

# Lecture 4: Equivalence of Regular Expressions and Finite Automata

David Dill

Department of Computer Science

# Outline

---

- Converting Regular Expressions to  $\epsilon$ -NFAs
- Converting NFAs to regular expressions.

# From Regular Expressions to Finite Automata

**Theorem:** For every regular expression  $R$ , there is an  $\epsilon$ -NFA  $E$  such that  $L(R) = L(E)$ .

It is easier to prove a stronger theorem:

**Theorem:** For every regular expression  $R$ , there is an  $\epsilon$ -NFA  $E$  with these properties:

- $L(E) = L(R)$ .
- There is a single final state.
- No transitions enter the start state.
- No transitions leave the final state.

The theorem is proved by induction on the structure of regular expressions.

(A fully detailed proof would have a lot of tedious manipulation of  $\hat{\delta}$ s)

## Proof of $R \rightarrow \epsilon\text{-NFA}$

The proof is by induction on the structure of regular expressions.

- When  $R = \emptyset$ , the NFA is:

⟨ See figure in text. ⟩

All four properties are obviously met.

- When  $R = \epsilon$ , the NFA is

⟨ See figure in text. ⟩

All four properties are obviously met.

## Proof of $R \rightarrow \epsilon$ -NFA, cont.

- When  $R = \mathbf{a}$ , the NFA is  
〈 See figure in text. 〉

All four properties are obviously met.

- When  $R = R_1 + R_2$ , the NFA is:  
〈 See figure in text. 〉

Assuming the induction hypothesis for  $R_1$  and  $R_2$ , all four conditions are obviously met.

## Proof of $R \rightarrow \epsilon$ -NFA, cont.

- $R_1 R_2$

⟨ See figure in text. ⟩

Assuming the induction hypothesis for  $R_1$  and  $R_2$ , all four conditions are obviously met.

- $R^*$

⟨ See figure in text. ⟩

## Example: Regular expression to $\epsilon$ -NFA

**Example:**  $(0 + 11)^*0$

## Implications

We already know that  $\epsilon$ -NFAs, NFAs, and DFAs have equal expressive power.

The latest theorem shows that FA are at least as expressive as regular expressions.

Next: Can we do anything with an FA that we cannot do with Regular Expression?

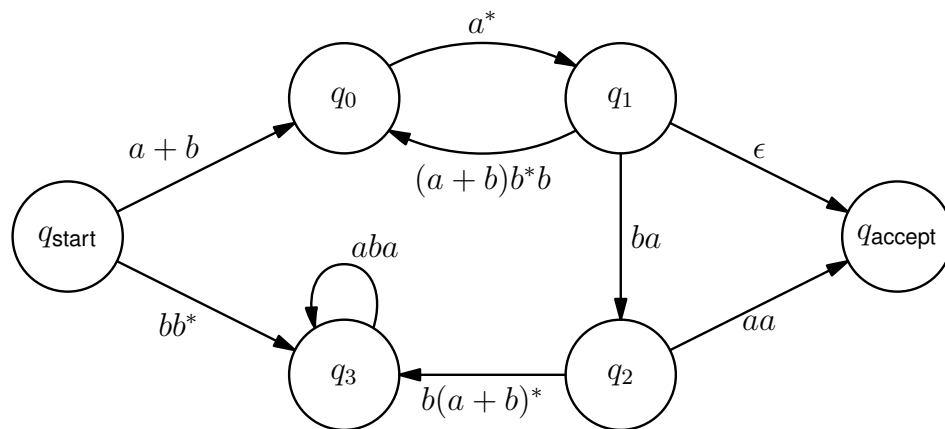
# From FA to Regular Expressions

Theorem: For every finite automaton (any type)  $A$ , there is a regular expressions  $R$  such that  $L(R) = L(A)$ .

The heart of this is a construction. I'm going to use the treatment from Sipser's book "Introduction to the Theory of Computation" because it is easier to describe in a lecture.

The method in the textbook is very similar.

First, let's define yet another type of finite automaton, just for purposes of this proof: A "Generalized Nondeterministic Finite Automaton" (GNFA).



## Formal definition of a GNFA

A GNFA is a quintuple :  $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ .

$Q$  and  $\Sigma$  are the same as other FA.

$\delta : Q \times Q \rightarrow \text{RegExps}$  where RegExps are the regular expressions over  $\Sigma$ .

Since  $\delta$  is a total function, there must be exactly one regular expression between every pair of states. However, that might be  $\emptyset$ , which I will generally omit from diagrams.

$q_{\text{start}}$  and  $q_{\text{accept}}$  are the start and accept states.

