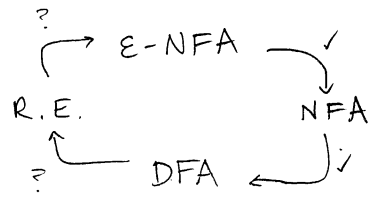
Lecture 5 1/19/11 reading H-M-U 3.3

Definition: Languages generated by regular expressions are called regular languages

Main Theorem: Regular languages are exactly the languages decided by DFAs (or NFAs, equivalently).



Theorem (R.E. $\longrightarrow$ ε-NFA):

For any regular language L, there exists an ε-NFA N such that $L = L(N)$.

Proof

Assume: $L = L(r)$ for a regular expression r

Goal: Construct ε-NFA N_r with $L(N_r) = L(r)$.

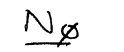
Proof by structural induction, ie., induction on # of operators in r.

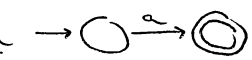
Examples: $r = (a+b)^*c$ $\quad \#r = 3 = \#s + \#t + 1$

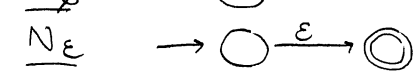
$s = (a+b)^* \quad \#s = 2$

$t = c \quad \#t = 0$

Base case: $\#r = 0 \Rightarrow r = \varepsilon$, $r = \emptyset$ or $r = a$ for an $a \in \Sigma$

$N_\emptyset$ 

N_a 

N_ε 

Induction step:

Assume that for all r with $\#r \leq n$, there is an ε-NFA N_r with $L(N_r) = L(r)$.

Consider an r with $\#r = n+1$.

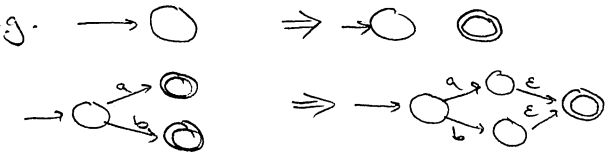
$\Rightarrow$ 3 cases $\begin{cases} r = s+t \\ r = s \cdot t \\ r = s^* \end{cases}$

in each case $\#s, \#t \leq n$

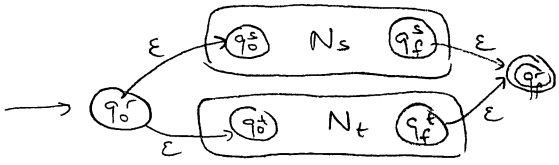
By the induction hypothesis, there exist ε-NFAs N_s, N_t with $L(N_s) = L(s)$, $L(N_t) = L(t)$.

Furthermore, we may assume N_s and N_t each have exactly one accepting state! [Convenient for our pictures below]

Why? Otherwise, add a new accepting state, add ε-transitions from all the other accepting states to it, and then make them non-accepting.

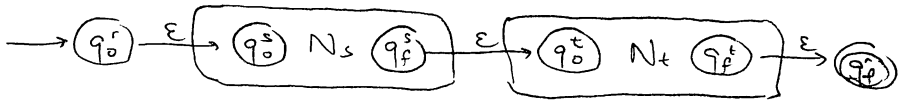
Eg. 

Case I : $r = s + t$

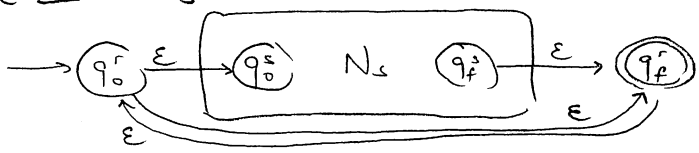


$$L(N_r) = L(N_s) \cup L(N_t)$$
$$= L(s) \cup L(t)$$
$$= L(s+t) = L(r) \quad \checkmark$$

Case II : $r = s \cdot t$



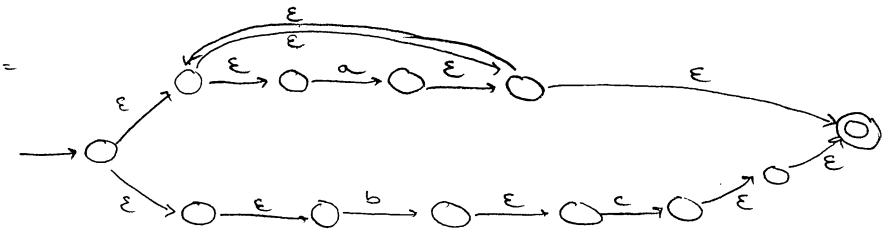
Case III : $r = s^*$



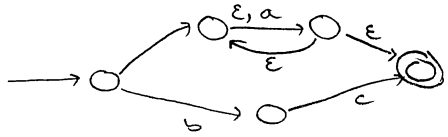
$\checkmark$ $\square$

Example $r = a^* + b \cdot c$ $(\#r = 3)$

$N_r =$



which simplifies to



(but not further!)

Converting DFAs to Regular Expressions (see H-M-U 3.2.1-3.2.2)

Theorem Let M be a DFA.
Then $L(M)$ is regular.

Proof idea Given $M = (Q, \Sigma, \delta, q_0, F)$
Construct regexp. r such that $L(r) = L(M)$.

