

Pre-lecture brain teaser

Find the regular expression for the language containing all binary strings that do not contain the subsequence 111000

ECE-374-B: Lecture 3 - NFAs

Instructor: Nickvash Kani

September 04, 2025

University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

Find the regular expression for the language containing all binary strings that **do not** contain the subsequence 111000



$$\begin{aligned} R_{\text{NOT } 111000} = & 0^* + 0^*10^* + 0^*10^*10^* \\ & + 0^*10^*0^*1^* + 0^*10^*10^*1^*01^* \\ & + 0^*10^*10^*1^*01^*01^* \end{aligned}$$

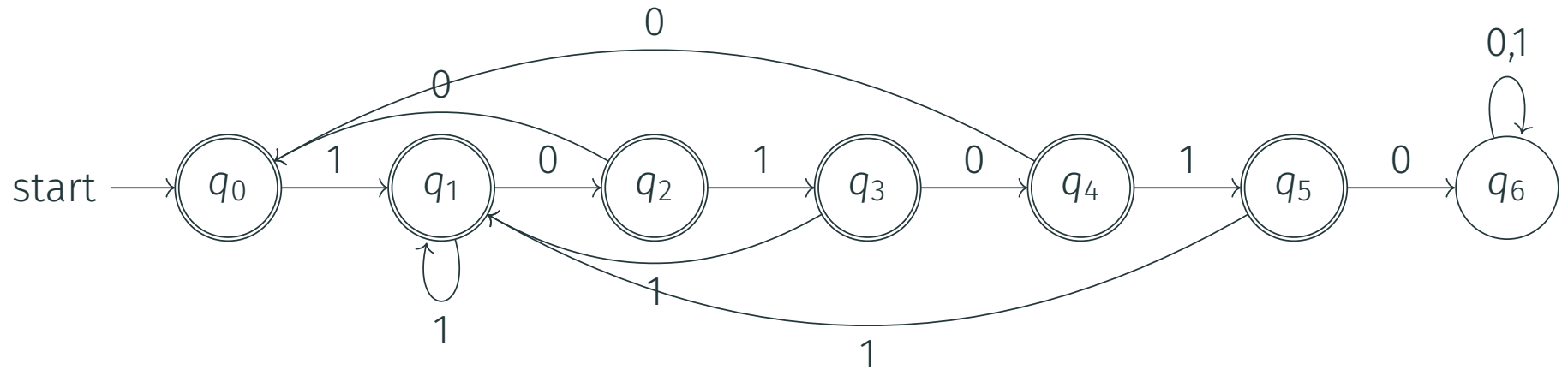
$$1^* = 1_2^*$$

Pre-lecture brain teaser II

Find the regular expression for the language containing all binary strings that **do not** contain the substring 101010

Pre-lecture brain teaser II

Find the regular expression for the language containing all binary strings that **do not** contain the substring **101010**

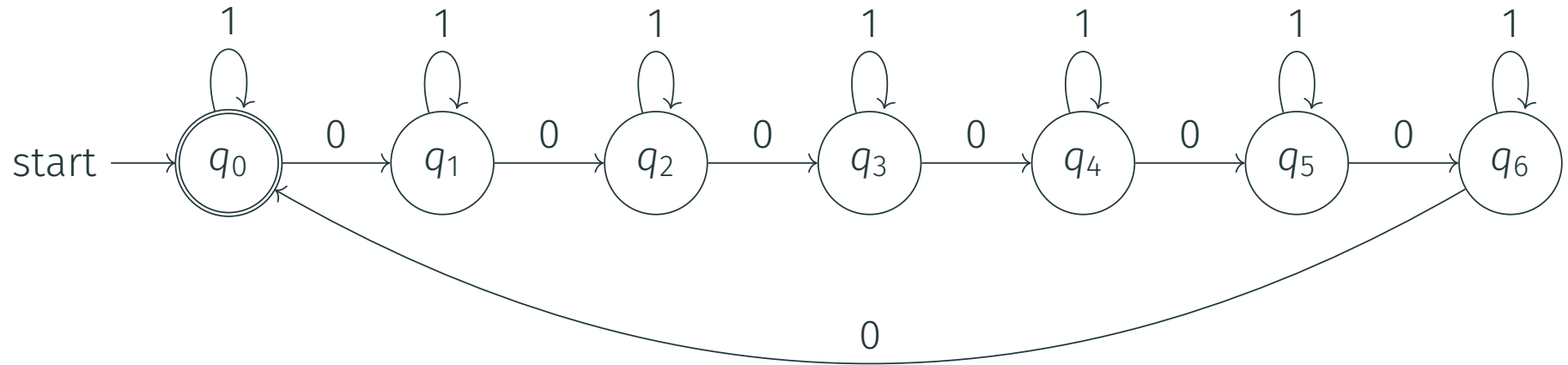


Pre-lecture brain teaser III

Find the regular expression for the language contains all binary strings whose $\#_0(w) \% 7 = 0$ (number of 0's divisible by 7).

Pre-lecture brain teaser III

Find the regular expression for the language contains all binary strings whose $\#_0(w) \% 7 = 0$ (number of 0's divisible by 7).



Pre-lecture brain teaser III

Show that the following string(w) is a member of the language that:

- does not contain the subsequence 111000 or
- does not contain the substring 101010 or
- or has a number of 0's divisible by 7

Pre-lecture brain teaser III

Show that the following string(w) is a member of the language that:

- does not contain the subsequence 111000 or
- does not contain the substring 101010 or
- or has a number of 0's divisible by 7

$w =$ 1001110110111001
1000010111110010
0101010011001111
1001001011111100

You have 30 seconds.

Pre-lecture brain teaser III

Show that the following string(w) is a member of the language that:

- does not contain the subsequence 111000 or
- does not contain the substring 101010 or
- or has a number of 0's divisible by 7

$w =$ 1001110110111001
1000010111110010
0101010011001111
1001001011111100

Yes

You have 30 seconds. Pray, choose a strategy and hope you get **lucky**.

Tangential Thought

Does luck allow us to solve unsolvable problems?

Tangential Thought

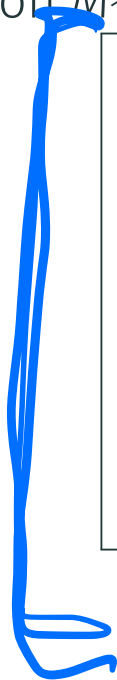
Does luck allow us to solve unsolvable problems? New example: Consider two machines: M_1 and M_2

- M_1 is a classic deterministic machine.
- M_2 is a “lucky” machine that will always make the right choice.

Lucky machine programs

Problem: Find shortest path from a to b

Program on M_1 (Dijkstra's algorithm):



```
Initialize for each node  $v$ ,  $\text{Dist}(s, v) = d'(s, v) = \infty$   
Initialize  $X = \emptyset$ ,  $d'(s, s) = 0$   
for  $i = 1$  to  $|V|$  do  
    Let  $v$  be node realizing  $d'(s, v) = \min_{u \in V - X} d'(s, u)$   
     $\text{Dist}(s, v) = d'(s, v)$   
     $X = X \cup \{v\}$   
    Update  $d'(s, u)$  for each  $u$  in  $V - X$  as follows:  
         $d'(s, u) = \min(d'(s, u), \text{Dist}(s, v) + \ell(v, u))$ 
```

Lucky machine programs

Problem: Find shortest path from a to b

Program on M_2 (Blind luck):

```
Initialize  $path = []$   
 $path += a$   
While( $not at b$ )  
    take an outgoing edge  $(u, v)$  from current node  $u$  to  $v$   
     $current = v$   
     $path += v$   
return  $path$ 
```

Tangential Thought

Does luck allow us to solve unsolvable problems?

Consider two machines: M_1 and M_2

- M_1 is a classic deterministic machine.
- M_2 is a “lucky” machine that will always make the right choice.

Question:

Tangential Thought

Does luck allow us to solve unsolvable problems?

Consider two machines: M_1 and M_2

- M_1 is a classic deterministic machine.
- M_2 is a “lucky” machine that will always make the right choice.

Question: Are there problems which M_2 can solve that M_1 cannot.

Tangential Thought

Does luck allow us to solve unsolvable problems?

Consider two machines: M_1 and M_2

- M_1 is a classic deterministic machine.
- M_2 is a “lucky” machine that will always make the right choice.

Question: Are there problems which M_2 can solve that M_1 cannot.

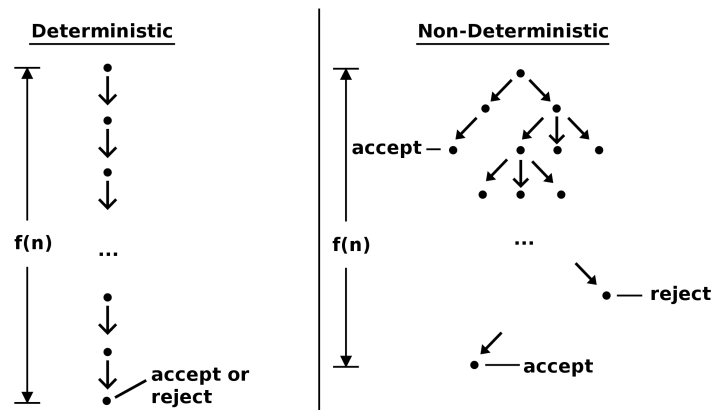
The notion was first posed by **Robert W. Floyd** in 1967.

