

Administrative Notes

- The midterm is scheduled (**still tentatively**) to Tuesday, April 7th.
- To set up an appointment with Mr. Hod, the teaching assistant, please use the e-address **ranihod AT tau dot ac dot il**.

Administrative Notes 2

- Homework assignment 1 was published. It is unlikely you'll be able to solve it **on your own** if the first time you think about is the night before the deadline.
- A non-zero fraction of the mid term will be based directly on the homework assignments (of course, it will **not** be identical).
- While we can hardly detect dependencies in preparation of the homeworks (still, we sometime succeed), we actively try to enforce mutual independence in the midterm.

Computational Models - Lecture 2

- Non-Deterministic Finite Automata (NFA)
 - Closure of Regular Languages Under $\cup, \circ, *$
 - Regular **expressions**
 - Equivalence with finite automata
-
- Sipser's book, 1.1-1.3

DFA Formal Definition (reminder)

A **deterministic finite automaton** (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set called the **states**,
- Σ is a finite set called the **alphabet**,
- $\delta : Q \times \Sigma \rightarrow Q$ is the **transition function**,
- $q_0 \in Q$ is the **start state**, and
- $F \subseteq Q$ is the set of **accept states**.

Languages and DFA (reminder)

Definition: Let L ($L \subseteq \Sigma^*$) be the set of strings that M accepts. $L(M)$, the language of a DFA M , is defined as $L(M) = L$.

Note that

- M may accept many strings, but
- M accepts only one language.

A language is called **regular** if some deterministic finite automaton accepts it.

The Regular Operations (reminder)

Let A and B be languages.

The **union** operation:

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

The **concatenation** operation:

$$A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$$

The **star** operation:

$$A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$$

Claim: Closure Under Union (reminder)

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.
- in our construction, states of M were Cartesian product of M_1 and M_2 states.

What About Concatenation?

Thm: If L_1, L_2 are regular languages, so is $L_1 \circ L_2$.

Example: $L_1 = \{\text{good, bad}\}$ and $L_2 = \{\text{boy, girl}\}$.

$$L_1 \circ L_2 = \{\text{goodboy, goodgirl, badboy, badgirl}\}$$

This is much harder to prove.

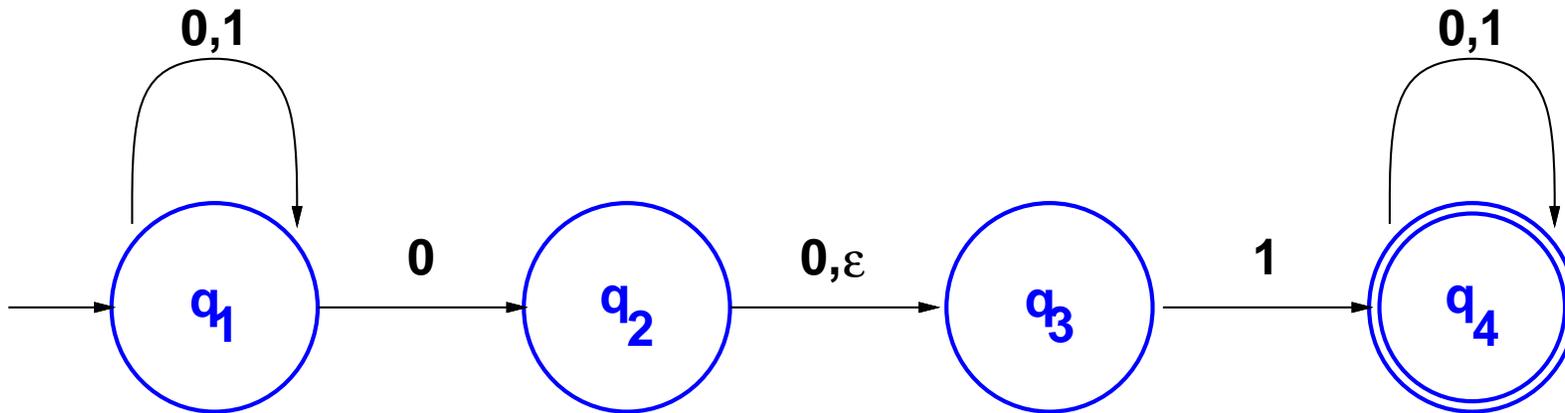
Idea: Simulate M_1 for a while, then **switch** to M_2 .

Problem: But **when** do you switch?

Seems hard to do with DFAs.

This leads us into **non-determinism**.

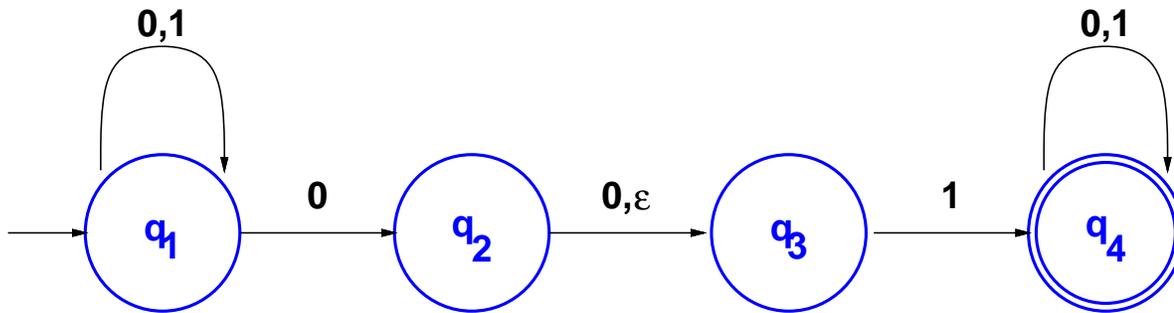
Non-Deterministic Finite Automata



- an NFA may have **more than one transition** labeled with the same symbol,
- an NFA may have **no transitions** labeled with a certain symbol, and
- an NFA may have transitions labeled with ε , the **empty string**.

Comment: Every **DFA** is also a non-deterministic finite automata (**NFA**).

Non-Deterministic Computation

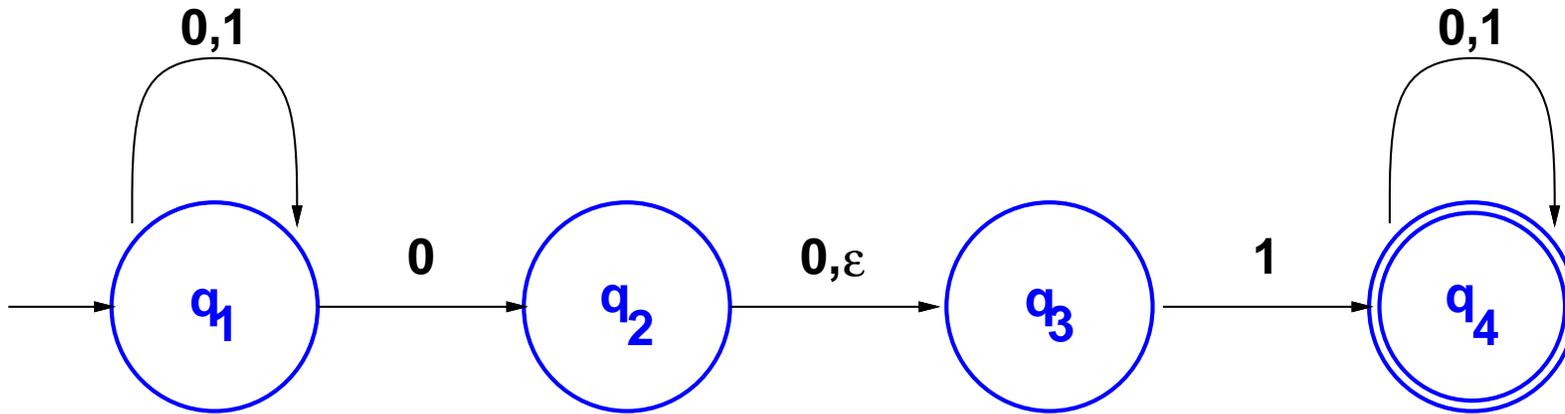


What happens when more than one transition is possible?

- the machine “splits” into **multiple copies**
- each branch follows one possibility
- together, branches follow **all** possibilities.
- If the input doesn't appear, that branch “dies”.
- Automaton accepts if **some** branch accepts.

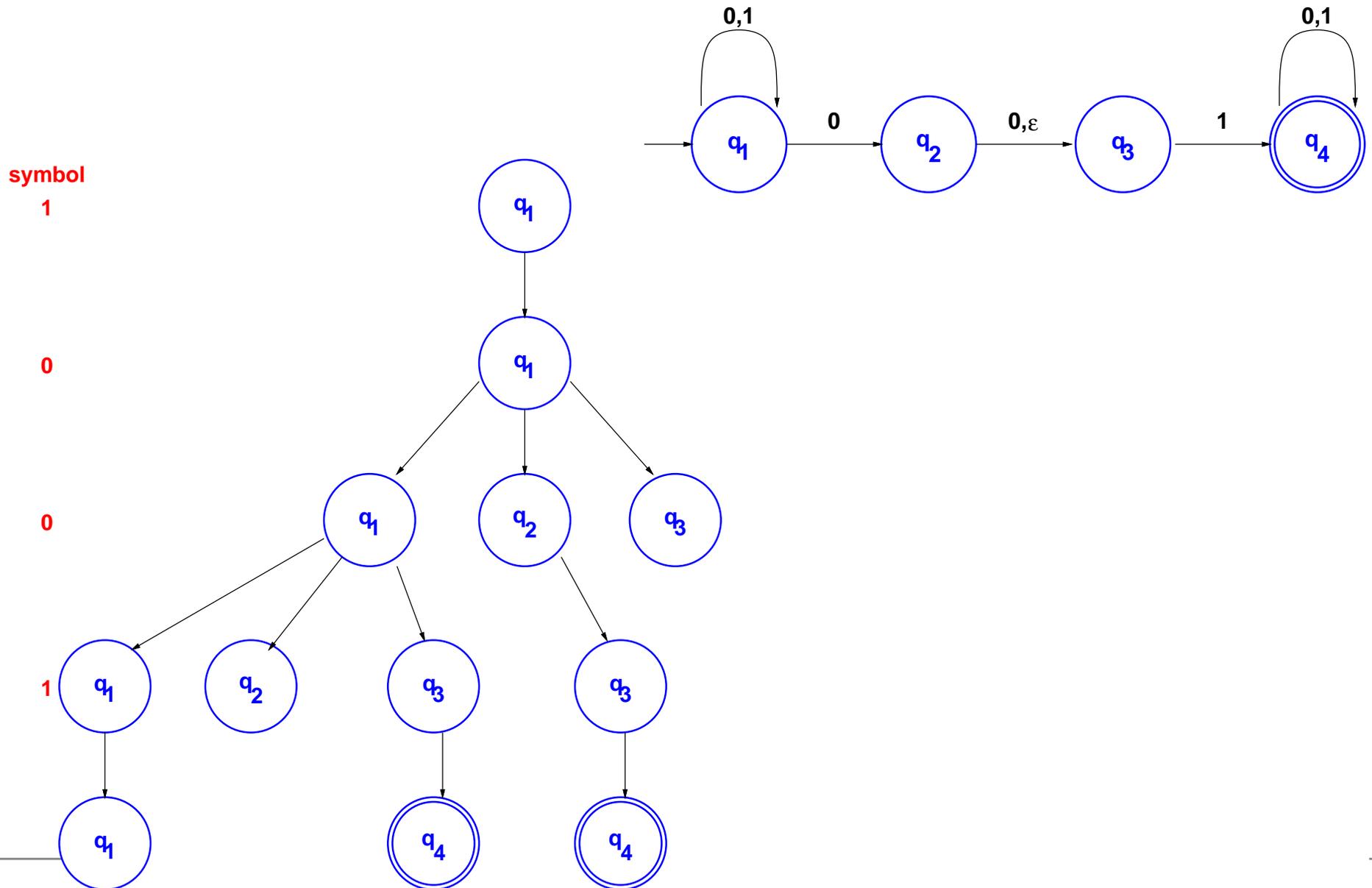
What does an ε transition do?

Non-Deterministic Computation



What happens on string **1001**?

The String 1001



Why Non-Determinism?

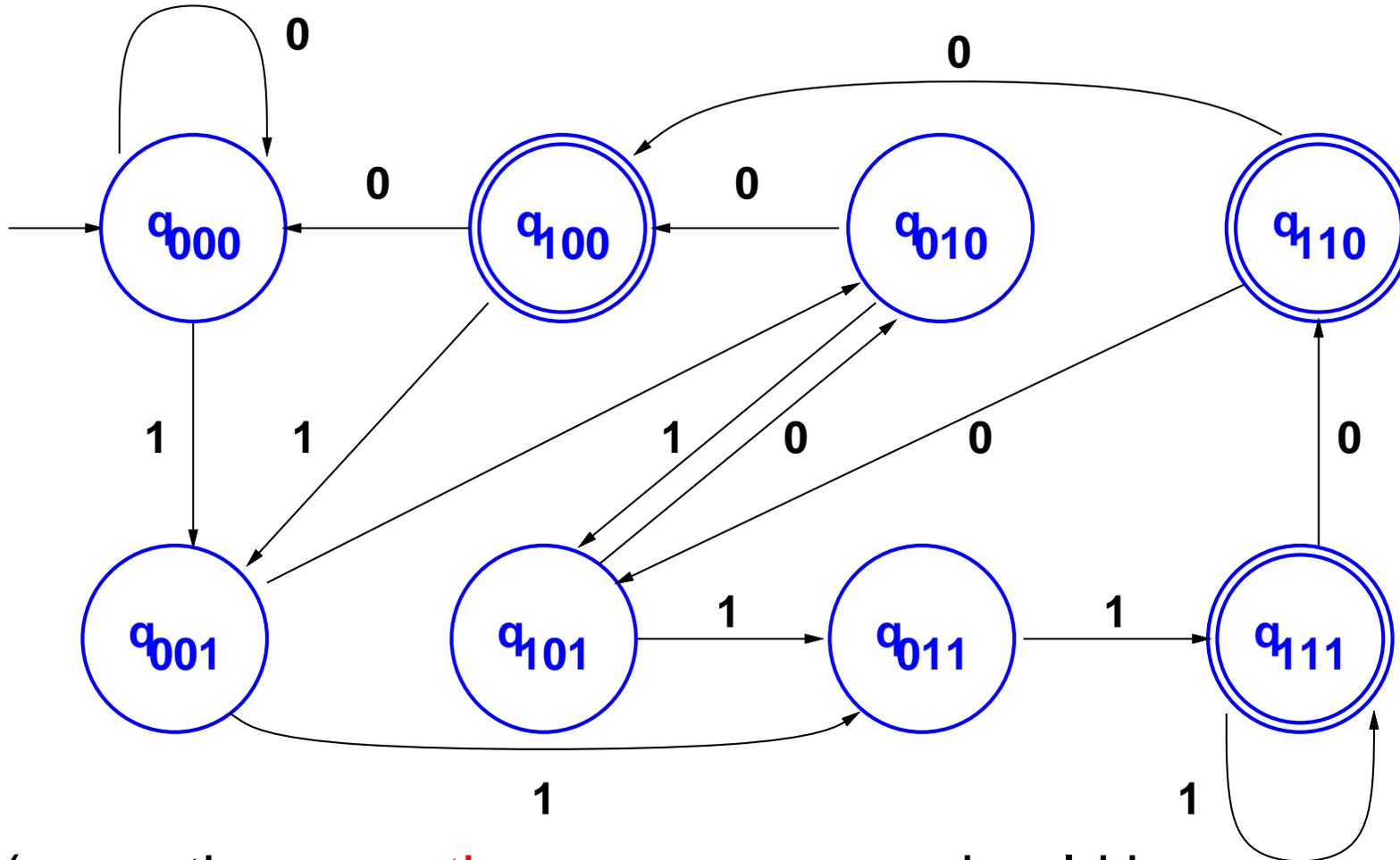
Theorem (to be proved soon): Deterministic and non-deterministic finite automata **accept** exactly the **same set of languages**.

Q.: So **why** do we need them?

A.: NFAs are usually **easier to design** than equivalent DFAs.

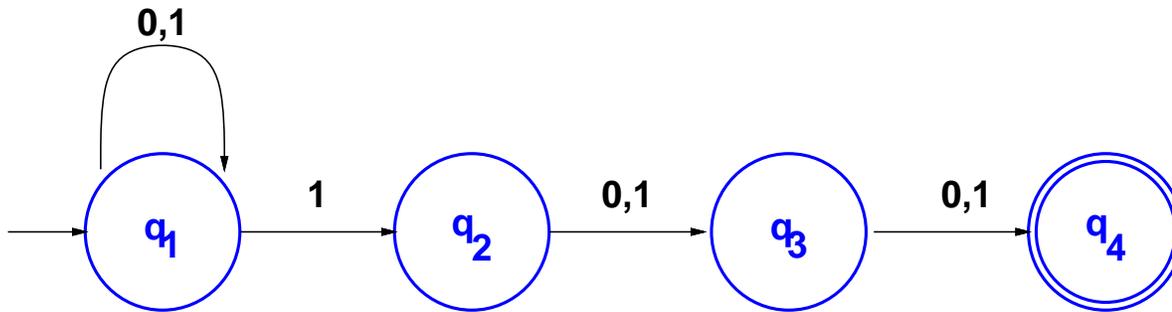
Example: Design a finite automaton that accepts all strings with a 1 in their third-to-the-last position?

Solving with DFA



(oops, there are **three** errors: q_{101} should be an **accept** state, there should be a **"0"** labeled arrow from q_{011} to q_{110} , and the existing arrow from q_{110} to q_{101} should be labeled **"0"**. But overall it is OK.)

Solving with NFA



- “Guesses” which symbol is third from the last, and
- checks that indeed it is a 1.
- If guess is premature, that branch “dies”, and no harm occurs.

NFA – Formal Definition

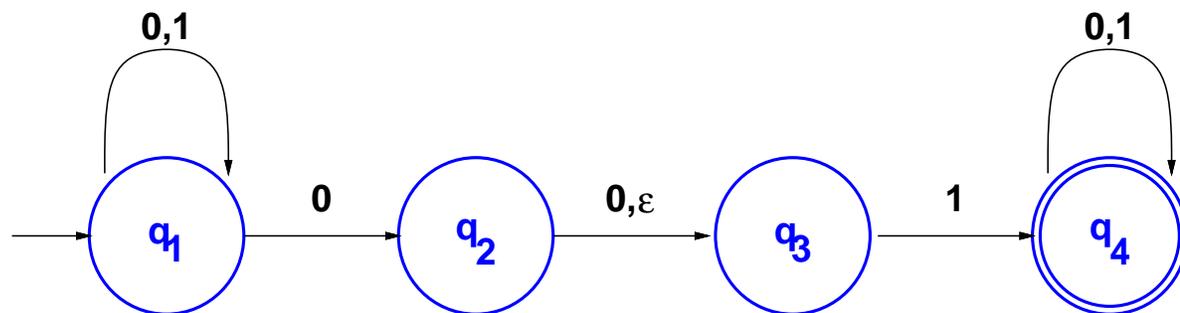
Transition function δ is going to be different.

- Let $\mathcal{P}(Q)$ denote the powerset of Q .
- Let Σ_ϵ denote $\Sigma \cup \{\epsilon\}$.

A non-deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states.

Example



$$N_1 = (Q, \Sigma, \delta, q_1, F)$$

where

- $Q = \{q_1, q_2, q_3, q_4\}, \Sigma = \{0, 1\},$

- δ is

	0	1	ε
q_1	$\{q_1, q_2\}$	$\{q_1\}$	\emptyset
q_2	$\{q_3\}$	\emptyset	$\{q_3\}$
q_3	\emptyset	$\{q_4\}$	\emptyset
q_4	$\{q_4\}$	$\{q_4\}$	\emptyset

- q_1 is the start state, and $F = \{q_4\}.$

Formal Model of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA, and
- w be a string over Σ_ϵ that has the form $y_1 y_2 \cdots y_m$ where $y_i \in \Sigma_\epsilon$.
- u be the string over Σ obtained from w by omitting all occurrences of ϵ .

Suppose there is a sequence of *states* (in Q), r_0, \dots, r_n , such that

- $r_0 = q_0$
- $r_{i+1} \in \delta(r_i, y_{i+1}), 0 \leq i < n$
- $r_n \in F$

Then we say that M **accepts** u .

Equivalence of NFA's and DFA's

- Given an **NFA**, N , we construct a **DFA**, M , that accepts the **same language**.
- To begin with, we make things easier by **ignoring ϵ** transitions.
- Make DFA simulate **all possible** NFA states.
- As consequence of the construction, if the NFA has k states, the DFA has 2^k states (an exponential blow up).

Equivalence of NFA's and DFA's

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA accepting A .

Construct a DFA $M = (Q', \Sigma, \delta', q'_0, F')$.

- $Q' = \mathcal{P}(Q)$.

- For $R \in Q'$ and $a \in \Sigma$, let

$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$$

- $q'_0 = \{q_0\}$

- $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$



Notice: F' is a set whose elements are subsets of Q , so (as expected) F' is a subset of $\mathcal{P}(Q)$.

Equivalence of NFA's and DFA's

Formally, use induction on m to show that if $y_1y_2 \cdots y_m$ is a string over Σ^* , and the set of all possible states that $N = (Q, \Sigma, \delta, q_0, F)$ could reach on it is $R \subseteq Q$, then the deterministic DFA, M , reaches state R on $y_1y_2 \cdots y_m$. Then, use the definition of acceptance by N , and of accept states for M .

Dealing with ε -Transitions

For any state R of M , define $E(R)$ to be the collection of states reachable from R by ε transitions only.

$$E(R) = \{q \in Q \mid q \text{ can be reached from some } r \in R \text{ by 0 or more } \varepsilon \text{ transitions}\}$$

Define transition function:

$$\delta'(R, a) = \{q \in Q \mid \text{there is some } r \in R \text{ such that } q \in E(\delta(r, a))\}$$

Change start state to

$$q'_0 = E(\{q_0\})$$



Regular Languages, Revisited

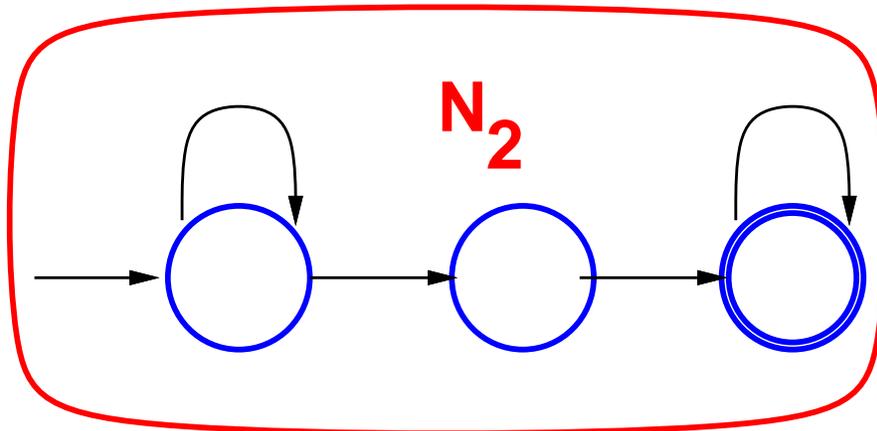
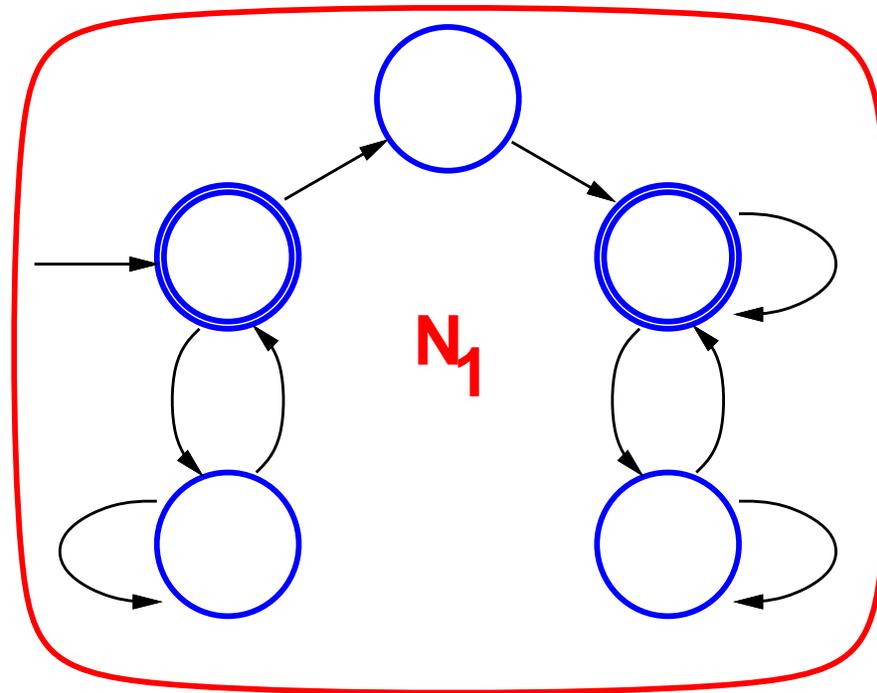
By definition, a language is regular if it is accepted by some **DFA**.

Corollary: A language is regular if and only if it is accepted by some **NFA**.

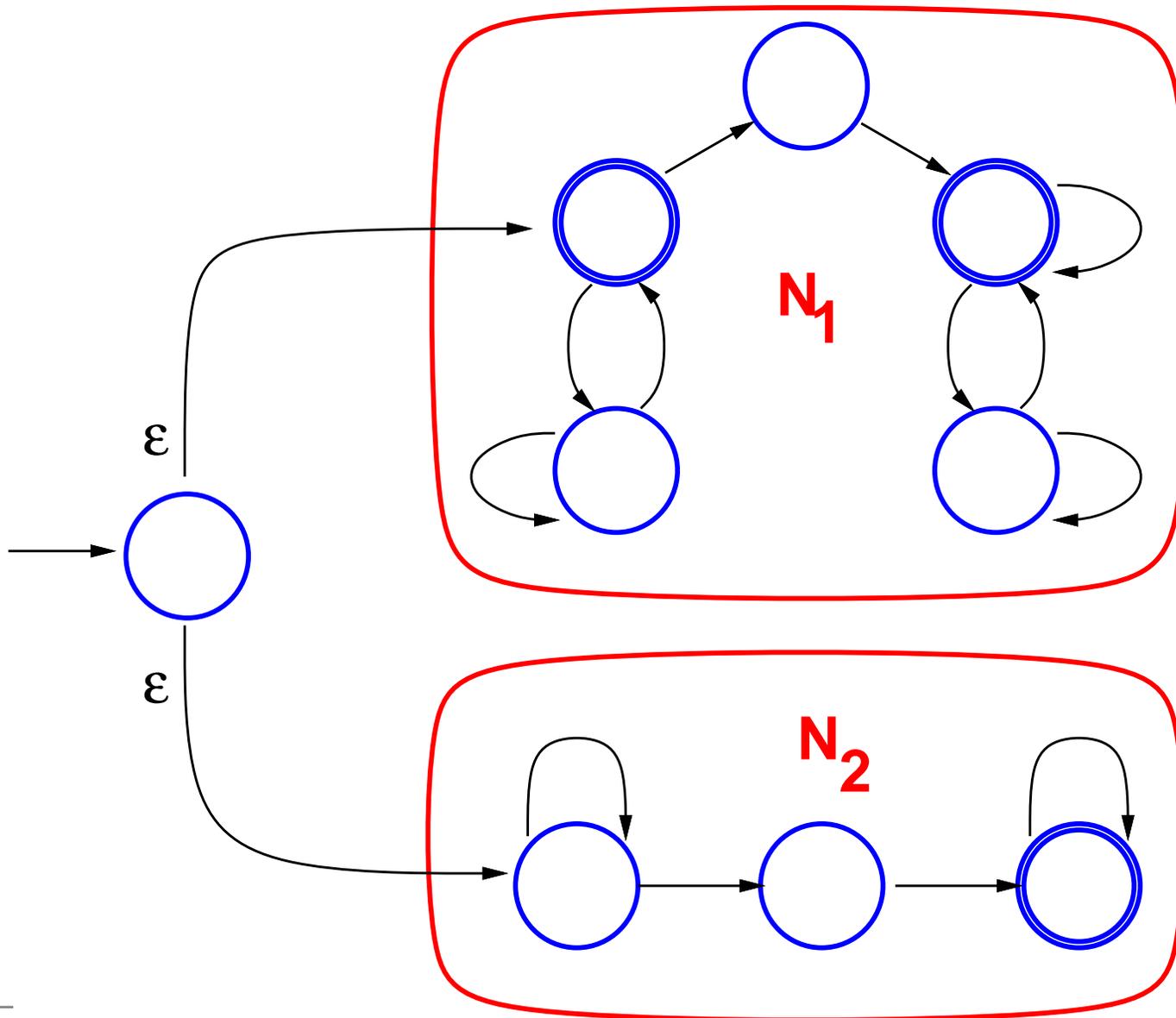
This is an alternative way of characterizing regular languages.

We will now use the equivalence to show that regular languages are **closed** under the regular operations (union, concatenation, star).

Closure Under Union (alternative proof)



Regular Languages Closed Under Union



Regular Languages Closed Under Union

Suppose

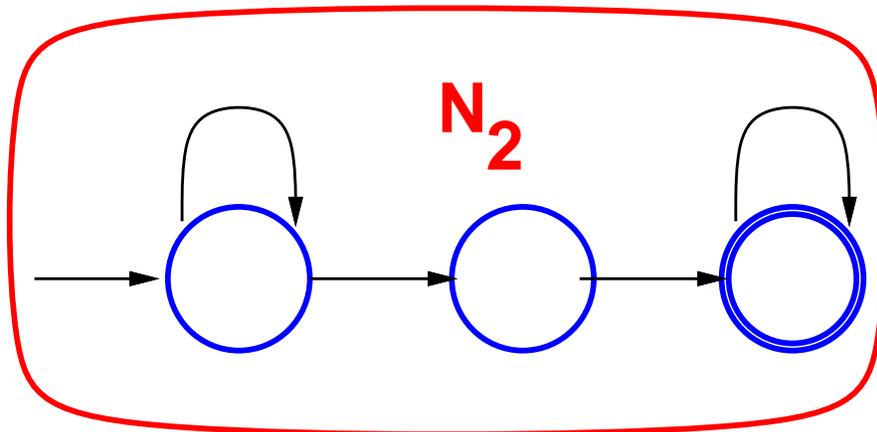
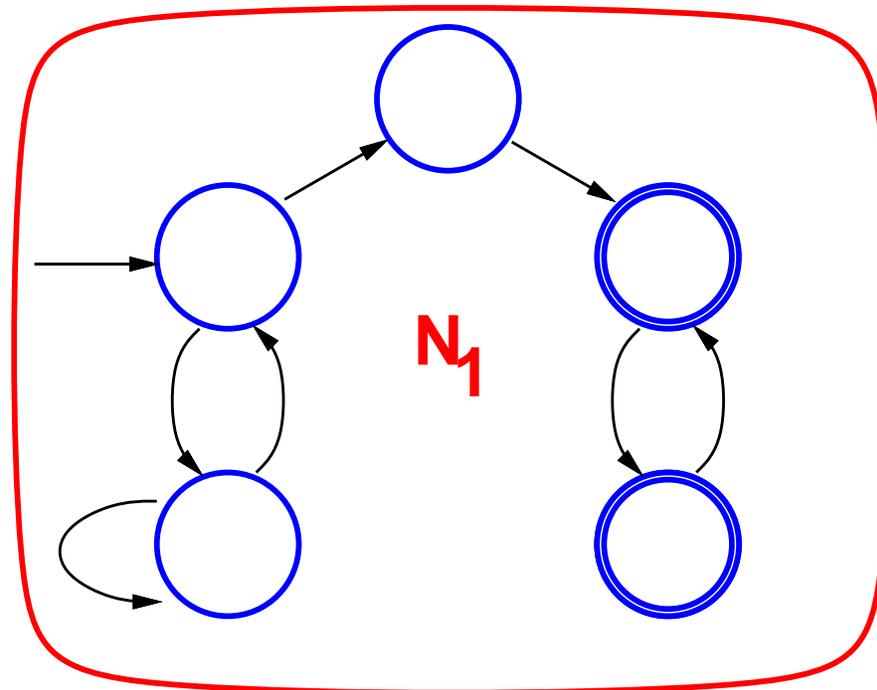
- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept L_1 , and
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept L_2 .

Define $N = (Q, \Sigma, \delta, q_0, F)$:

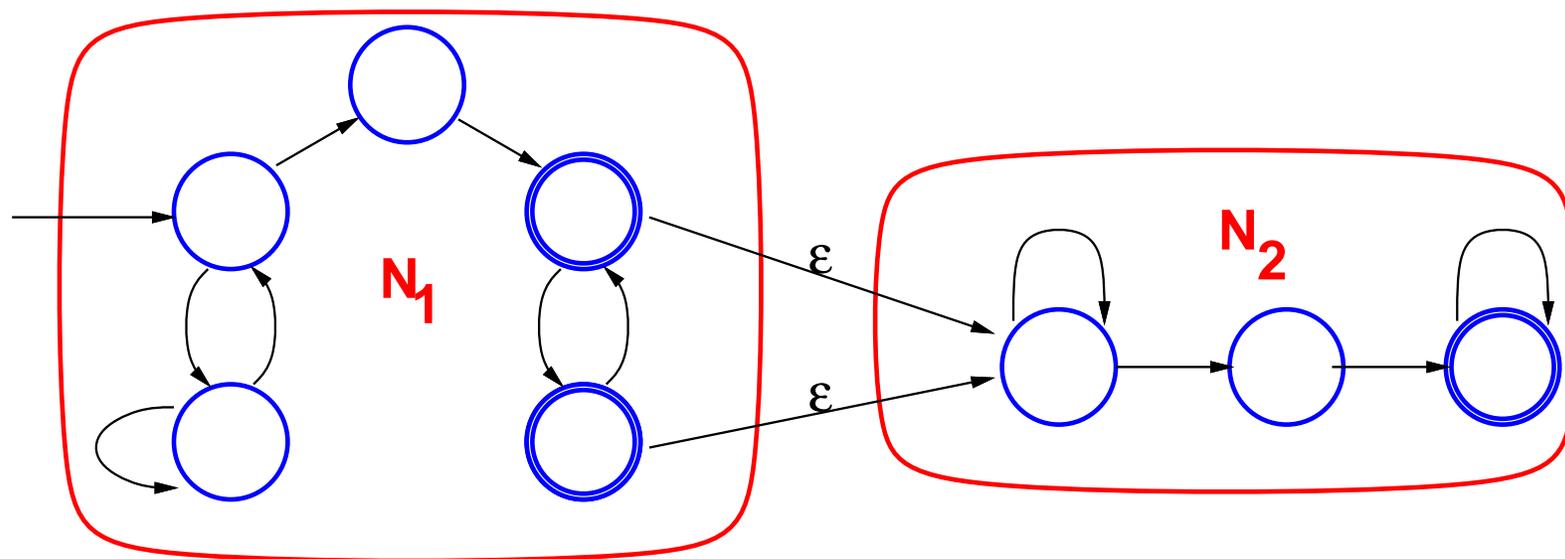
- $Q = q_0 \cup Q_1 \cup Q_2$
- Σ is the same, q_0 is the start state
- $F = F_1 \cup F_2$

$$\bullet \delta'(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

Regular Languages Closed Under Concatenation



Regular Languages Closed Under Concatenation



Remark: Final states are exactly those of N_2 .

Regular Languages Closed Under Concatenation

Suppose

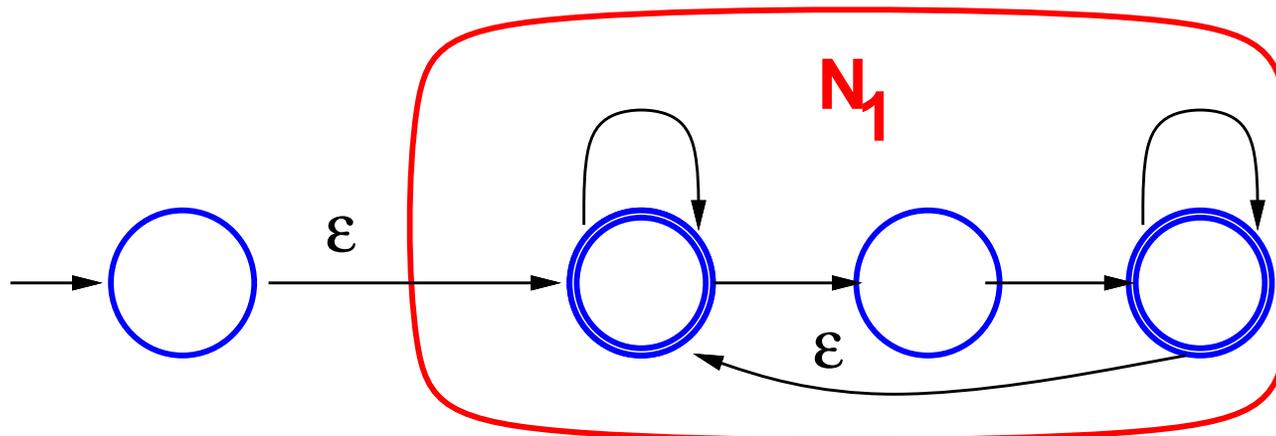
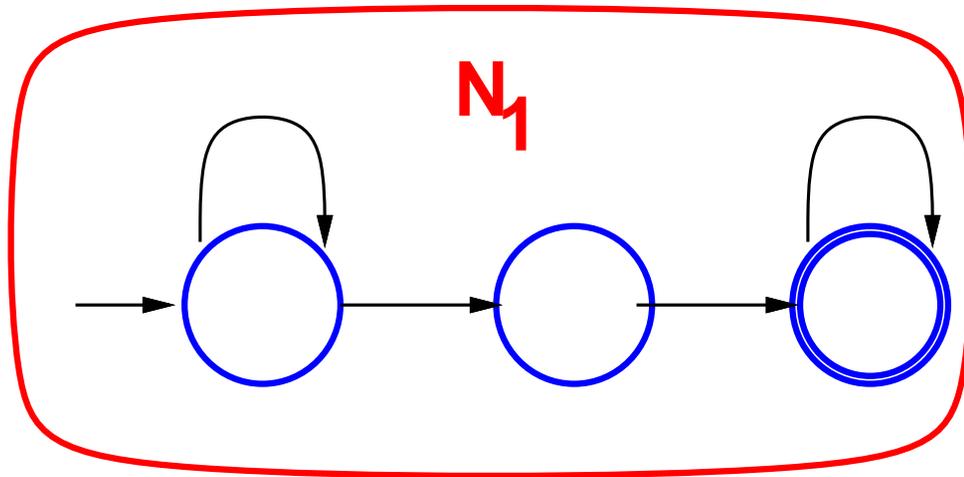
- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept L_1 , and
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept L_2 .

Define $N = (Q, \Sigma, \delta, q_1, F_2)$:

- $Q = Q_1 \cup Q_2$
- q_1 is the start state of N
- F_2 is the set of **accept states** of N

$$\delta'(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in Q_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q, a) & q \in Q_2 \end{cases}$$

Regular Languages Closed Under Star

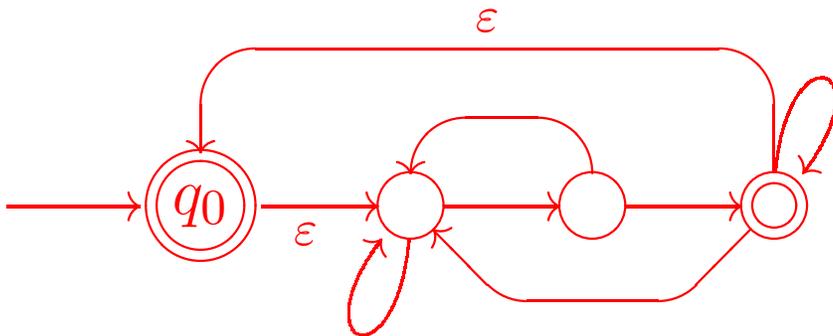
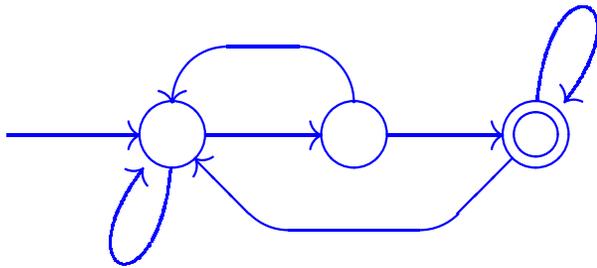


Oops - bad construction (really? why?).

How do we fix it?

Regular Languages Closed Under Star

N_1 accepts R_1 . Wanna build NFA for $R = (R_1)^*$.



Ahaa - a better construction!

Regular Languages Closed Under Star

Suppose $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1 .

Define $N = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0\} \cup Q_1$
- q_0 is the new start state.
- $F = \{q_0 \cup F_1\}$

$$\delta'(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, \varepsilon) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

Summary

- **Regular languages** are closed under
 - union
 - concatenation
 - star
- **Non-deterministic** finite automata
 - are equivalent to **deterministic** finite automata
 - but much easier to use in some proofs and constructions.

Regular Expressions

A notation for building up languages by describing them as expressions, e.g. $(0 \cup 1)0^*$.

- 0 and 1 are shorthand for $\{0\}$ and $\{1\}$
- so $(0 \cup 1) = \{0, 1\}$.
- 0^* is shorthand for $\{0\}^*$.
- concatenation, like multiplication, is implicit, so 0^*10^* is shorthand for the set of all strings over $\Sigma = \{0, 1\}$ having exactly a single 1 .

Q.: What does $(0 \cup 1)0^*$ stand for?

Remark: Regular expressions are often used in text editors or shell scripts.

More Examples

Let Σ be an alphabet.

- The regular expression Σ is the language of one-symbol strings.
- Σ^* is all strings.
- Σ^*1 all strings ending in 1.
- $0\Sigma^* \cup \Sigma^*1$ strings starting with 0 or ending in 1.

Just like in arithmetic, operations have precedence:

- star first
- concatenation next
- union last
- parentheses used to change default order

Regular Expressions – Formal Definition

Syntax: R is a regular expression if R is of form

- a for some $a \in \Sigma$
- ε
- \emptyset
- $(R_1 \cup R_2)$ for regular expressions R_1 and R_2
- $(R_1 \circ R_2)$ for regular expressions R_1 and R_2
- (R_1^*) for regular expression R_1

Regular Expressions – Formal Definition

Let $L(R)$ be the language denoted by regular expression R .

R	$L(R)$
a	$\{a\}$
ε	$\{\varepsilon\}$
$(R_1 \cup R_2)$	$L(R_1) \cup L(R_2)$
$(R_1 \circ R_2)$	$L(R_1) \circ L(R_2)$
$(R_1)^*$	$L(R_1)^*$

Q.: What's the difference between \emptyset and ε ?

Q.: Isn't this definition circular?

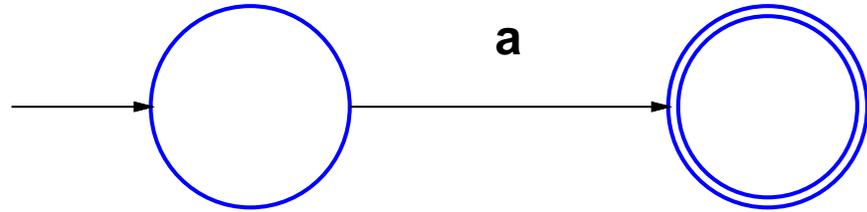
Remarkable Fact

Thm.: A language, L , is described by a regular expression, R , if and only if L is regular.

\implies construct an NFA accepting R .

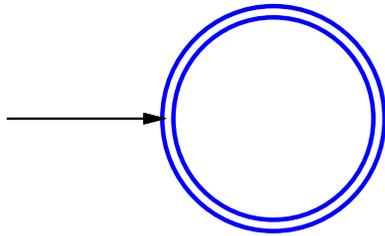
\impliedby Given a regular language, L , construct an equivalent regular expression.

Given R , Build NFA Accepting It (\implies)

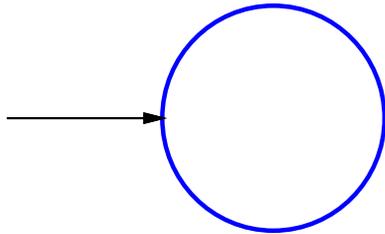


1. $R = a$, for some $a \in \Sigma$

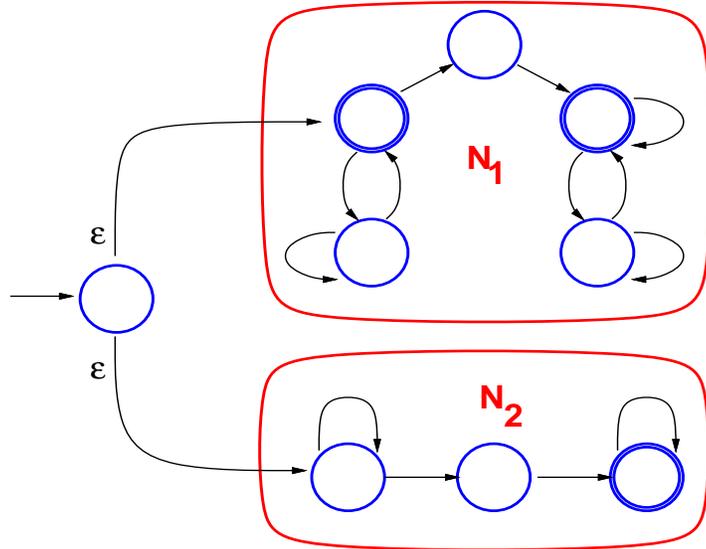
2. $R = \varepsilon$



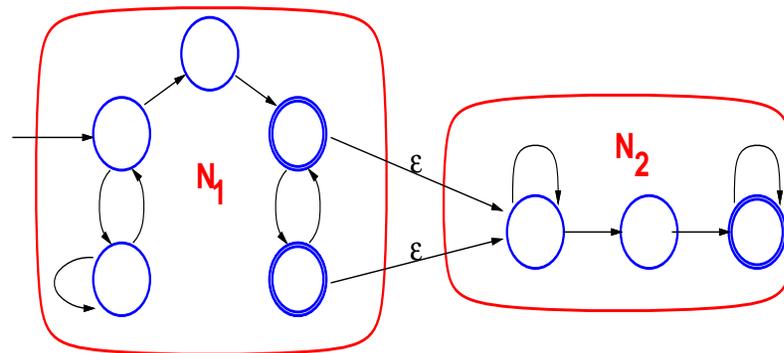
3. $R = \emptyset$



Given R , Build NFA Accepting It (\implies)

