

How to unify the arc cases:

Type	Common Generalization
Char arc: $p \xrightarrow{c} q$ <i>Resistance "c"</i>	$Q \times \text{char} \times Q$
$\epsilon$ -arc: $p \xrightarrow{\epsilon} q$ <i>No Resistance</i>	$Q \times \text{string} \times Q$
No arc: $p \xrightarrow{\emptyset} q$ <i>Total Resistance</i>	$Q \times \text{zipper??} \times Q$

~~Basic~~  
 $Q \times \text{Regex}(\Sigma) \times Q$

To some extent:  $Q \xrightarrow{0,1} \Rightarrow$

Intuition:

Def<sup>n</sup>: A generalized NFA (GNFA)  $N = (Q, \Sigma, \delta, s, F)$  has  $\delta \subseteq Q \times \text{Regex}(\Sigma) \times Q$ .

A GNFA can process a (sub-)string  $y$  of length  $n$  from state  $p$  to state  $q$  if there is a sequence  $(q_0, u_1, q_1, u_2, q_2, \dots, q_{m-1}, u_m, q_m)$  such that  $u_1 \dots u_m = y$ ,  $q_0 = p, q_m = q$ , and for each  $i, 1 \leq i \leq m$ , there is an instruction  $(q_{i-1}, \alpha, q_i)$  in  $\delta$  such that  $u_i$  matches  $\alpha$ .

As before,  $L_{pq}$  is the set of such  $y$  and  $L(N) = \bigcup_{p \in F, s \in F} L_{sf}$ . i.e.  $\alpha$  is a regex and  $u_i \in L(\alpha)$ .

Trivial example:  $N = \begin{matrix} s & \xrightarrow{\alpha} & f \end{matrix}$ . Then  $L(N) = L(\alpha)$ . These were base cases of our proof.

$L_{sg} =$

$L_{sg}$  also

$L_{sq}$

$L_{qs}$

Intuition:  $L_{ss}$  means zero or more of

- taking a "pit stop" at state  $s$  with  $\alpha$  or
- going down the racetrack on  $\beta$  spinning around  $\eta$  or more times with  $\gamma$  and coming back via  $\eta$ .

$(\alpha + \beta \cdot \eta^* \cdot \gamma)^*$

In the latter option,  $\beta$  and  $\eta$  are required. If either is  $\emptyset$  (e.g. impossible because the track is blocked),  $\beta \gamma \eta^*$  cannot happen. But if  $\gamma = \emptyset, \gamma^* = \emptyset = \epsilon$  which still allows going down and coming right+back.

and then

$L_{sg} = L_{ss} \cdot \beta \gamma^*$

$L_{sg}$  also =  $\alpha \cdot \beta \cdot L_{sg}$

$L_{sq} = (\gamma + \eta \alpha^* \beta)^*$

$L_{qs}$  = exercise.

If  $q \notin F, s \notin F$ , then  $L(N) = L_{sq}$ .

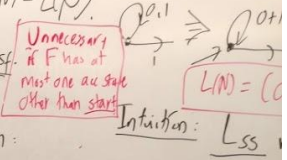
If both are accepting,  $L(N) = L_{ss} \cup L_{sq}$   
 $= L_{ss} \cup L_{ss} \beta \gamma^* = L_{ss} \cdot (\epsilon + \beta \gamma^*) = (\alpha + \beta \gamma^*) (\epsilon + \beta \gamma^*)$

$Q \xrightarrow{0,1} \Rightarrow$

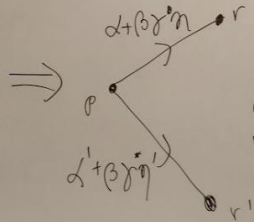
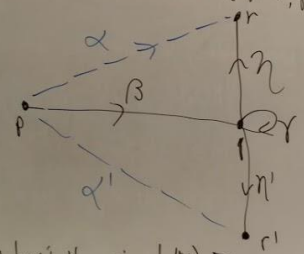
$Q \xrightarrow{\alpha} s \xrightarrow{\beta} q \xrightarrow{\gamma} s \xrightarrow{\eta} s$

Theorem: Given any GNFA  $N$ , we can compute a regular expression  $\rho$  (rho) s.t.  $L(N) = L(\rho)$ . To some extent w/d.

Proof: Add a new final state  $f$  with  $\epsilon$ -arcs from all old final states, re-make  $F = \{f\}$ , compute  $L_{sf}$ .  
 Now all states  $q \neq s$  in  $N$  are nonaccepting states. Hence any processing that enters  $q$  via some state  $p$  must eventually leave via some state  $r$  ( $r=p$  allowed) in order eventually to end at  $f$ . We can:

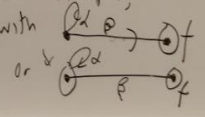


bypass the in-edge  $(p, \beta, q)$  and the out-edge  $(q, \eta, r)$  along with  $(q, \gamma, q)$  by creating  $(p, \alpha + \beta \eta \gamma, r)$



When we have bypassed all (in-from  $q$ , out-to- $r$ ) combinations, we can delete  $q$ .

When we have deleted all  $q \neq s$ , we are left with



The basis then gives  $L(N)$ .  $\square$

$L_{sf} = L_{ss}$

'-sq also =  $\alpha$ '

$L_{sf} = ($

$L_{ss} = \epsilon \alpha$