We require that  $\delta(q_i, q_{\text{start}}) = \emptyset$  for all  $q_i$ , and  $\delta(q_{\text{accept}}, q_j) = \emptyset$  for all  $q_j$  (effectively, no incoming states to start, no exits from accept).

## Acceptance by a GNFA

An *accepting run* of a GNFA is a sequence of states  $p_0, p_1, \dots, p_n$  where

- $p_0 = q_{\text{start}}$
- $p_n = q_{\text{accept}}$
- $x = w_1 w_2 \dots w_n$  where  $w_i \in L[\delta(p_{i-1}, p_i)]$  for all  $i$  (note: getting from one state to another may involve a long string).

If  $G$  is a GNFA,  $L(G) = \{x \mid \text{There is an accepting run of } G \text{ on } x\}$ .

In the previous example GNFA,  $x = w_1 w_2 w_3 w_4 w_5 w_6 = aabbaabaaa$  where  $w_1 = a$ ,  $w_2 = \epsilon$ ,  $w_3 = abb$ ,  $w_4 = aa$ ,  $w_5 = ba$  and  $w_6 = aa$  is in  $L(G)$  because of the accepting run  $q_{\text{start}} q_0 q_1 q_0 q_1 q_2 q_{\text{accept}}$ .

## From Finite automata to GNFA

It should be clear that any FA  $A$  presented so has an equivalent GNFA:

- Add new  $q_{\text{start}}$  and  $q_{\text{accept}}$  states.
- $\delta(q_{\text{start}}, q_0) = \epsilon$ .
- $\delta(q_i, q_{\text{accept}}) = \epsilon$  for each  $q_i \in F$ .
- Whenever there are multiple transitions from  $q_i$  to  $q_j$  in the original FA, merge them into a single reg exp using “+”.
- $\delta(q_i, q_j) = \emptyset$  if  $A$  has no transition from  $q_i$  to  $q_j$ .

## Eliminating states

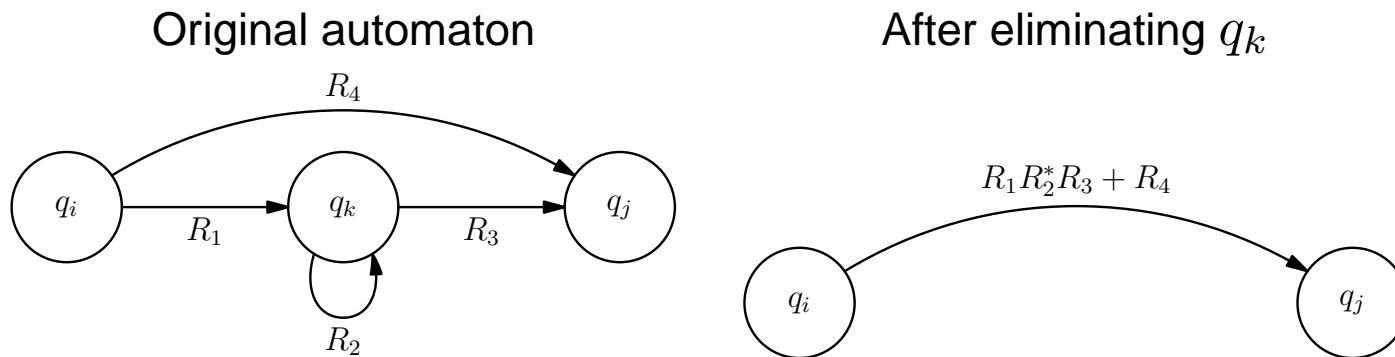
We will eliminate one state at a time, giving a smaller GNFA with more complex regular expressions until we have just  $q_{\text{start}}$  going to  $q_{\text{accept}}$  via the regular expression for the whole language.

Choose any state except one of those for elimination. Call it  $q_k$ .

Construct a new GNFA,  $G' = (Q - \{q_k\}, \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$

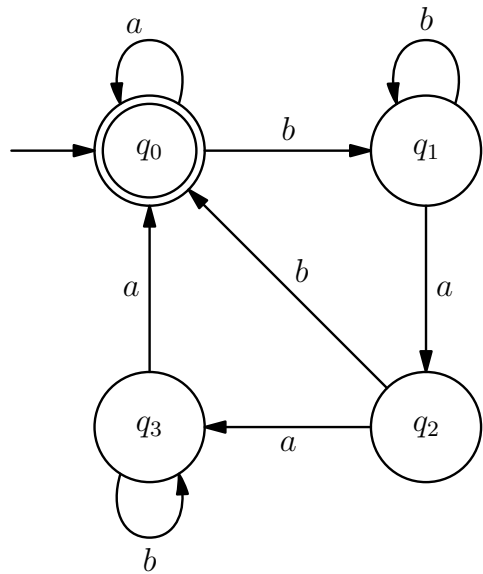
$\delta'(q_i, q_j) = \delta(q_i, q_k)\delta(q_k, q_k)^* \delta(q_k, q_j) + \delta(q_i, q_j)$  for each  $q_i, q_j \in Q$ .

