

# Introduction to Computability Theory

## Lecture3: Regular Expressions

Lecture Slide Reference: [UC San Diego](#)

# Introduction

**Regular languages** are defined and described by use of **finite automata**.

In this lecture, we introduce **Regular Expressions** as an equivalent way, yet more elegant, to describe regular languages.

# Motivation

If one wants to describe a regular language,  $La$ , she can use the a DFA,  $D$  or an NFA  $N$ , such that that  $L(D) = La$  .

This is not always very convenient.

Consider for example the regular expression  $0^*10^*$  describing the language of binary strings containing a single 1.

# Basic Regular Expressions

A ***Regular Expression*** (RE in short) is a string of symbols that describes a **regular Language**.

- Let  $\Sigma$  be an alphabet. For each  $\sigma \in \Sigma$ , the symbol  $\sigma$  is an RE representing the set  $\{\sigma\}$ .
- The symbol  $\varepsilon$  is an RE representing the set  $\{\varepsilon\}$ . (The set containing the empty string).
- The symbol  $\phi$  is an RE representing the empty set.

# Inductive Construction

Let  $R_1$  and  $R_2$  be two regular expressions representing languages  $L_1$  and  $L_2$  , resp.

- The string  $(R_1 \cup R_2)$  is a regular expression representing the set  $L_1 \cup L_2$  .
- The string  $(R_1 R_2)$  is a regular expression representing the set  $L_1 \circ L_2$  .
- The string  $(R_1)^*$  is a regular expression representing the set  $L_1^*$  .

# Inductive Construction - Remark

Note that in the inductive part of the definition larger RE-s are defined by smaller ones. This ensures that the definition is not **circular**.

This inductive definition also dictates the way we will prove theorems:

**Stage 1:** Prove  $Te$  correct for all base cases.

**Stage 2:** Assume  $Te$  is correct for  $R_1$  and  $R_2$  and prove its correctness for  $(R_1 \cup R_2)$   $(R_1 R_2)$   $(R_1)^*$

# Some Useful Notation

Let  $R$  be a regular expression:

- The string  $R^+$  represents  $RR^*$ , and it also holds that  $R^+ \cup \{\varepsilon\} = R^*$ .
- The string  $R^k$  represents  $\underbrace{RR\dots R}_{k \text{ times}}$ .
- The string  $\Sigma$  represents  $\{\sigma_1, \sigma_1\dots, \sigma_k\}$ .
- The Language represented by  $R$  is denoted by  $L(R)$ .

# Precedence Rules

- The star ( $*$ ) operation has the highest precedence.
- The concatenation ( $\circ$ ) operation is second on the preference order.
- The union ( $\cup$ ) operation is the least preferred.
- Parentheses can be omitted using these rules.



# Examples

- $0^*10^*$  –  $\{w \mid w \text{ contains a single } 1\}$  .
- $\Sigma^*1\Sigma^*$  –  $\{w \mid w \text{ has at least a single } 1\}$  .
- $\Sigma^*(str)\Sigma^*$  –  $\{w \mid w \text{ contains } str \text{ as a substring}\}$  .
- $1^*(01^+)^*$  –  $\left\{ w \mid \begin{array}{l} \text{every } 0 \text{ in } w \text{ is followed} \\ \text{by at least a single } 1 \end{array} \right\}$  .
- $(\Sigma\Sigma)^*$  –  $\{w \mid w \text{ is of even length}\}$  .

# Examples

- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$  - all words starting and ending with the same letter.
- $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$  - all strings of forms 1,11, 111... and 0,01,011....
- $R\phi = \phi$  - A set concatenated with the empty set yields the empty set .
- $\phi^*$  -  $\phi^* = \{\varepsilon\}$  .

# Equivalence With Finite Automata

Regular expressions and finite automata are equivalent in their descriptive power. This fact is expressed in the following Theorem:

## Theorem

A set is regular **if and only if** it can be described by a regular expression.

The proof is by two Lemmata (Lemmas):

## Lemma ->

If a language  $L$  can be described by regular expression then  $L$  is regular.

# Proofs Using Inductive Definition

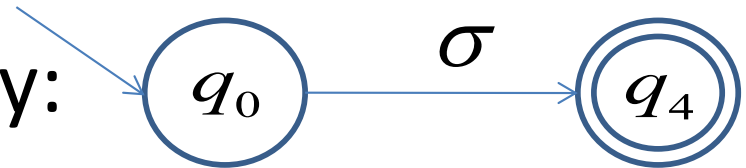
This inductive definition of Regular expressions dictates the way we will prove theorems. The proof for the Theorem follows the following stages:

**Stage 1:** Prove correctness for all base cases.

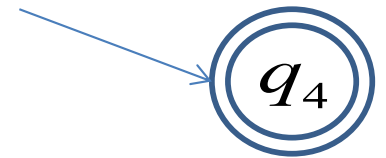
**Stage 2:** Assume correctness for  $R_1$  and  $R_2$ , and show its correctness for  $(R_1 \cup R_2)$ ,  $(R_1 R_2)$  and  $(R_1)^*$ .

# Induction Basis

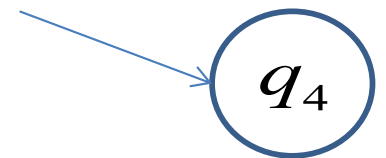
1. For any  $\sigma \in \Sigma$ , the expression  $\sigma$  describes the set  $\{\sigma\}$ , recognized by:



2. The set represented by the expression  $\varepsilon$  is recognized by:



3. The set represented by the expression  $\phi$  is recognized by:



# The Induction Step

Now, we assume that  $R_1$  and  $R_2$  represent two regular sets and claim that  $R_1 \cup R_2$ ,  $R_1 \circ R_2$  and  $R_1^*$  represent the corresponding regular sets.

The proof for this claim is straight forward using the constructions given in the proof for the closure of the three regular operations.

# Examples

Show that the following regular expressions represent regular languages:

1.  $(ab \cup a)^*$  .
2.  $(a \cup b)^* aba$  .



## Lemma <-

If a language  $L$  is regular then  $L$  can be described by regular expression.

# Proof Stages

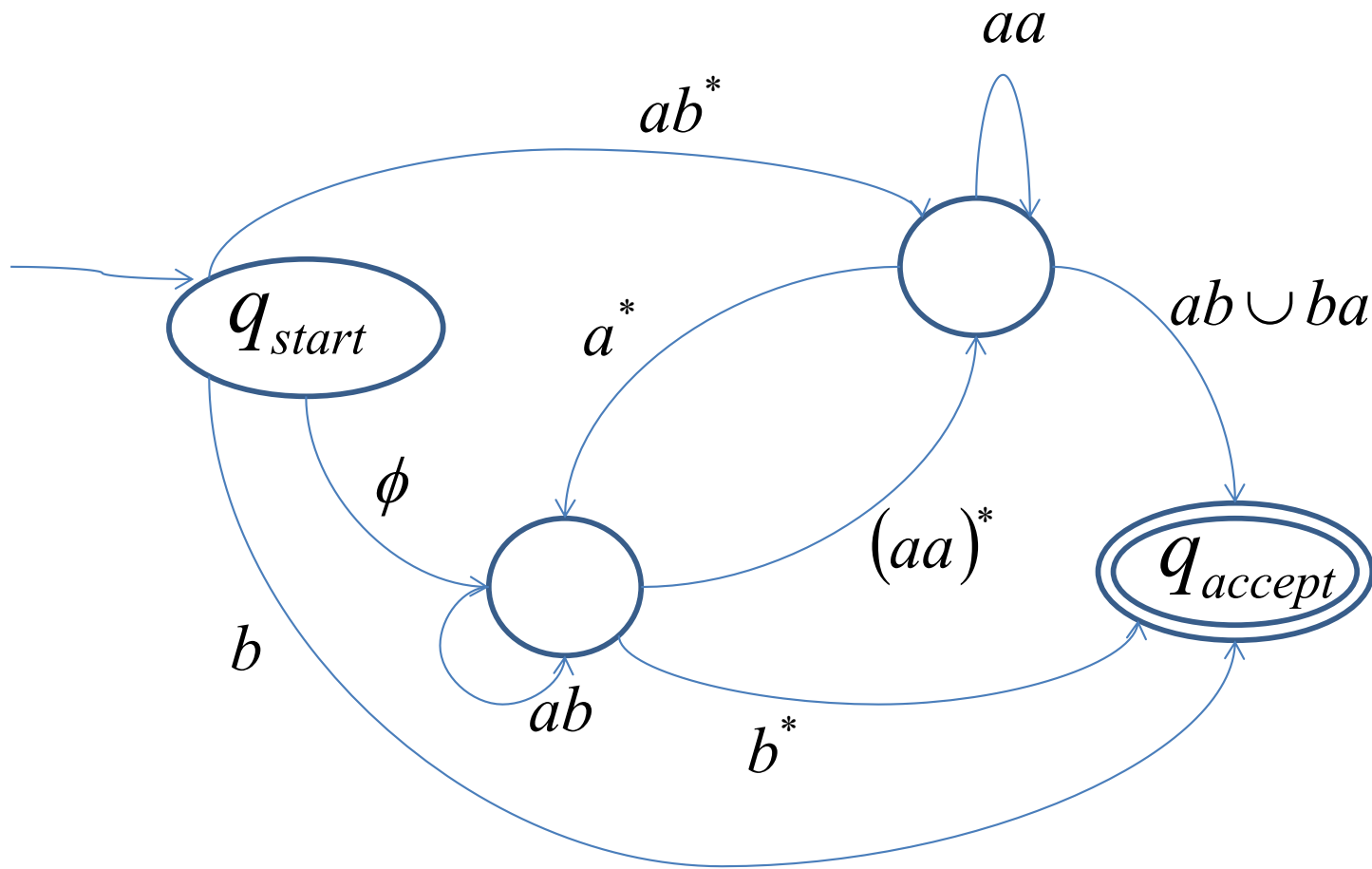
The proof follows the following stages:

1. Define Generalized Nondeterministic Finite Automaton (GNFA in short).
2. Show how to convert any DFA to an equivalent GNFA.
3. Show an algorithm to convert any GNFA to an equivalent GNFA with 2 states.
4. Convert a 2-state GNFA to an equivalent RE.

# Properties of a Generalized NFA

1. A GNFA is a finite automaton in which each transition is labeled with a regular expression over the alphabet  $\Sigma$ .
2. A single **initial state** with all possible outgoing transitions and no incoming trans.
3. A single **final state** without outgoing trans.
4. A single transition between every two states, including self loops.

# Example of a Generalized NFA



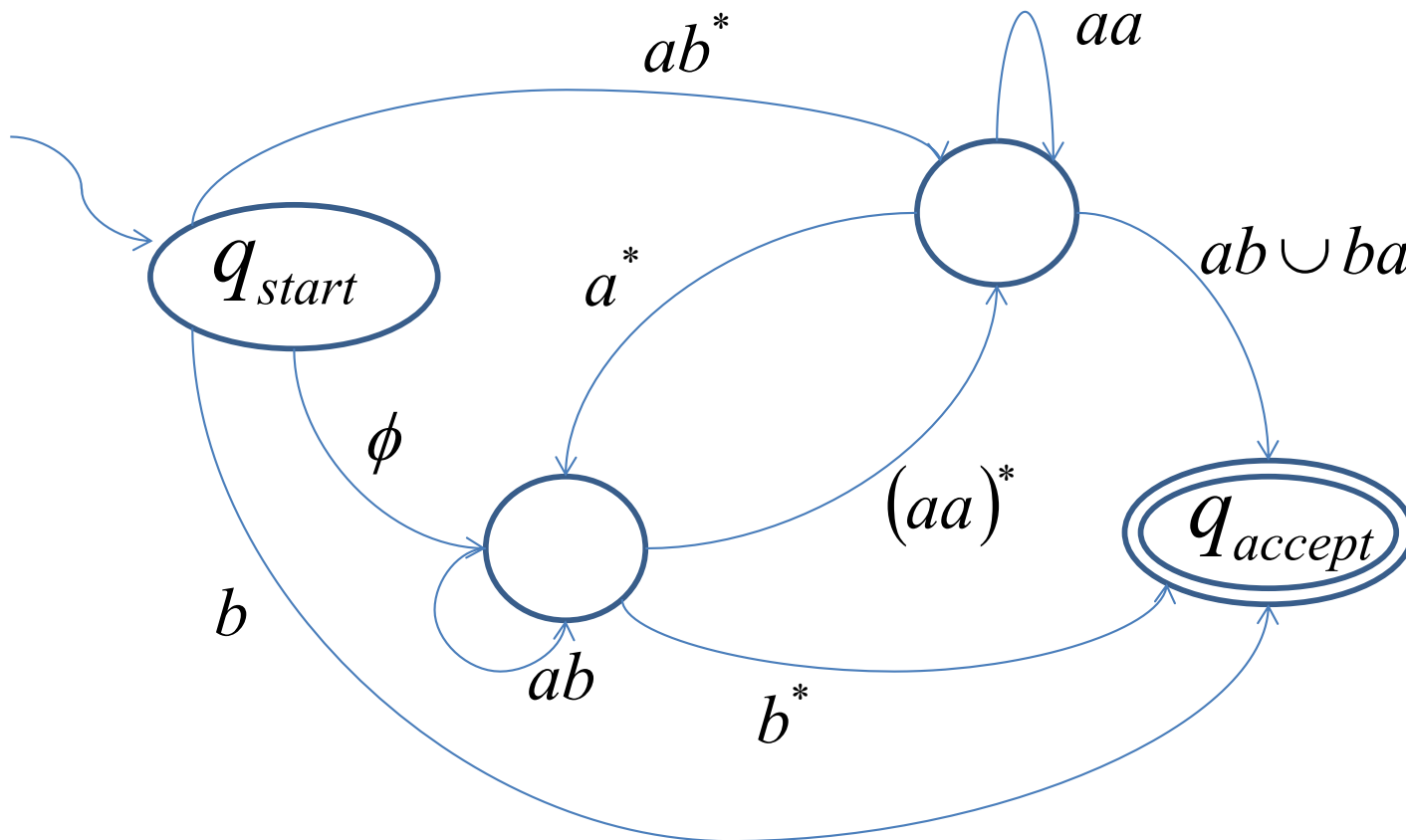
# A Computation of a GNFA

A *computation* of a GNFA is similar to a computation of an NFA. except:

In each step, a GNFA consumes *a block of symbols* that matches the RE on the transition used by the NFA.

# Example of a GNFA Computation

Consider  $abbbbaaaaaabbbbbb$  or  $bb$  or  $abba$



# Converting a DFA to a GNFA

Conversion is done by a very simple process:

1. Add a new start state with an  $\varepsilon$  - transition from the **new** start state to the **old** start state.
2. Add a new accepting state with  $\varepsilon$ - transition from every **old** accepting state to the **new** accepting state.

## Converting a DFA to a GNFA (Cont)

4. Replace any transition with multiple labels by a single transition labeled with the ***union*** of all labels.
5. Add any missing transition, including self transitions; label the added transition by  $\phi$ .

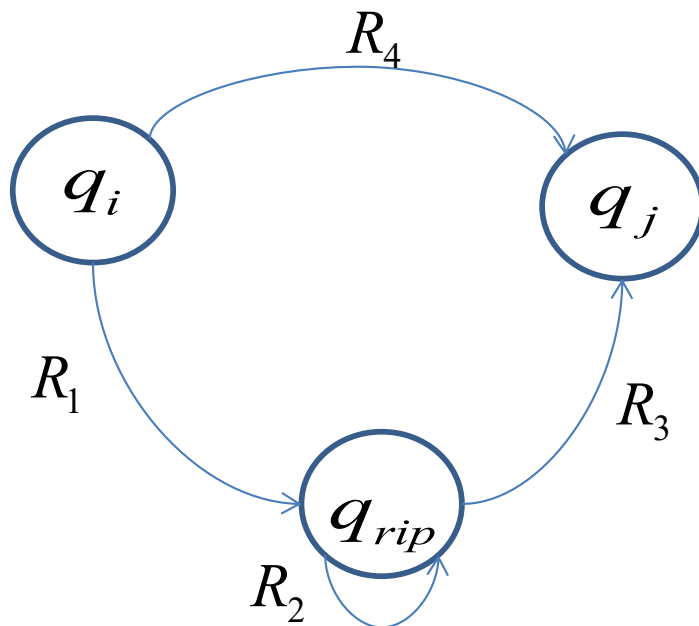


# Ripping a state from a GNFA

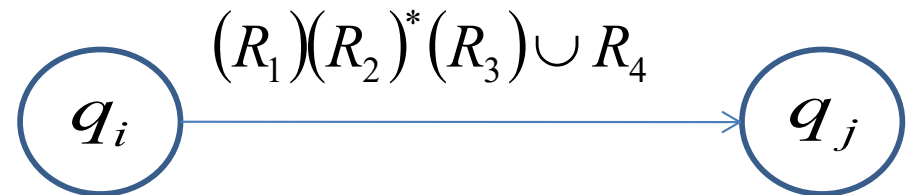
The final element needed for the proof is a procedure in which for any GFN  $G$ , any state of  $G$ , not including  $q_{start}$  and  $q_{accept}$ , can be **ripped** off  $G$ , while preserving  $L(G)$ . This is demonstrated in the next slide by considering a general state, denoted by  $q_{rip}$ , and an arbitrary pair of states,  $q_i$  and  $q_j$ , as demonstrated in the next slide:

# Removing a state from a GNFA

## Before Ripping



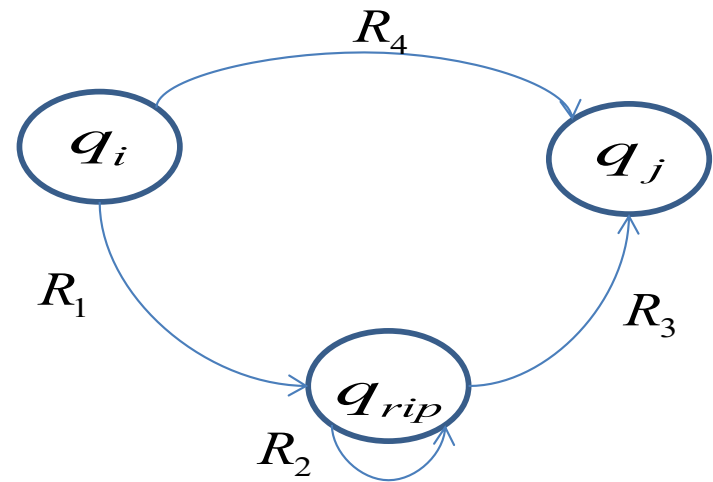
## After Ripping



**Note:** This should be done for **every pair** of outgoing and incoming outgoing  $q_{rip}$ .

# Elaboration

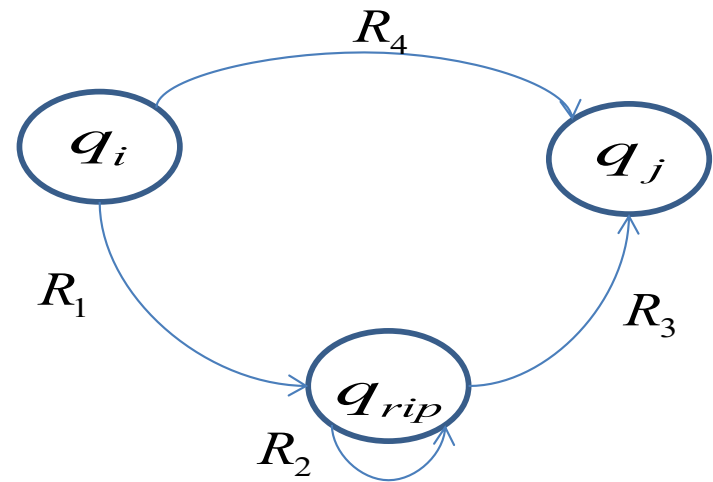
Consider the RE  $(R_1)(R_2)^*R_3$ ,  
representing all strings  
that enable transition  
from  $q_i$  via  $q_{rip}$  to  $q_j$ .



What we want to do is to augment the Regular  
expression of transition  $(q_i, q_j)$ , namely  $R_4$ , so  
These strings can pass through  $(q_i, q_j)$ . This is  
done by setting it to  $R_4 \cup (R_1)(R_2)^*(R_3)_4$ .

# Elaboration

Note that this change does not affect all pairs in which either  $(q_i, q_{rip})$  or  $(q_{rip}, q_j)$  participate.



Thus, before  $q_{rip}$  is removed all these pairs should be processed in the same way, as demonstrated on the next slide:

# Elaboration

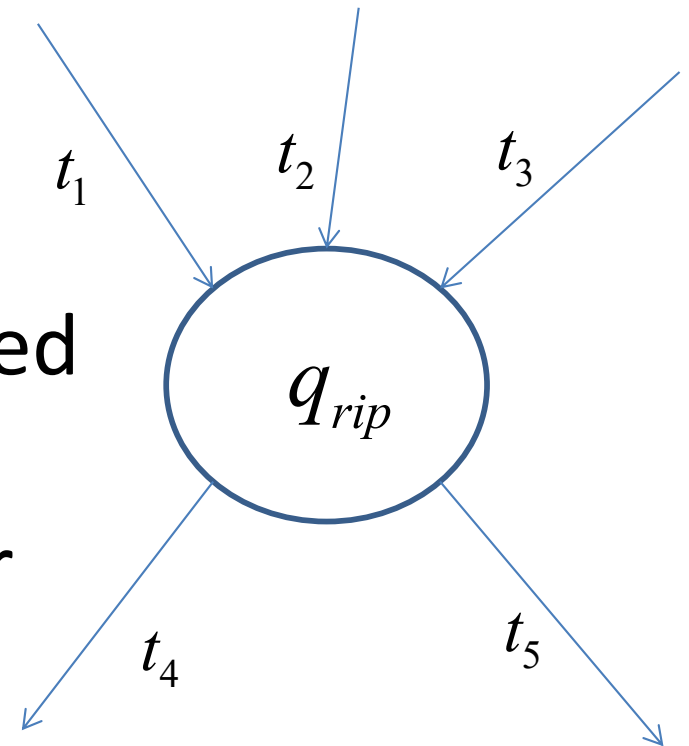
Assume the following situation:

In order to rip  $q_{rip}$ , all pairs of incoming and outgoing transitions should be considered in the way showed on the

previous slide namely consider

$(t_1, t_4), (t_1, t_5), (t_2, t_4), (t_2, t_5), (t_3, t_4),$

$(t_3, t_5)$  one after the other. After that  $q_{rip}$  can be ripped while preserving  $L(G)$ .



# A (half?) Formal Proof of Lemma<-

The first step is to formally define a GNFA.  
Each transition should be labeled with an RE.  
Define the transition function as follows:

$$\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow RE_{\Sigma}$$

where  $RE_{\Sigma}$  denotes all regular expressions over  $\Sigma$  .

**Note:** The def. of  $\delta$  is different then for NFA.

# Changes in $\delta$ Definition

**Note:** The definition of  $\delta$  as:

$$\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow RE_{\Sigma}$$

is different than the original definitions (For DFA and NFA).

In this definition we rely on the fact that every 2 states (except  $q_{start}$  and  $q_{accept}$ ) are connected in both directions.

# GNFA – A Formal Definition

A **Generalized Finite Automaton** is a 5-tuple  $(Q, \Sigma, \delta, q_{start}, q_{accept})$  where:

1.  $Q$  is a finite set called the **states**.
2.  $\Sigma$  is a finite set called the **alphabet**.
3.  $\delta : (Q - \{q_{accept}^*\}) \times (Q - \{q_{start}\}) \rightarrow RE_{\Sigma}$  is the **transition function**.
4.  $q_{start} \in Q$  is the **start state**, and
5.  $q_{accept} \subseteq Q$  is the **accept state**.



# GNFA – Defining a Computation

A GNFA ***accepts*** a string  $w \in \Sigma^*$  if  $w = w_1 w_2 \cdots w_k$  and there exists a sequence of states

$q_{start} q_1 q_2 \cdots q_{accept}$ , satisfying:

For each  $i$ ,  $1 \leq i \leq k$ ,  $w_i \in L(R_i)$ , where  $R_i = \delta(q_{i-1}, q_i)$ , or in other words,  $R_i$  is the expression on the arrow from  $q_{i-1}$  to  $q_i$ .

# Procedure *CONVERT*

Procedure *CONVERT* takes as input a GNFA  $G$  with  $k$  states.

If  $k = 2$  then these 2 states must be  $q_{start}$  and  $q_{accept}$ , and the algorithm returns  $\delta(q_{start}, q_{accept})$ .

If  $k > 2$ , the algorithm converts  $G$  to an equivalent  $G'$  with  $k - 1$  states by use of the ripping procedure described before.

# Procedure *CONVERT*

*Convert* ( $G$ )

1.  $k \leftarrow |Q_G|$  ;
2. If ( $k = 2$ ) return  $\delta(q_{start}, q_{accept})$  ;
3.  $q_{rip} \leftarrow GetRandomState(Q_G)$  ;
4.  $Q' \leftarrow Q_G - q_{rip}$  ;
5. For any  $q_i \in Q' - q_{accept}$  and any  $q_j \in Q' - q_{start}$   
 $\delta'(q_i, q_j) \leftarrow (R_1)(R_2)^*(R_3) \cup R_4$   
for  $R_1 = \delta(q_i, q_{rip})$   $R_2 = \delta(q_{rip}, q_{rip})$   $R_3 = \delta(q_{rip}, q_i)$   $R_4 = \delta(q_i, q_j)$   
return  $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$  ;

# Recap

In this lecture we:

1. Motivated and defined regular expressions as a more concise and elegant method to represent **regular Languages**.
2. Proved that FA-s (Deterministic as well as Nondeterministic) and RE-s is identical by:
  - 2.1 Defined GNFA – s.
  - 2.2 Showed how to convert a DFA to a GNFA.
  - 2.3 Showed an algorithm to converted a GNFA with  $K$  states to an equivalent GNFA with  $K - 1$  states.