Non-determinism in computing

In computer science, a nondeterministic machine is a theoretical device that can have more than one output for the same input.

A machine that is capable of taking multiple states concurrently. Whenever it reaches a choice, it takes both paths.

If there is a path for the string to be accepted by the machine, then the string is part of the language.



Non-determinism in media

Placeholder slide for youtube.

Why non-determinism?

- Non-determinism adds power to the model; richer programming language and hence (much) easier to “design” programs
- Fundamental in **theory** to prove many theorems
- Very important in **practice** directly and indirectly
- Many deep connections to various fields in Computer Science and Mathematics

Many interpretations of non-determinism. Hard to understand at the outset. Get used to it and then you will appreciate it slowly.

Non-deterministic finite automata (NFA) Introduction

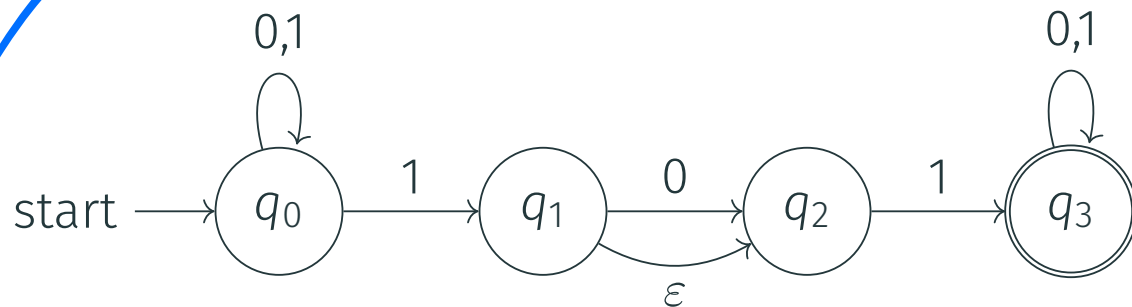
Non-deterministic Finite State Automata by example

When you come to a fork in the road, take it.

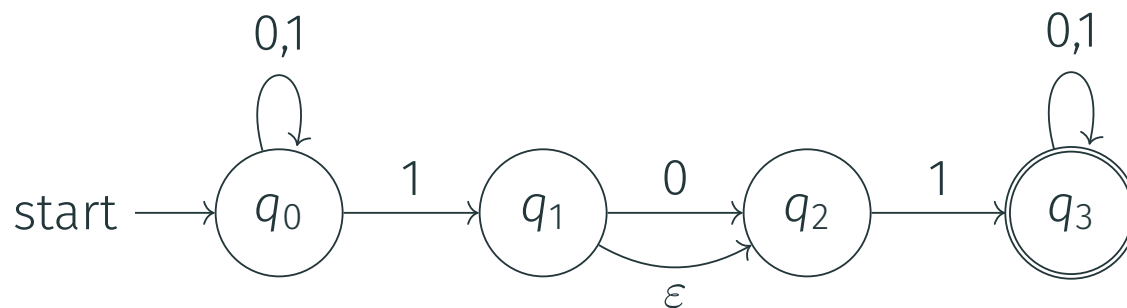
Non-deterministic Finite State Automata by example

When you come to a fork in the road, take it.

Today we'll talk about automata whose logic **is not** deterministic.

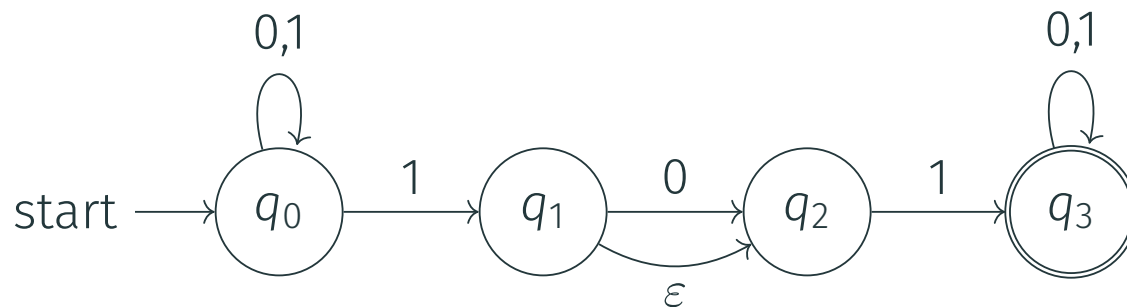


NFA acceptance: Informal



Informal definition: An NFA N **accepts a string** w iff some accepting state is reached by N from the start state on input w .

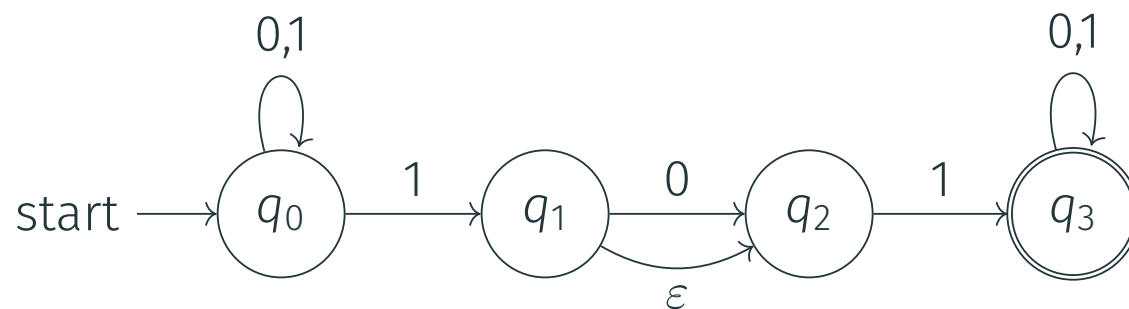
NFA acceptance: Informal



Informal definition: An NFA N accepts a string w iff some accepting state is *possible* reached by N from the start state on input w .

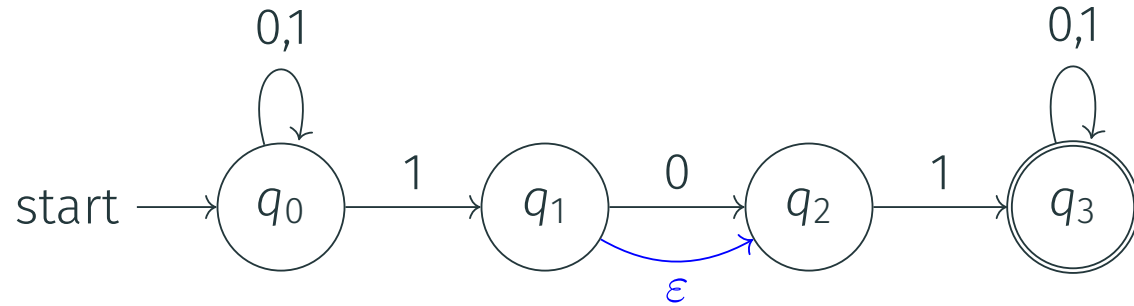
The language accepted (or recognized) by a NFA N is denoted by $L(N)$ and defined as: $L(N) = \{w \mid N \text{ accepts } w\}$.

NFA acceptance: Example

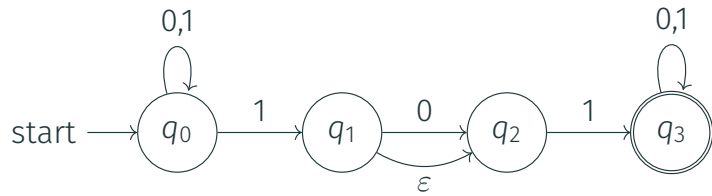


- Is 010110 accepted? *Yes*

NFA acceptance: Wait! what about the ϵ ?!

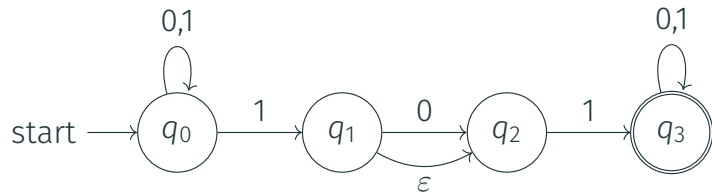


NFA acceptance: Example

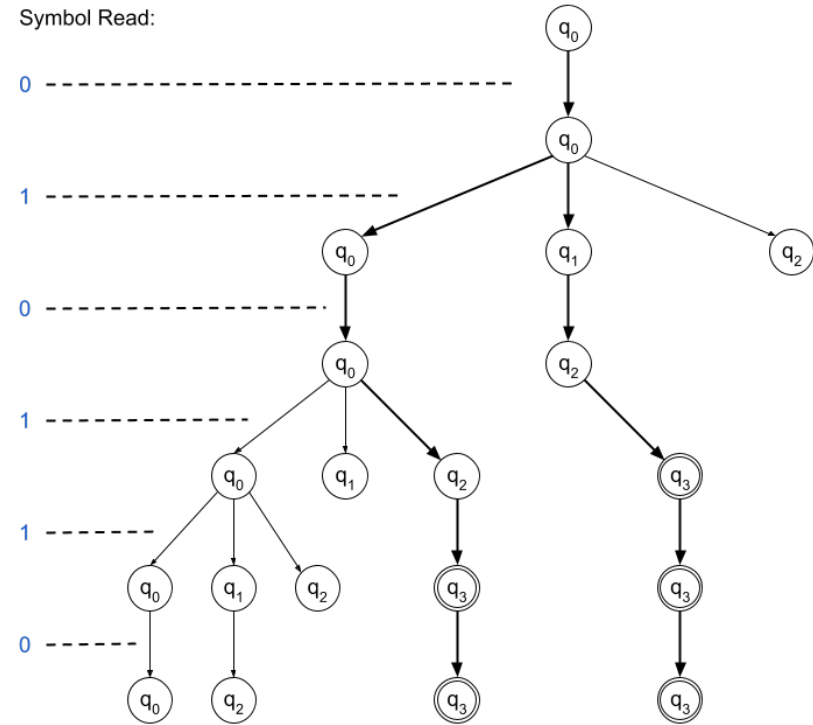


Is 010110 accepted?