[Each  $\delta(q_i, q_k)$  is a regular expression, so this is a bigger reg exp.]

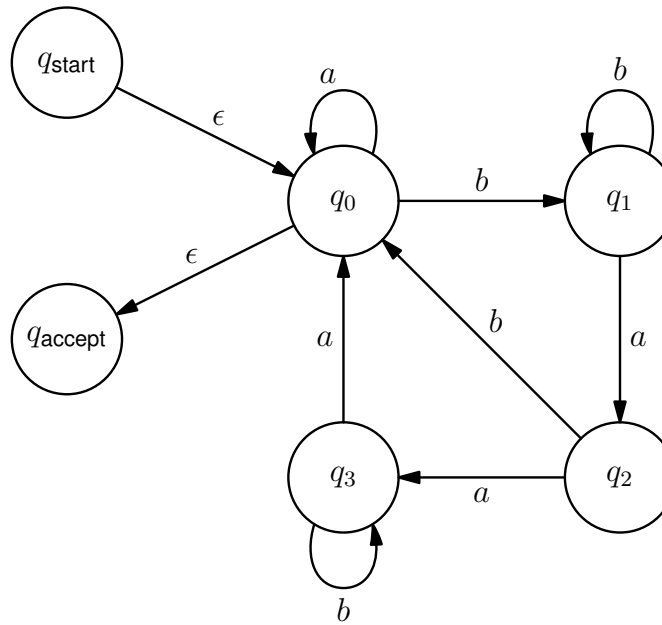


# State elimination example

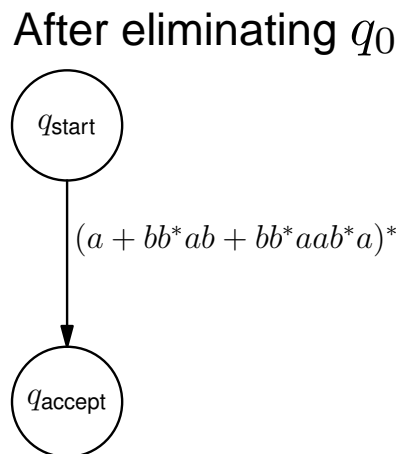
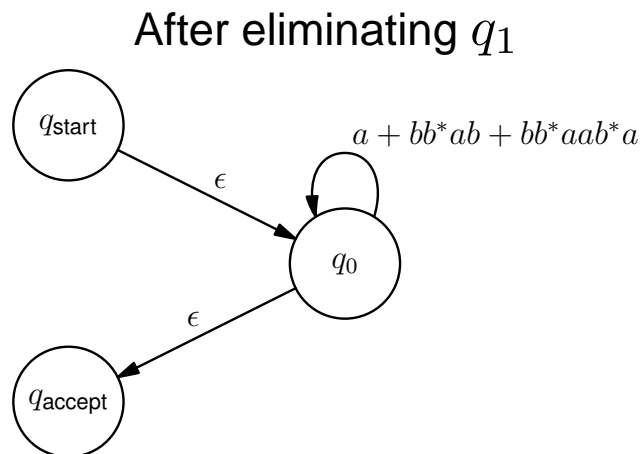
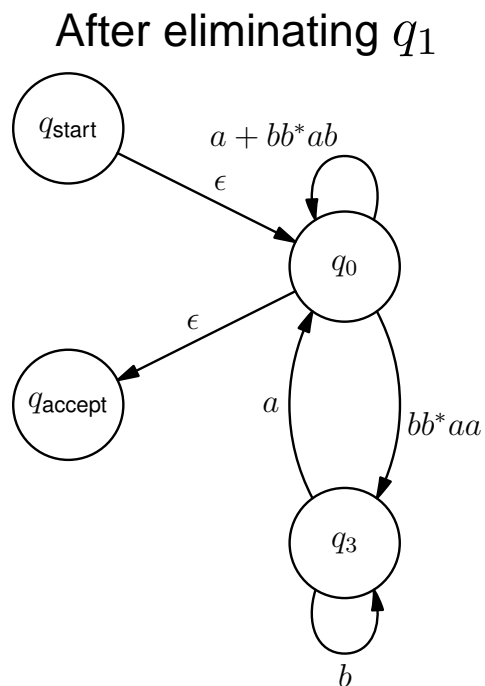
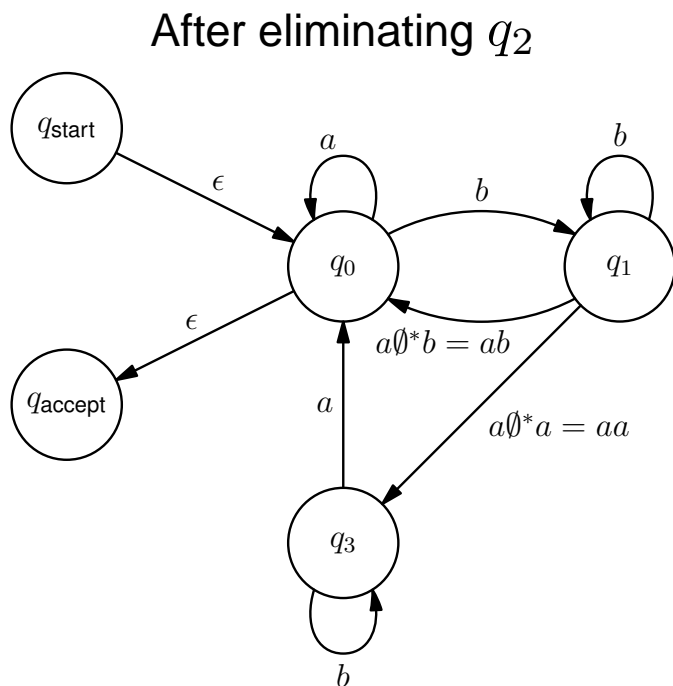
Original automaton



Equivalent GNFA



# State elimination example, cont.



## State elimination preserves equivalence

Claim:  $L(G) = L(G')$ .

$L(G) \subseteq L(G')$ : Let  $x \in L(G)$ . Then  $x = w_1 w_2 \dots w_n$  and there is some accepting run  $p = p_0 p_1 \dots p_n$  in  $G$ .

The accepting run  $p$  can be converted to an accepting run  $p'$  in  $G'$  by removing all the  $q_k$  states (it won't remove the start or accept states, because they are not eliminated).

To see this, consider a substring of states  $q_i q_k q_k \dots q_k q_j$ . The concatenation of the strings taking  $q_i$  to  $q_j$  in this sequence will be in  $L[\delta(q_i, q_j)]$ . So  $x \in L(G')$ .

$L(G') \subseteq L(G)$ : Let  $x \in L(G')$ . Then there is an accepting run  $p'$  in  $G'$ , and  $x = w'_1 w'_2 \dots w'_n$  where  $w'_i \in L[\delta(p'_i, p'_{i+1})]$  for all  $0 \leq i \leq n$ .

Each  $w'_i$  can be segmented into one or more substrings  $w_{i_0} w_{i_1} \dots w_{i_m}$  so that  $w_{i_0} \in L[\delta(p'_i, q_k)]$ ,  $w_{i_m} \in L[\delta(q_k, p'_{i+1})]$  and  $w_{i_\ell} \in L[\delta(q_k, q_k)]$ , by the construction of  $\delta'(p'_i, p'_{i+1})$ . So, the appropriate number of  $q_k$ 's can be inserted between consecutive states in  $p$  to construct an accepting run  $p$  in  $G$ . Hence,  $x \in L(G)$ .

## Induction

---

Theorem: Every GNFA can be reduced by state elimination to an equivalent two-state GNFA.

Proof: By induction on the number of states.

**Base:** The theorem obviously holds for two-state GNFA's.

**Induction:** If the GNFA has  $n + 1$  states, one of those states can be eliminated to give an equivalent GNFA with  $n$  states, by the previous claim.

By induction, that  $n$ -state GNFA can be reduced to an equivalent two-state GNFA.

## The whole thing

Theorem: For every finite automaton (any type)  $A$ , there is a regular expressions  $R$  such that  $L(R) = L(A)$ .

$A$  can be converted to an equivalent GNFA  $G$ .

$G$  can be reduced to an equivalent two-state GNFA  $G'$ .

The language of  $G'$  is the same as the regular expression  $R = \delta(q_{\text{start}}, q_{\text{accept}})$ .

## Summary of Regular Language Equivalence

$\epsilon$ -NFAs, NFAs, DFAs, and Regular Expressions all describe the same class of languages (the regular languages).

This is proved by providing translations between the different representations.

Reg Exp to  $\epsilon$ -NFA – implement all the regular operators.

NFA to a Reg Exp – convert paths to regular expressions.