DFA construction
2. NFA backtracking algorithm
3. In flight DFA (Thomson's method)

Method 1 : DFA construction

Construct DFA and run.

Upfront blowup, but matching is faster

Tools like, Awk use this.

Method 2 : NFA backtracking algorithm

Backtracking algorithm.

- Run the NFA on the word
- Keep record of the backtracking points at each nondeterministic choice
- If the current run finishes with a match or not, backtrack to the latest nondeterministic choice.

Potential exponential backtracking.

Modern languages like default Python and Perl use this method.

Method 3 : In flight DFA (Thomson's method)

Compute the extended transition as we read inputs.

$$\hat{\delta}(q_0, \epsilon) \triangleq ECLOSE(\{q\})$$
$$\hat{\delta}(q_0, wa) \triangleq \bigcup_{q' \in \hat{\delta}(q_0, w)} ECLOSE(\delta(q', a))$$

Significantly, faster than the other methods. However, implementations become complex.

Exercise 7.4

How to obtain longest match?

Exercise 7.5

Extend the above method for unanchored matching, i.e., longest match that can start from anywhere.

End of Lecture 7