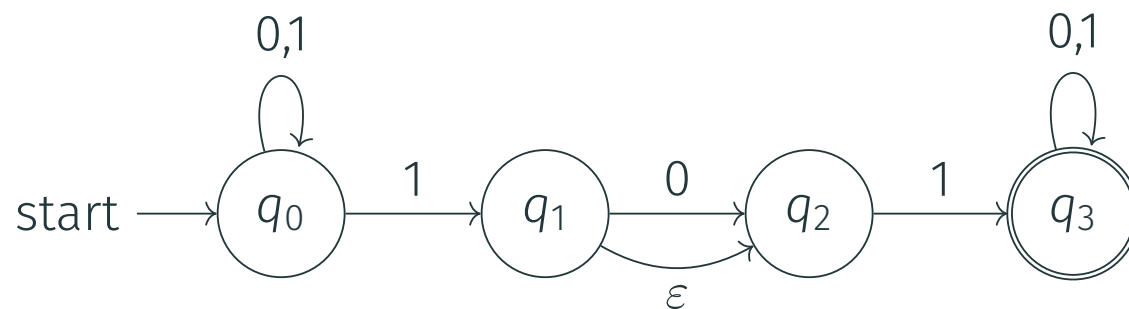
NFA acceptance: Example



Is 010110 accepted?

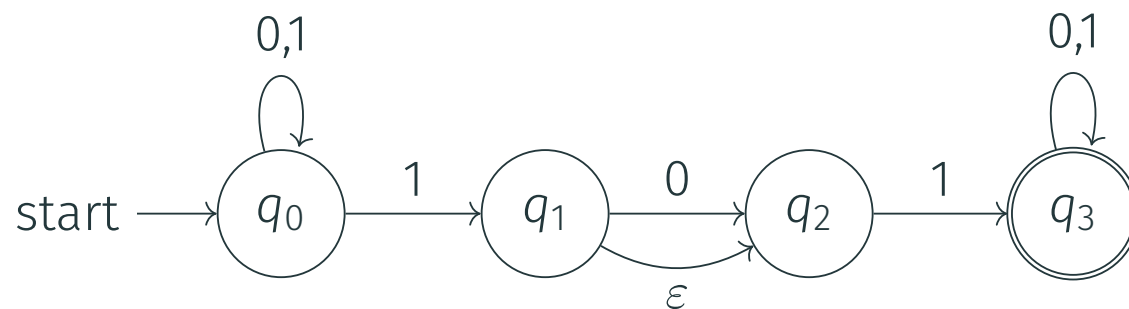


NFA acceptance: Example



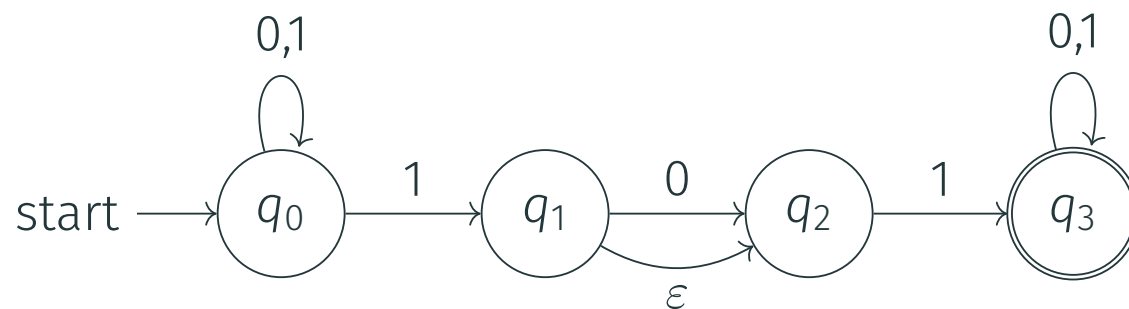
- Is 010110 accepted?

NFA acceptance: Example



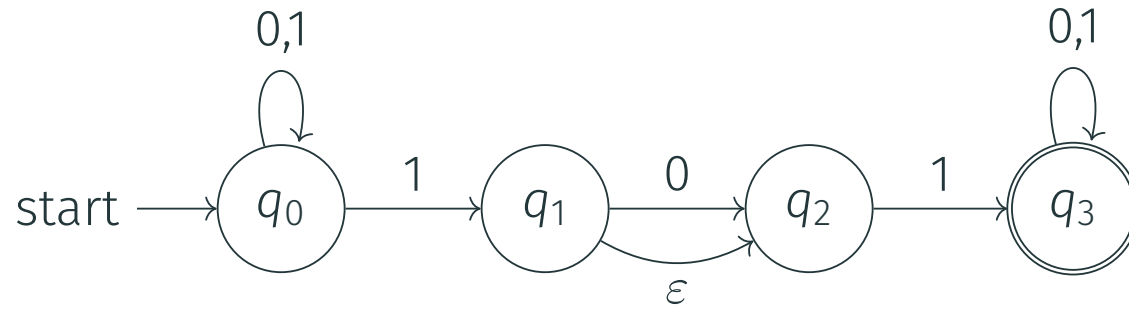
- Is 010110 accepted?
- Is 010 accepted? No

NFA acceptance: Example



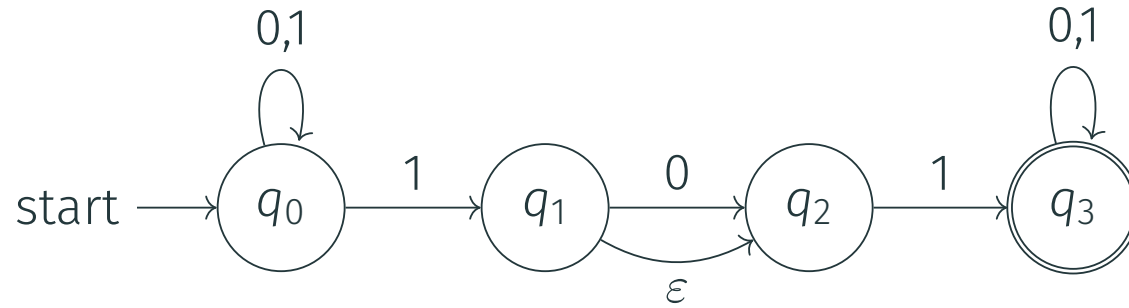
- Is 010110 accepted?
- Is 010 accepted?
- Is 101 accepted? *Yes*

NFA acceptance: Example



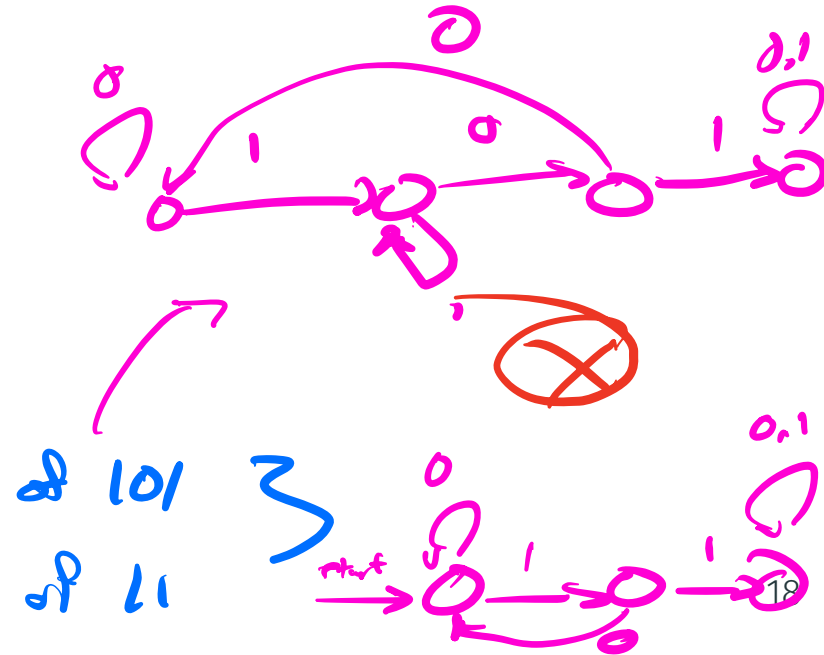
- Is 010110 accepted?
- Is 010 accepted?
- Is 101 accepted?
- Is 10011 accepted? *Yes*

NFA acceptance: Example

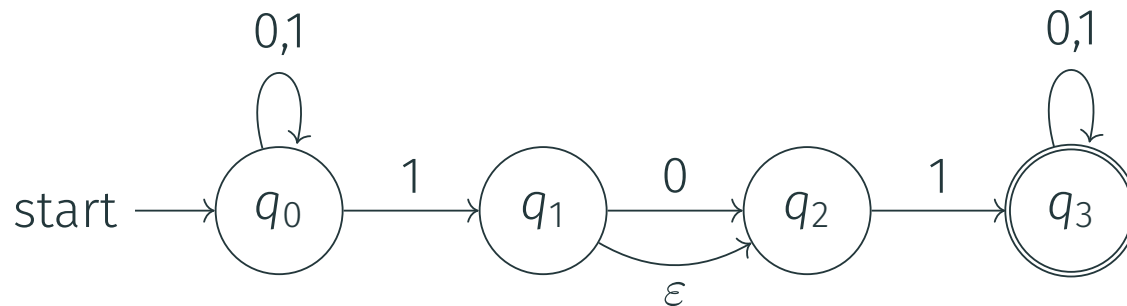


- Is 010110 accepted?
- Is 010 accepted?
- Is 101 accepted?
- Is 10011 accepted?
- What is the language accepted by N?

$L = \{ w \mid w \text{ has substring } 101 \text{ or substring } 11 \}$

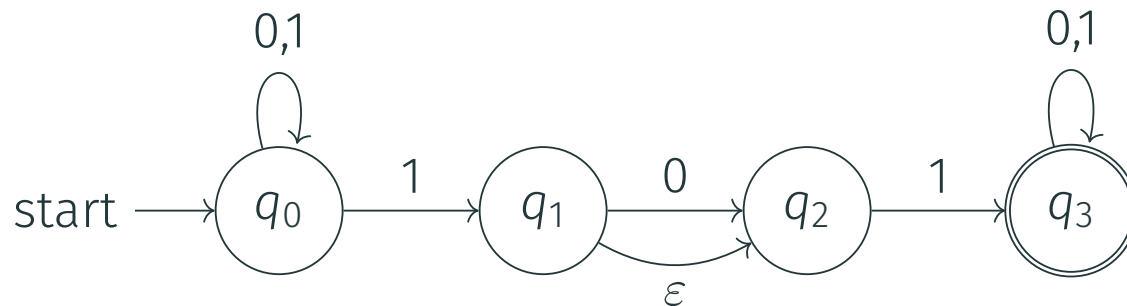


NFA acceptance: Example



- Is 010110 accepted?
- Is 010 accepted?
- Is 101 accepted?
- Is 10011 accepted?
- What is the language accepted by N ?

NFA acceptance: Example



- Is 010110 accepted?
- Is 010 accepted?
- Is 101 accepted?
- Is 10011 accepted?
- What is the language accepted by N ?

Comment: Unlike DFAs, it is easier in NFAs to show that a string is accepted than to show that a string is **not** accepted.

Formal definition of NFA

Formal Tuple Notation

Definition

A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

Formal Tuple Notation

Definition

A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,

Formal Tuple Notation

Definition

A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, \underline{A})$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,

- s 

- A accepts states

Formal Tuple Notation

Definition

A **non-deterministic finite automata (NFA)** $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called **states**,
- Σ is a finite set called the **input alphabet**,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the **transition function** (here $\mathcal{P}(Q)$ is the power set of Q),

Formal Tuple Notation

Definition

A **non-deterministic finite automata (NFA)** $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called **states**,
- Σ is a finite set called the **input alphabet**,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the **transition function** (here $\mathcal{P}(Q)$ is the power set of Q),

$\mathcal{P}(Q)$?

Reminder: Power set

Q : a set. Power set of Q is: $\mathcal{P}(Q) = 2^Q = \{X \mid X \subseteq Q\}$ is set of all subsets of Q .

Example

$$Q = \{1, 2, 3, 4\}$$

$$\mathcal{P}(Q) = \left\{ \begin{array}{c} \{1, 2, 3, 4\}, \\ \{2, 3, 4\}, \{1, 3, 4\}, \{1, 2, 4\}, \{1, 2, 3\}, \\ \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \\ \{1\}, \{2\}, \{3\}, \{4\}, \\ \{\} \end{array} \right\}$$

$$|Q| = n$$

$$|\mathcal{P}(Q)| = 2^n$$

Formal Tuple Notation

Definition

A **non-deterministic finite automata (NFA)** $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called **states**,
- Σ is a finite set called the **input alphabet**,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the **transition function** (here $\mathcal{P}(Q)$ is the power set of Q),

Formal Tuple Notation

Definition

A **non-deterministic finite automata (NFA)** $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called **states**,
- Σ is a finite set called the **input alphabet**,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the **transition function** (here $\mathcal{P}(Q)$ is the power set of Q),
- $s \in Q$ is the **start state**,

Formal Tuple Notation

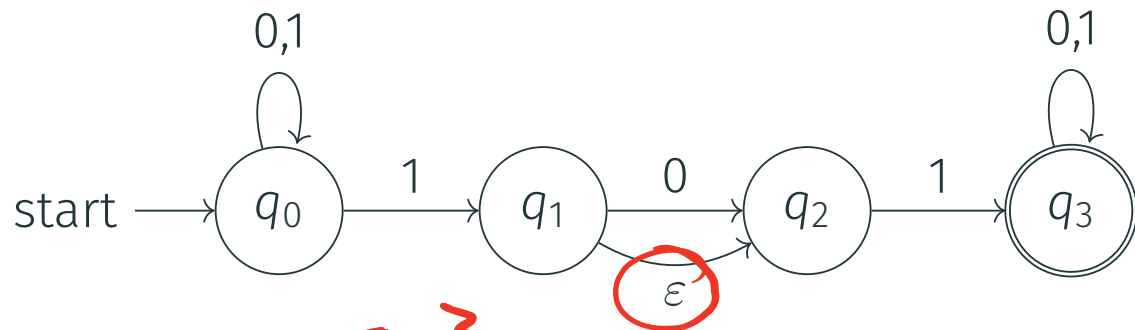
Definition

A **non-deterministic finite automata (NFA)** $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called **states**,
- Σ is a finite set called the **input alphabet**,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the **transition function** (here $\mathcal{P}(Q)$ is the power set of Q),
- $s \in Q$ is the **start state**,
- $A \subseteq Q$ is the set of **accepting/final** states.

$\delta(q, a)$ for $a \in \Sigma \cup \{\varepsilon\}$ is a subset of Q — a set of states.

Example



• $Q = \{q_0, q_1, q_2, q_3\}$

• $\Sigma = \{0, 1\}$

• $\delta =$

$A = \{q_3\}$

	q_0	q_1	q_2	q_3
ϵ	$\{q_0\}$	$\{q_1, q_2\}$	$\{q_2\}$	$\{q_0\}$
0	$\{q_0\}$	$\{q_2\}$	$\{\}$	$\{q_3\}$
1	$\{q_0, q_1\}$	$\{\}$	$\{q_3\}$	$\{q_3\}$

• $S = q_0$

Extending the transition function to strings

Extending the transition function to strings

- NFA $N = (Q, \Sigma, \delta, s, A)$

Extending the transition function to strings

- NFA $N = (Q, \Sigma, \delta, s, A)$
- $\delta(q, a)$: **set of states** that N can go to from q on reading $a \in \Sigma \cup \{\epsilon\}$.

Extending the transition function to strings

- NFA $N = (Q, \Sigma, \delta, s, A)$
- $\delta(q, a)$: set of states that N can go to from q on reading $a \in \Sigma \cup \{\epsilon\}$.
- Want transition function $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$

Extending the transition function to strings

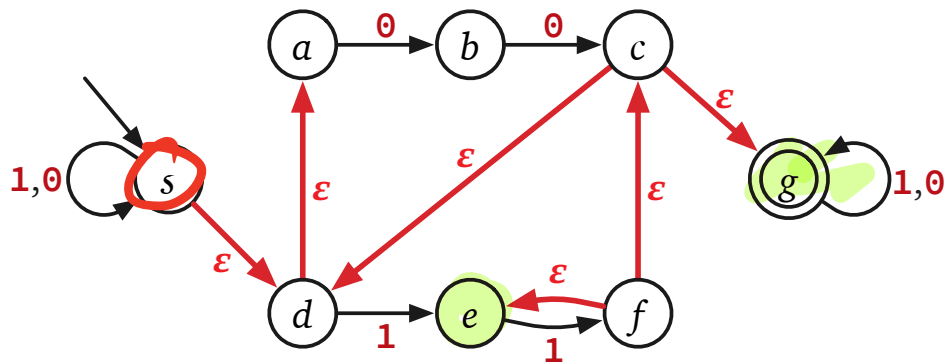
- NFA $N = (Q, \Sigma, \delta, s, A)$
- $\delta(q, a)$: set of states that N can go to from q on reading $a \in \Sigma \cup \{\epsilon\}$.
- Want transition function $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$
- $\delta^*(q, w)$: set of states reachable on input w starting in state q .

Extending the transition function to strings

Definition

For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\epsilon\text{reach}(q)$ is the set of all states that q can reach using only ϵ -transitions.

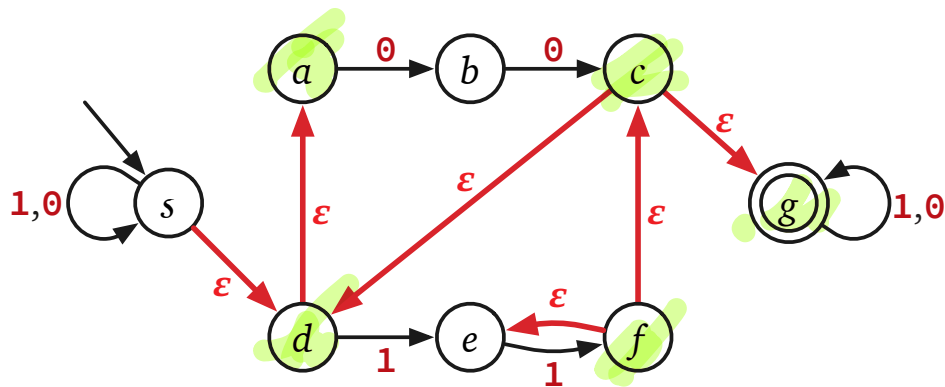
001



Extending the transition function to strings

Definition

For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\epsilon\text{reach}(q)$ is the set of all states that q can reach using only ϵ -transitions.



Definition

For $X \subseteq Q$: $\epsilon\text{reach}(X) = \bigcup_{x \in X} \epsilon\text{reach}(x)$.

$\epsilon\text{reach}(s) = \{s, d, a\}$
 $\epsilon\text{reach}(P)$

Extending the transition function to strings

$\epsilon\text{reach}(q)$: set of all states that q can reach using only ϵ -transitions.

Definition

Inductive definition of $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

- if $w = \epsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$

Extending the transition function to strings

$\epsilon\text{reach}(q)$: set of all states that q can reach using only ϵ -transitions.

Definition

Inductive definition of $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

- if $w = \epsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$
- if $w = a$ where $a \in \Sigma$:
$$\delta^*(q, a) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a)\right)$$

Extending the transition function to strings

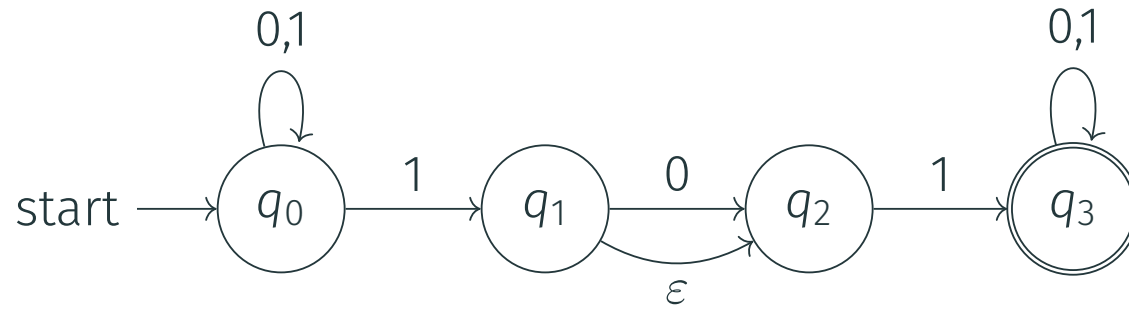
$\epsilon\text{reach}(q)$: set of all states that q can reach using only ϵ -transitions.

Definition

Inductive definition of $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

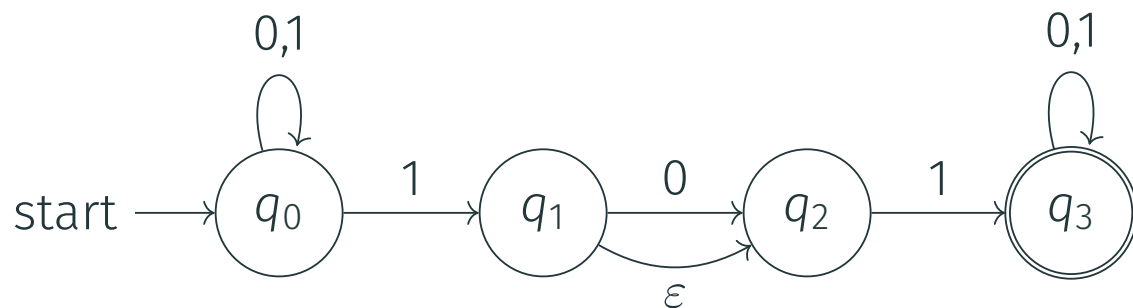
- if $w = \epsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$
- if $w = a$ where $a \in \Sigma$:
$$\delta^*(q, a) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a)\right)$$
- if $w = ax$:
$$\delta^*(q, w) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \left(\bigcup_{r \in \delta^*(p, a)} \delta^*(r, x)\right)\right)$$

Example of extended transition function



Find $\delta^*(q_0, 11)$:

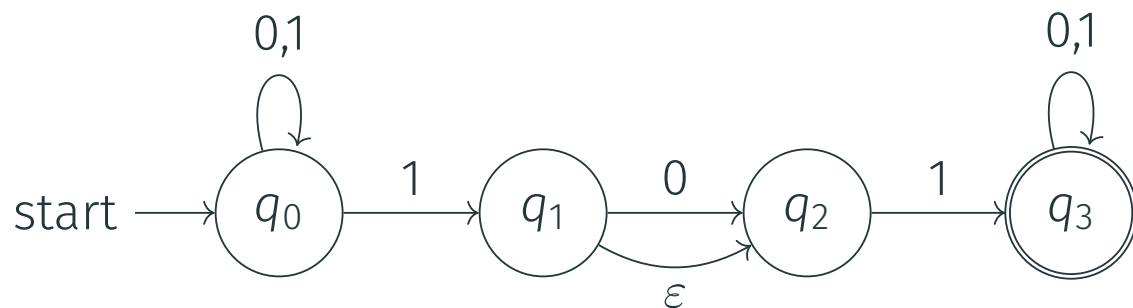
Example of extended transition function



Find $\delta^*(q_0, 11)$:

$$\delta^*(q, w) = \epsilon\text{reach} \left(\bigcup_{p \in \epsilon\text{reach}(q)} \left(\bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right) \right)$$

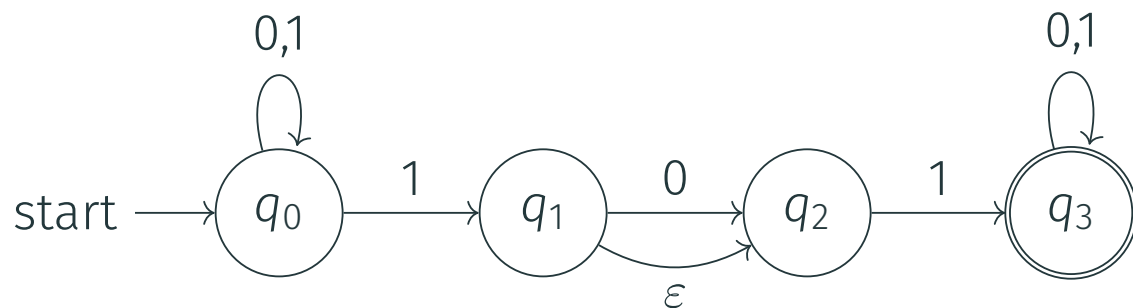
Example of extended transition function



We know $w = 11 = ax$ so $a = 1$ and $x = 1$

$$\delta^*(q_0, 11) = \epsilon\text{reach} \left(\bigcup_{p \in \epsilon\text{reach}(q_0)} \left(\bigcup_{r \in \delta^*(p, 1)} \delta^*(r, 1) \right) \right)$$

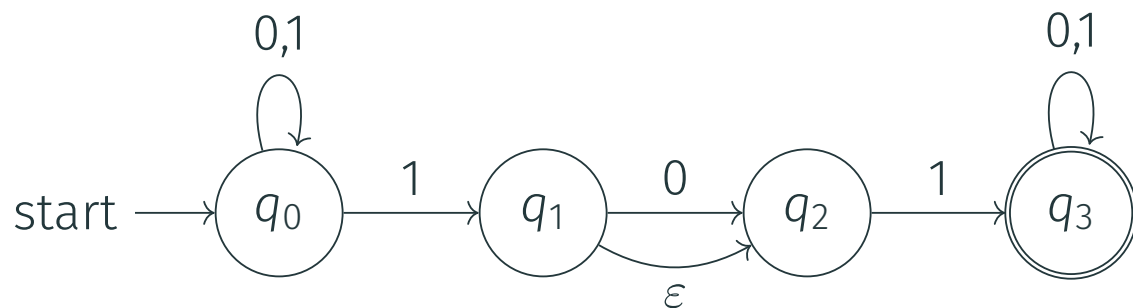
Example of extended transition function



$$\epsilon\text{reach}(q_0) = \{q_0\}$$

$$\delta^*(q_0, \textcolor{blue}{11}) = \epsilon\text{reach} \left(\bigcup_{p \in \{q_0\}} \left(\bigcup_{r \in \delta^*(p, \textcolor{blue}{1})} \delta^*(r, \textcolor{blue}{1}) \right) \right)$$

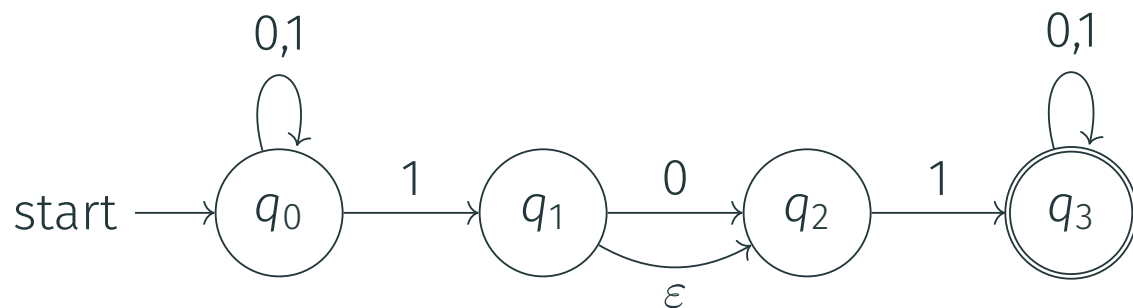
Example of extended transition function



Simplify:

$$\delta^*(q_0, \mathbf{11}) = \epsilon\text{reach} \left(\bigcup_{r \in \delta^*({q_0}, \mathbf{1})} \delta^*(r, \mathbf{1}) \right)$$

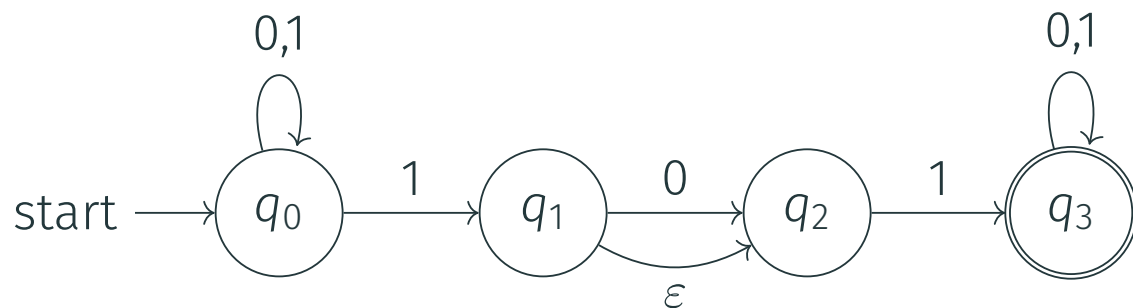
Example of extended transition function



Need $\delta^*(q_0, \mathbf{1}) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a)\right) = \epsilon\text{reach}(\delta(q_0, \mathbf{1}))$:
 $= \epsilon\text{reach}(\{q_0, q_1\}) = \{q_0, q_1, q_2\}$

$$\delta^*(q_0, \mathbf{11}) = \epsilon\text{reach}\left(\bigcup_{r \in \delta^*(\{q_0\}, \mathbf{1})} \delta^*(r, \mathbf{1})\right)$$

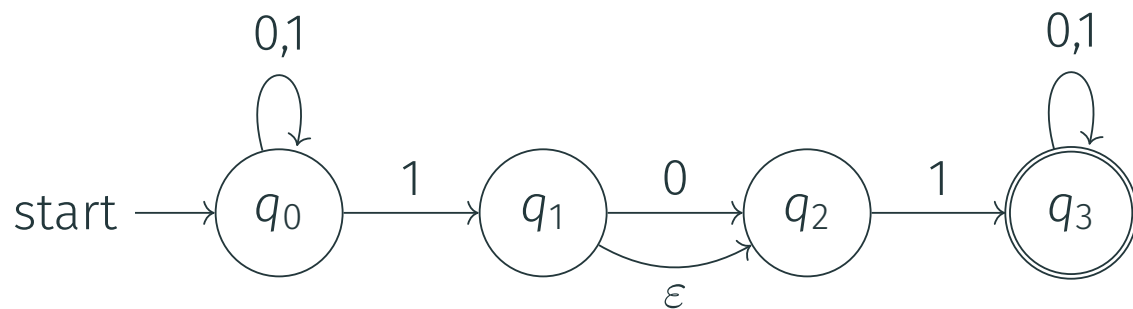
Example of extended transition function



Need $\delta^*(q_0, \mathbf{1}) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a)\right) = \epsilon\text{reach}(\delta(q_0, \mathbf{1}))$:
 $= \epsilon\text{reach}(\{q_0, q_1\}) = \{q_0, q_1, q_2\}$

$$\delta^*(q_0, \mathbf{11}) = \epsilon\text{reach}\left(\bigcup_{r \in \{q_0, q_1, q_2\}} \delta^*(r, \mathbf{1})\right)$$

Example of extended transition function



Simplify

$$\delta^*(q_0, 11) = \epsilon\text{reach}(\delta^*(q_0, 1) \cup \delta^*(q_1, 1) \cup \delta^*(q_2, 1))$$

Transition for strings: $w = ax$

$$\delta^*(q, w) = \epsilon\text{reach} \left(\bigcup_{p \in \epsilon\text{reach}(q)} \left(\bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right) \right)$$

- $R = \epsilon\text{reach}(q) \implies \delta^*(q, w) = \epsilon\text{reach} \left(\bigcup_{p \in R} \bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right)$
- $N = \bigcup_{p \in R} \delta^*(p, a)$: All the states reachable from q with the letter a .
- $\delta^*(q, w) = \epsilon\text{reach} \left(\bigcup_{r \in N} \delta^*(r, x) \right)$

Formal definition of language accepted by **N**

Definition

A string w is accepted by NFA N if $\delta_N^*(s, w) \cap A \neq \emptyset$.

Definition

The language $L(N)$ accepted by a NFA $N = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \cap A \neq \emptyset\}.$$

Formal definition of language accepted by **N**

Definition

A string w is accepted by NFA N if $\delta_N^*(s, w) \cap A \neq \emptyset$.

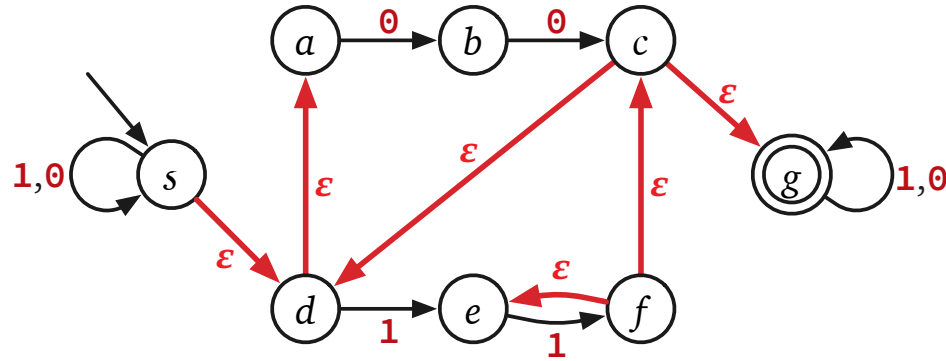
Definition

The language $L(N)$ accepted by a NFA $N = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \cap A \neq \emptyset\}.$$

Important: Formal definition of the language of NFA above uses δ^* and not δ . As such, one does not need to include ε -transitions closure when specifying δ , since δ^* takes care of that.

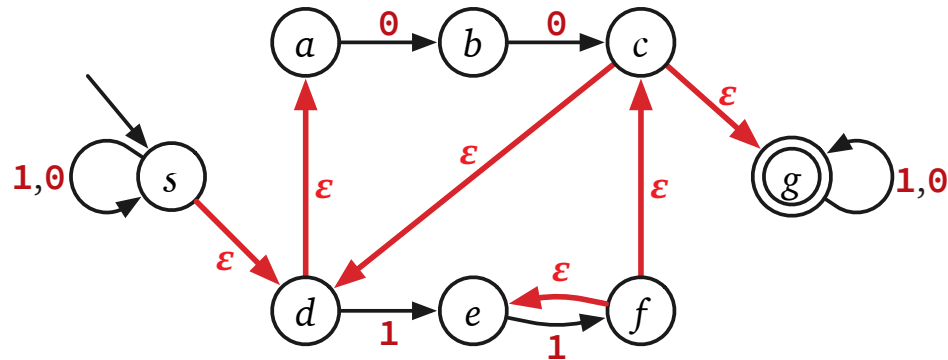
Example



What is:

- $\delta^*(s, \epsilon) =$

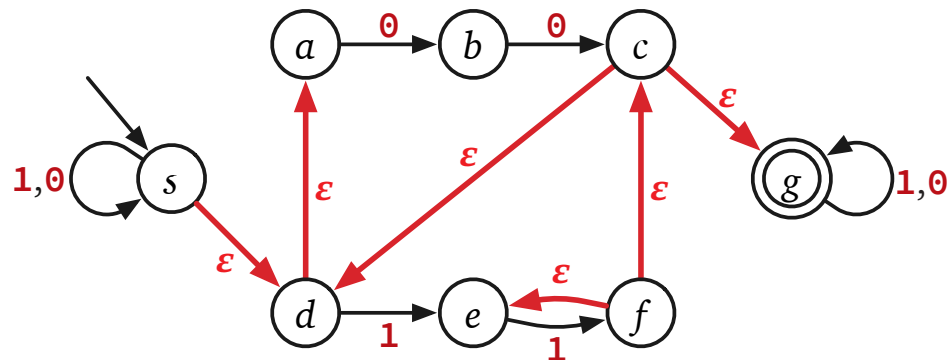
Example



What is:

- $\delta^*(s, \epsilon) =$
- $\delta^*(s, 0) =$

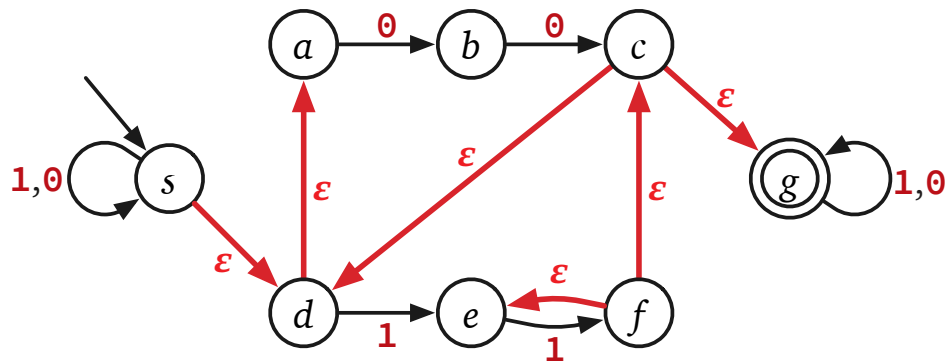
Example



What is:

- $\delta^*(s, \epsilon) =$
- $\delta^*(s, 0) =$
- $\delta^*(b, 0) =$

Example



What is:

- $\delta^*(s, \epsilon) =$
- $\delta^*(s, 0) =$
- $\delta^*(b, 0) =$
- $\delta^*(b, 00) =$

Constructing generalized NFAs

DFAs and NFAs

- Every DFA is a NFA so NFAs are at least as powerful as DFAs.
- NFAs prove ability to “guess and verify” which simplifies design and reduces number of states
- Easy proofs of some closure properties

Example

$L = \{\text{bitstrings that have a 1 three positions from the end}\}$



A simple transformation

Theorem

For every NFA N there is another NFA N' such that $L(N) = L(N')$ and such that N' has the following two properties:

- N' has single final state f that has no outgoing transitions*
- The start state s of N is different from f*

A simple transformation

Theorem

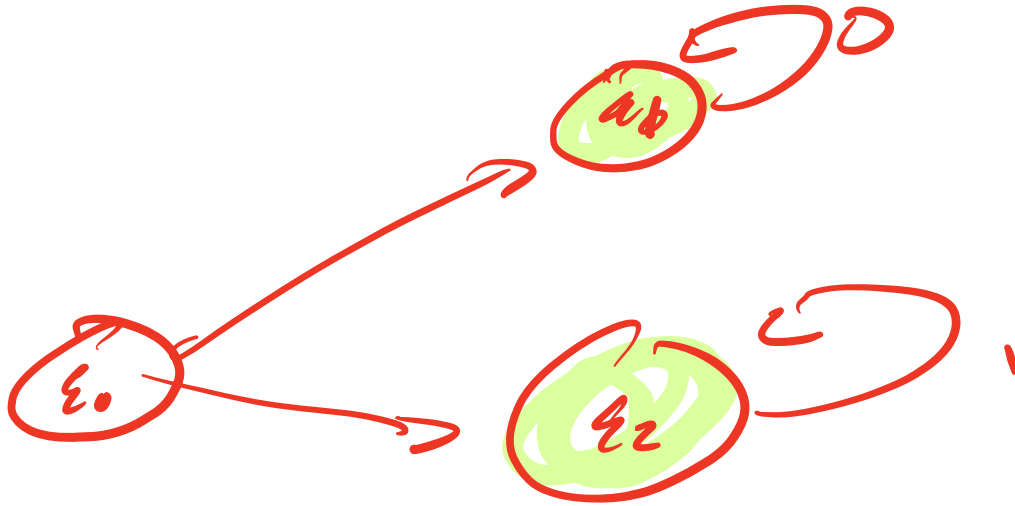
For every NFA N there is another NFA N' such that $L(N) = L(N')$ and such that N' has the following two properties:

- N' has single final state f that has no outgoing transitions*
- The start state s of N is different from f*

Why couldn't we say this for DFA's?

A simple transformation

Hint: Consider the $L = 0^* + 1^*$.



Closure Properties of NFAs

Closure properties of NFAs

Are the class of languages accepted by NFAs closed under the following operations?

- union
- intersection
- concatenation
- Kleene star
- complement

Closure under union

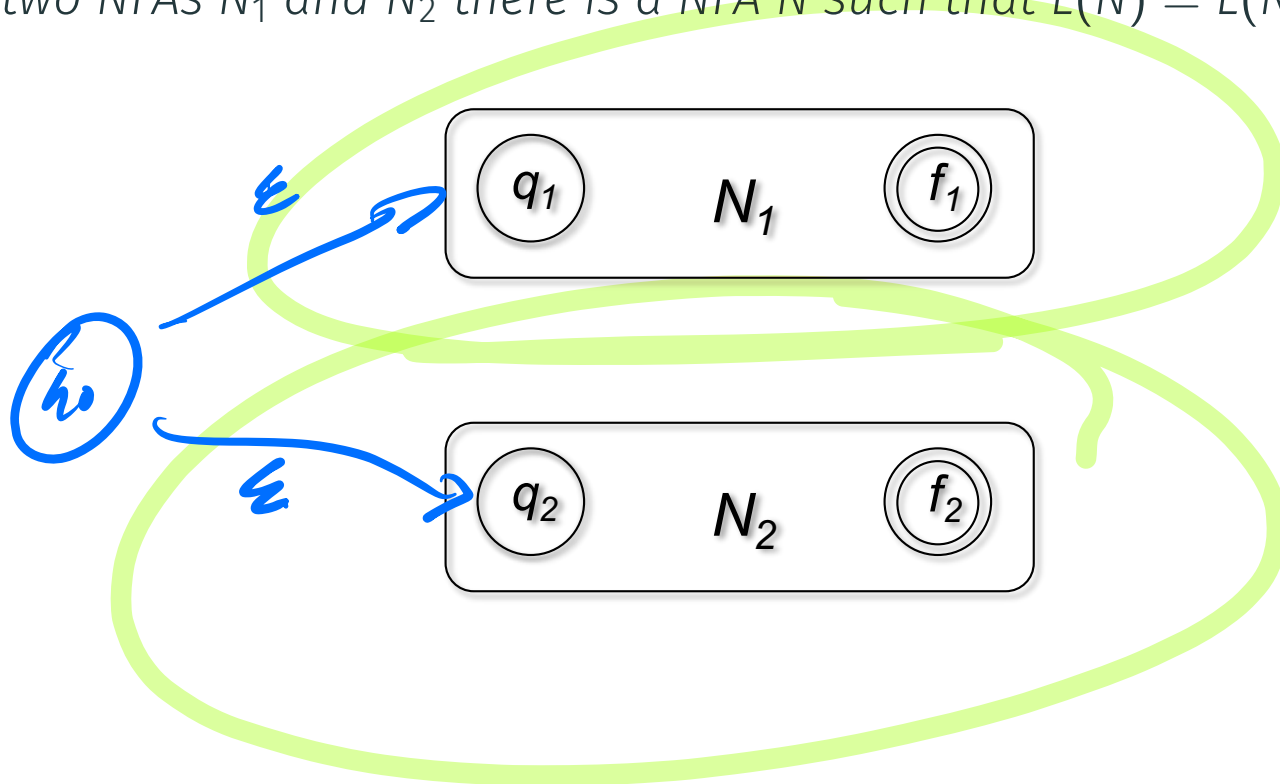
Theorem

For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \cup L(N_2)$.

Closure under union

Theorem

For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \cup L(N_2)$.



Closure under concatenation

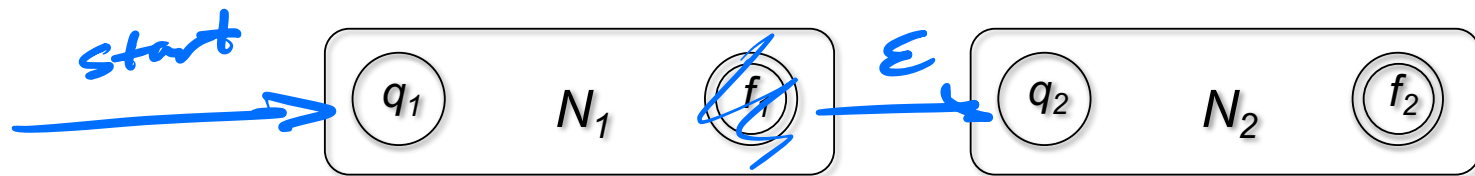
Theorem

For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \cdot L(N_2)$.

Closure under concatenation

Theorem

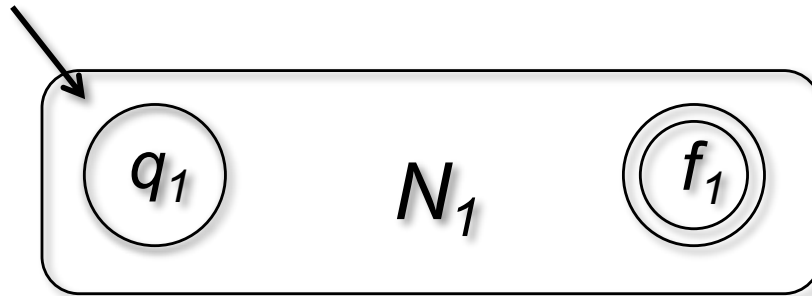
For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \cdot L(N_2)$.



Closure under Kleene star

Theorem

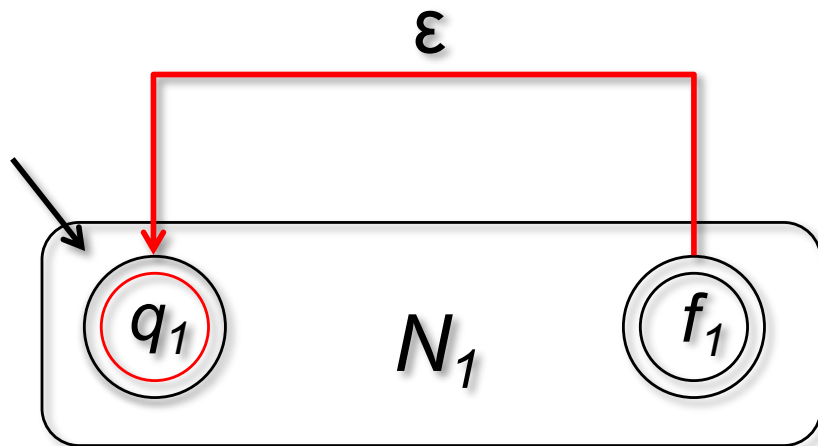
For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.



Closure under Kleene star

Theorem

For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.

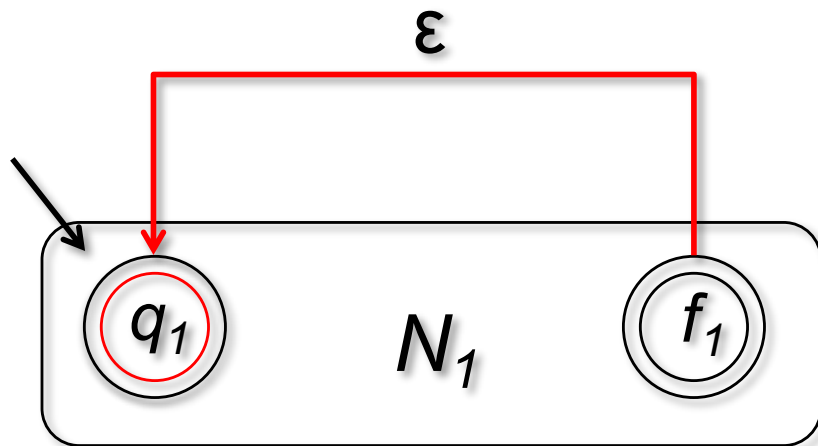


Why
not?

Closure under Kleene star

Theorem

For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.

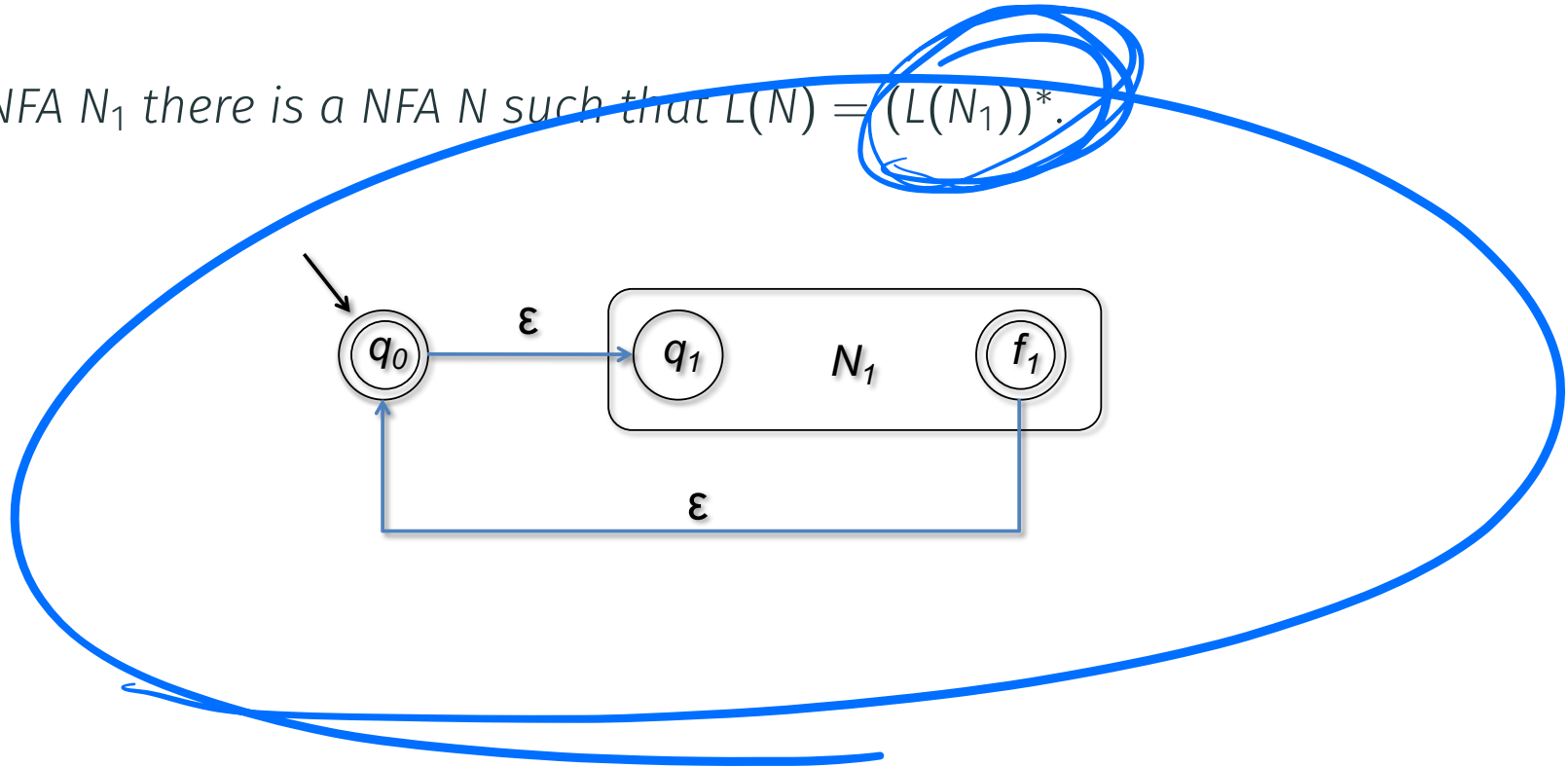


Does not work! Why?

Closure under Kleene star

Theorem

For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.



Transformations

All these examples are examples of language *transformations*.

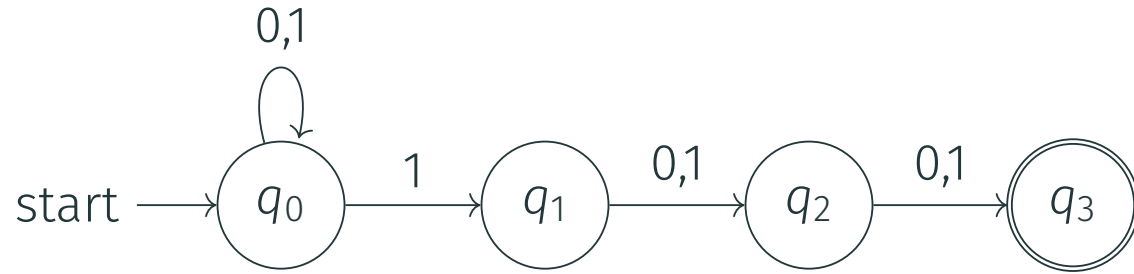
A language transformation is one where you take one class or languages, perform some operation and get a new language **that belongs to that same class (closure)**.

Tomorrow's lab will go over more examples of language transformations.

Last thought

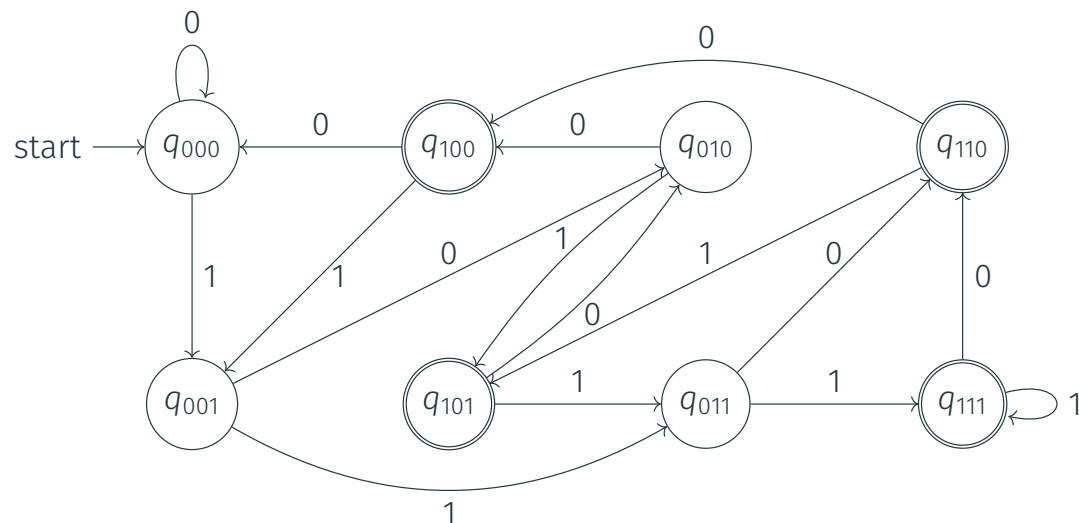
Equivalence

Do all NFAs have a corresponding DFA?



Equivalence

Do all NFAs have a corresponding DFA?



Yes but it likely won't be pretty.