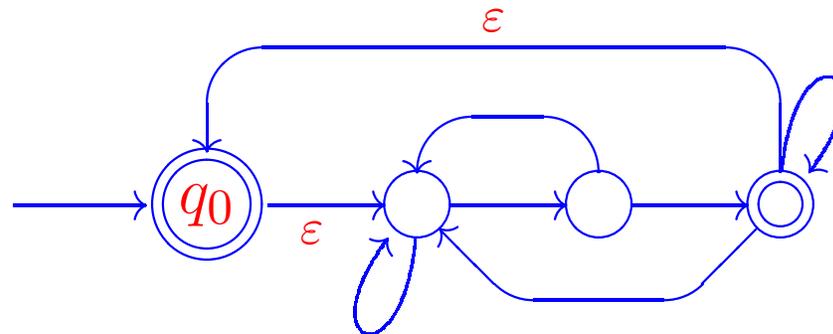


$$R = (R_1 \cup R_2)$$



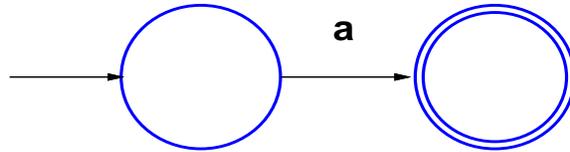
$$R = (R_1 \circ R_2)$$

$$R = (R_1)^*$$

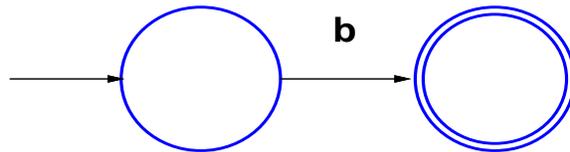


Approximately Correct Examples

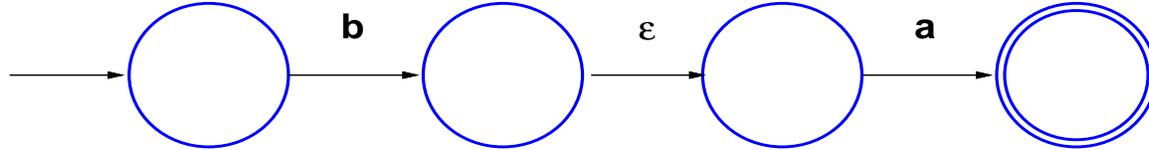
a



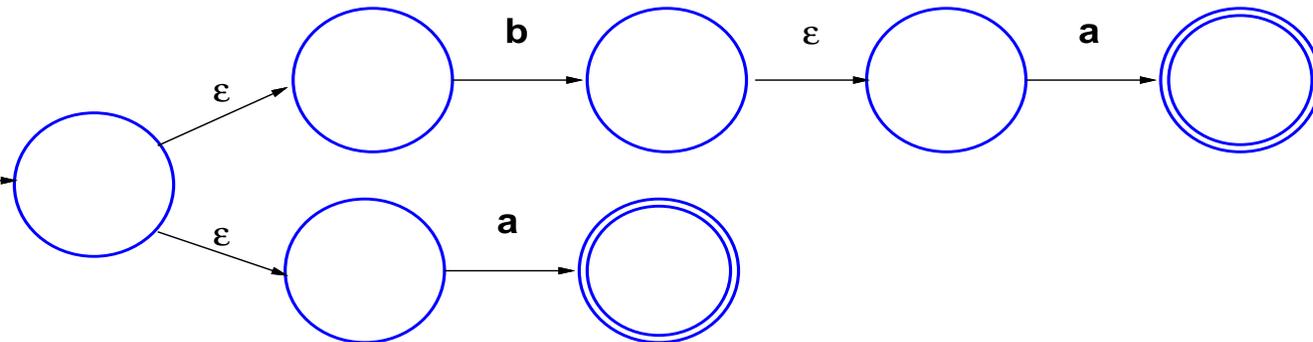
b



ab



ab ∪ a



only “approximately”?)

(why

Regular Expression from an NFA (\Leftarrow)

We now define **generalized** non-deterministic finite automata (**GNFA**).

An **NFA**:

- Each transition labeled with a symbol or ε ,
- reads **zero or one** symbols,
- takes matching transition, if any.

A **GNFA**:

- Each transition labeled with a **regular expression**,
- reads **zero** or **more** symbols,
- takes transition whose **regular expression** matches string, if any.

GNFAs are natural generalization of NFAs.

A Special Form of GNFA

- **Start state** has outgoing arrows to **every** other state, but no incoming arrows.
- Unique **accept state** has incoming arrows from **every** other state, but no outgoing arrows.
- Except for start and accept states, arrows goes **from every state to every other state**, including itself.

Easy to transform any GNFA into special form.

Really? How? ...

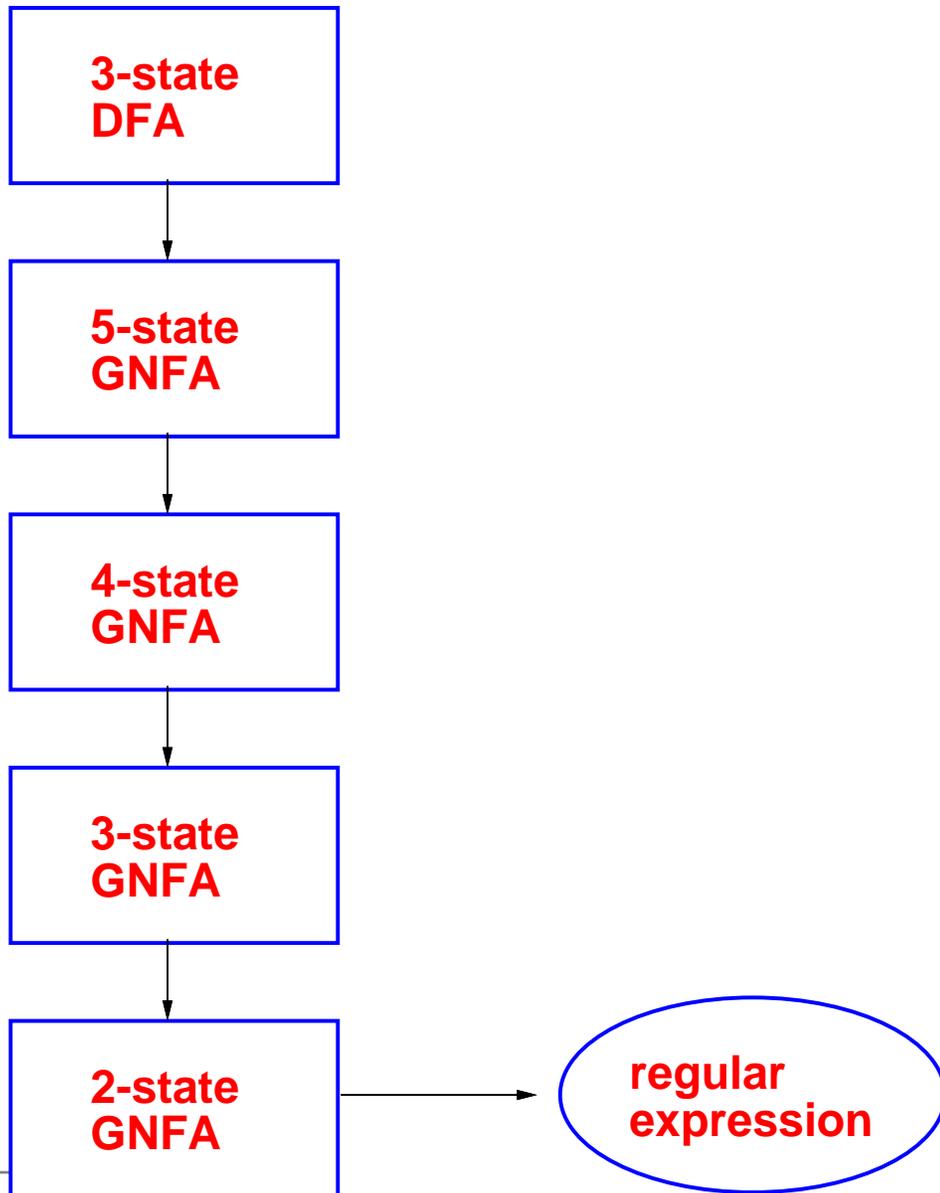
Converting DFA to Regular Expression (\iff)

Strategy – sequence of **equivalent** transformations

- given a k -state DFA
- transform into $(k + 2)$ -state GNFA
- while GNFA has **more than 2 states**, transform it into equivalent GNFA with **one fewer** state
- eventually reach **2-state** GNFA (states are just **start** and **accept**).
- label on single transition is the **desired regular expression**.

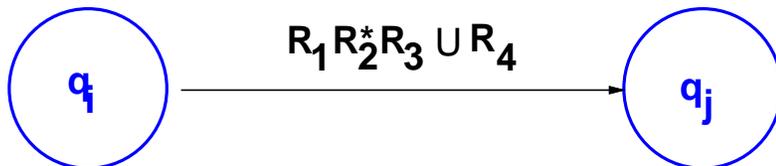
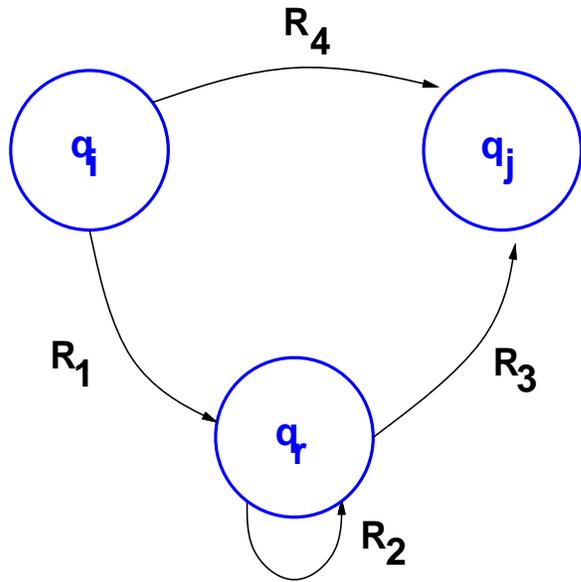


Converting Strategy (\Leftarrow)



Removing One State

We **remove** one state q_r , and then **repair** the machine by **altering** regular expression of other transitions.



Formal Treatment – GNDA Definition

- q_s is start state.
- q_a is accept state.
- \mathcal{R} is collection of regular expressions over Σ .

The transition function is

$$\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$$

If $\delta(q_i, q_j) = R$, then **arrow** from q_i to q_j has **label** R .

Arrows connect every state to every other state except:

- no arrow from q_a
- no arrow to q_s

Formal Definition

A **generalized** deterministic finite automaton (**GNFA**) is $(Q, \Sigma, \delta, q_s, q_a)$, where

- Q is a finite set of **states**,
- Σ is the **alphabet**,
- $\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$ is the **transition function**.
- $q_s \in Q$ is the **start state**, and
- $q_a \in Q$ is the unique **accept state**.

A Formal Model of GNFA Computation

A GNFA accepts a string $w \in \Sigma^*$ if **there exists** a *parsing* of w , $w = w_1 w_2 \cdots w_k$, where each $w_i \in \Sigma^*$, and **there exists** a sequence of states q_0, \dots, q_k such that

- $q_0 = q_s$, the start state,
- $q_k = q_a$, the accept state, and
- for each i , $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.
- (namely w_i is an element of the language described by the regular expression R_i .)

The CONVERT Algorithm

Given GNFA G , convert it to equivalent GNFA G' .

- let k be the number of states of G .
- If $k = 2$, return the regular expression labeling the only arrow.
- If $k > 2$, select any q_r distinct from q_s and q_a .
- Let $Q' = Q - \{q_r\}$.
- For any $q_i \in Q' - \{q_a\}$ and $q_j \in Q' - \{q_s\}$, let
 - $R_1 = \delta(q_i, q_r)$, $R_2 = \delta(q_r, q_r)$,
 - $R_3 = \delta(q_r, q_j)$, and $R_4 = \delta(q_i, q_j)$.
- Define $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$.
- Denote the resulting $k - 1$ states GNFA by G' .

The CONVERT Procedure

We define the recursive procedure **CONVERT**(\cdot):

Given GNFA G .

- Let k be the number of states of G .
- If $k = 2$, return the regular expression labeling the only arrow of G .
- If $k > 2$, let G' be the $k - 1$ states GNFA produced by the algorithm.
- Return **CONVERT**(G').

Conversion - Example

- We now construct a simple, 2 state DFA that accepts the language over $\{0, 1\}$ of all strings with an **even** number of **1**s.
- We followed the conversion through GNFA's to translate this DFA (on the blackboard) into the regular expression $(0 \cup 10^*1)^*$.

Correctness Proof of Construction

Theorem: G and $\text{CONVERT}(G)$ accept the same language.

Proof: By induction on number of states of G

Basis: When there are only 2 states, there is a single label, which characterizes the strings accepted by G .

Induction Step: Assume claim for $k - 1$ states, prove for k .

Let G' be the $k - 1$ states GNFA produced from G by the algorithm.

G and G' accept the same language

By the induction hypothesis, G' and $\text{CONVERT}(G')$ accept the same language.

On input G , the procedure returns $\text{CONVERT}(G')$.

So to complete the proof, it suffices to show that G and G' accept the same language.

Three steps:

1. If G accepts the string w , then so does G' .
2. If G' accepts the string w , then so does G .
3. Therefore G and G' are equivalent.

Step One

Claim: If G accepts w , then so does G' :

- If G accepts w , then there exists a “path of states” $q_s, q_1, q_2, \dots, q_a$ traversed by G on w , leading to the accept state q_a .
- If q_r does not appear on path, then G' accepts w because the the new regular expression on each edge of G' contains the old regular expression in the “union part”.
- If q_r does appear, consider the regular expression corresponding to $\dots q_i, q_r, \dots, q_r, q_j \dots$. The new regular expression $(R_{i,r})(R_{r,r})^*(R_{r,j})$ linking q_i and q_j encompasses any such string.
- In both cases, the claim holds.

Steps Two and Three

Claim: If G' accepts w , then so does G .

Proof: Each transition from q_i to q_j in G' corresponds to a transition in G , either directly or through q_r . Thus if G' accepts w , then so does G .

- This completes the proof of the claim that $L(G) = L(G')$.
- Combined with the induction hypothesis, this shows that G and the regular expression $\text{CONVERT}(G)$ accept the same language.
- This, in turn, proves our *remarkable claim*:
A language, L , is described by a regular expression, R , if and only if L is regular. 