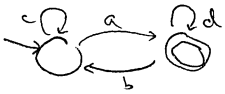
Two proofs in the text: (and they work directly on NFAs and ε-NFAs)

1. By adding states in one at a time [3.2.1]
2. By eliminating states from M one at a time [3.2.2]

State Elimination proof

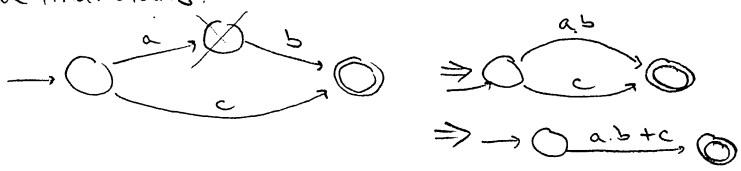
Idea: Start with an NFA N.

If N has ≤ 2 states, it is easy!

e.g. $N =$ 

$\Rightarrow r_N = (c^* a d^* b)^* c^* a d^*$
or $r_N = (c + a d^* b)^* a d^*$

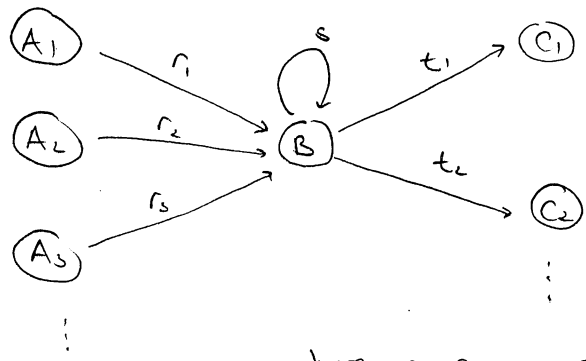
Try to simplify to this case by eliminating states between the initial and final states.

Example 

This creates generalized automata with edges labeled by regular expressions.

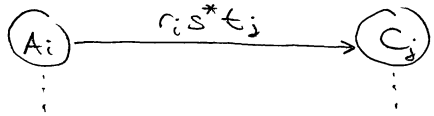
→

In general, it will look like...

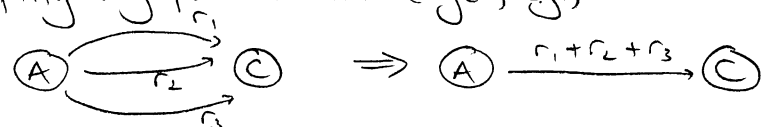


where $r_1, r_2, \ldots, s, t_1, t_2, \ldots$ are all regexps.
It may be that some A_i and C_j states are the same!

Now to eliminate B, add edges from each A_i to each C_j as:

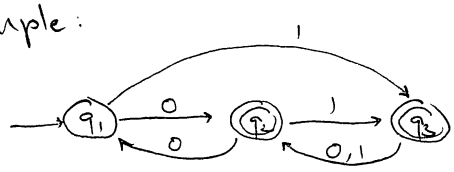


Then simplify any possible multi-edges, e.g.,

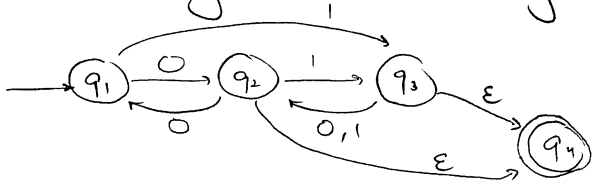


Idea: At each step of simplification, the arrow from state q to state q' should be labeled by a regexp. representing all possible ways of going from q to q' in the original NFA that only go through eliminated states.

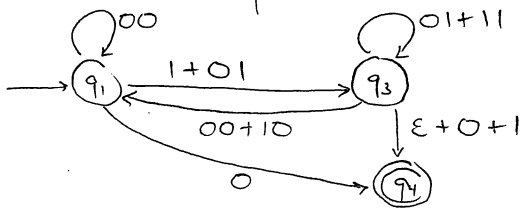
Example:



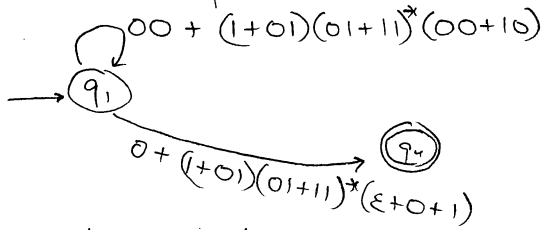
- First, change it so there is only one accepting state



- Now eliminate q_2:



- Now eliminate q_3:



- With only two states remaining, we can read off the answer:

$$r = \Big(00 + (1+01)(01+11)^*(00+10)\Big)^*\Big(0 + (1+01)(01+11)^*(\varepsilon+0+1)\Big)$$

Note: The answer you get will depend on the order in which you eliminate states. In exercises, show your work!
It is a good idea to eliminate less-connected states first.

This is not yet a proof, but you should convince yourself that the idea works. A formal proof is in H-M-U 3.2.1. □

⇒ DFAs and NFAs all accept, and regexps generate exactly the set of regular languages!

Examples:

- $L = \{ w \in \{0,1\}^* \mid n_0(w) \text{ is } \underline{\text{odd}} \}$

$1^*01^*(01^*0)^*1^*$ ✗

$1^*01^*(01^*01^*)^*1^*$ ✓

$1^*0(1^*01^*01^*)^*$ ✗

$1^*0(1^*01^*0)^*1^*$ ✓

$1^*0(1+01^*0)^*$ ✓

$(1+01^*0)^*01^*$ ✓

- $L = \{ w \in \{0,1\}^* \mid w \text{ ends with } 1 \text{ \& does not contain } 00 \}$

$$(1+01)^*(1+01)$$

- $L = \{ w \in \{0,1\}^* \mid n_0(w), n_1(w) \text{ are both even} \}$

$$\Big(00+11+(01+10)(00+11)^*(01+10)\Big)^*$$

- look at a prefix, find the shortest one that balances the parities of n_0 and n_1.

Example: Convert to a regexp: