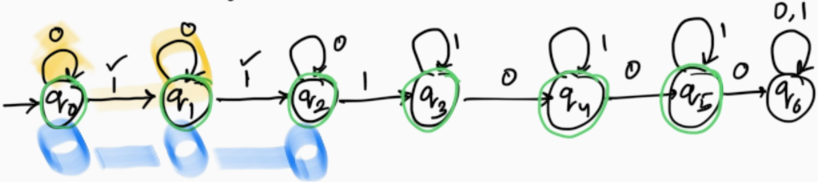
Pre-lecture brain teaser

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

Idea: How do you construct a DFA for this?



10101000

RE: $0^* + 0^*10^* + 0^*10^*10^* + \cdots$

ECE-374-B: Lecture 3 - NFAs

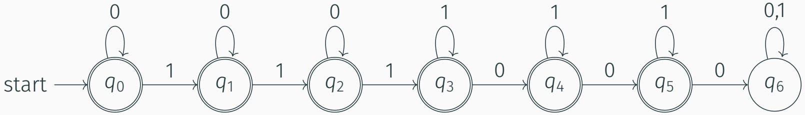
Instructor: Abhishek Kumar Umrawal

January 25, 2024

University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

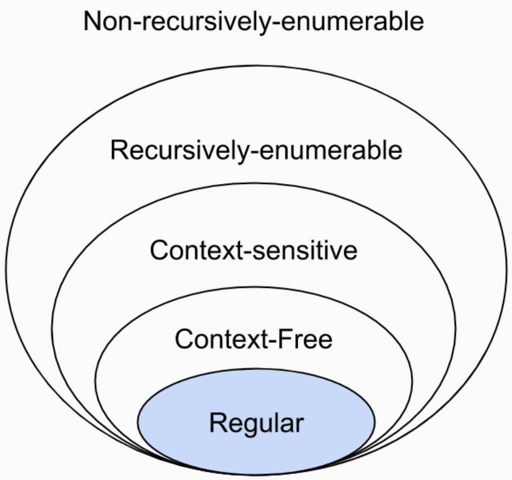
Find the regular expression for the language containing all binary strings that **do not** contain the subsequence 111000



0* + 0*10* + 0*10*10* + 0*10*10*1* + 0*10*10*101* + 0*10*10*101*01*

Tangential Thought

Does luck allow us to solve unsolvable problems?



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.

Lucky machine programs

Problem: Find shortest path from a to b

Program on M_1 (Dijkstra's algorithm):

```
Initialize for each node v, Dist(s,v) = d'(s,v) = ∞
Initialize X = ∅, d'(s,s) = 0
for i = 1 to |V| do
    Let v be node realizing d'(s,v) = min_{u∈V−X} d'(s,u)
    Dist(s,v) = d'(s,v)
    X = X ∪ {v}
    Update d'(s,u) for each u in V − X as follows:
        d'(s,u) = min(d'(s,u), Dist(s,v) + ℓ(v,u))
```

Lucky machine programs

Problem: Find shortest path from a to b

Program on M_2 (Blind luck):

```
path = []
current = a
While(not at b)
    take an outgoing edge from current node
    current = new location
    path += current
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.

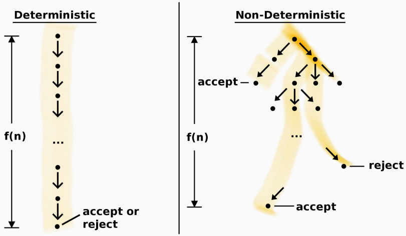
Question: Are there problems which M_2 can solve that M_1 cannot.

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.



Why non-determinism?

RIY

- 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

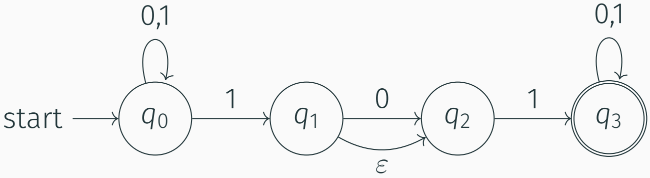
Non-deterministic Finite State Automata by example

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

NFA

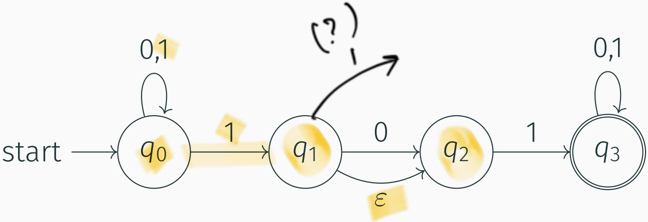
When you come to a fork in the road, take it.

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



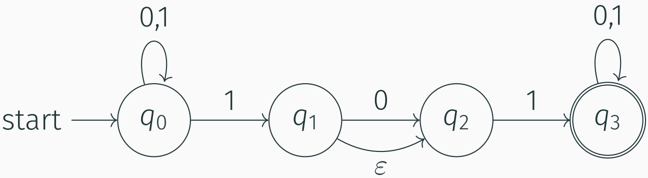
NFA acceptance: Informal

NFA



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

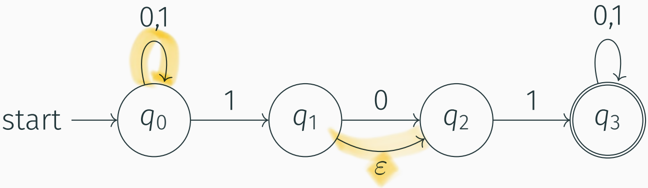
NFA acceptance: Informal

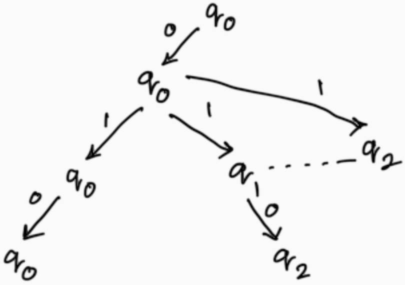


Informal definition: An NFA N accepts a string w iff some accepting state is reached by N from the start state on input w.

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

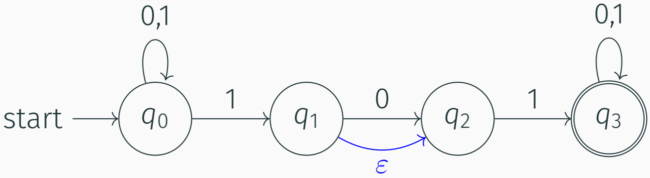
NFA acceptance: Example



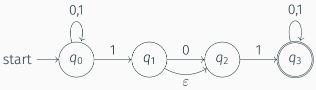


- Is 010110 accepted?

NFA acceptance: Wait! what about the ϵ?!

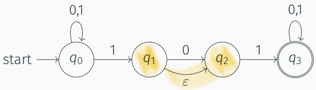


NFA acceptance: Example



Is 010110 accepted?

NFA acceptance: Example

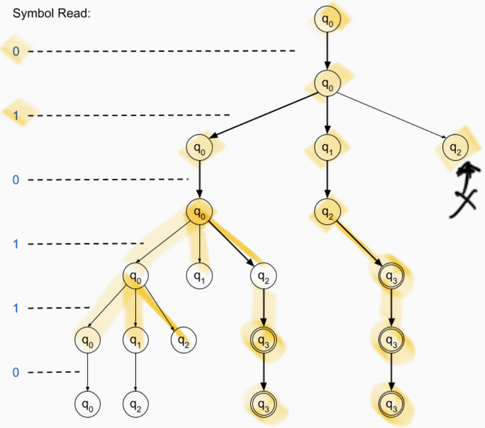


Is 010110 accepted? YES

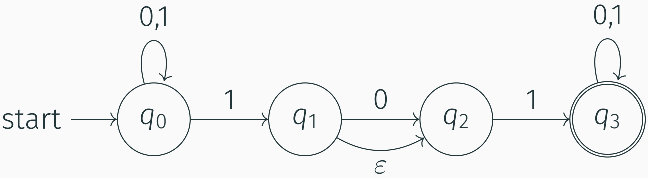
$q_0 \xrightarrow{1} q_1 \xrightarrow{\epsilon}$

$q_0 \xrightarrow{1} q_2$

$01\epsilon01 = 0101$

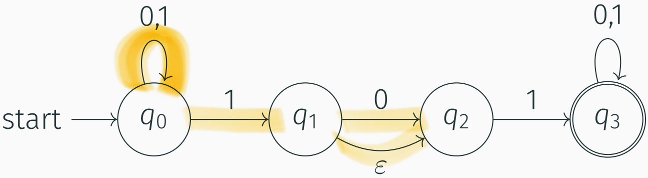


NFA acceptance: Example

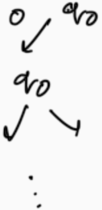


- Is 010110 accepted? **Yes**

NFA acceptance: Example

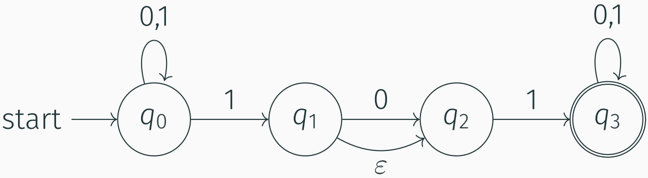


- Is 010110 accepted? **Yes**
- Is 010 accepted? **No**



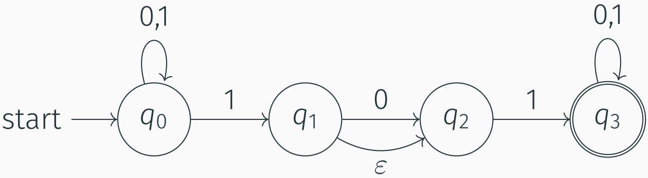
Practice!

NFA acceptance: Example



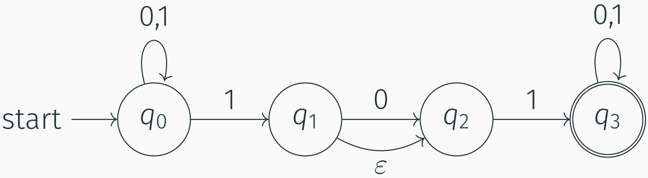
- Is 010110 accepted? **Yes**
- Is 010 accepted? **No**
- Is 101 accepted? **Yes**

NFA acceptance: Example



- Is 010110 accepted? **Yes**
- Is 010 accepted? **No**
- Is 101 accepted? **Yes**
- Is 10011 accepted? **Yes**

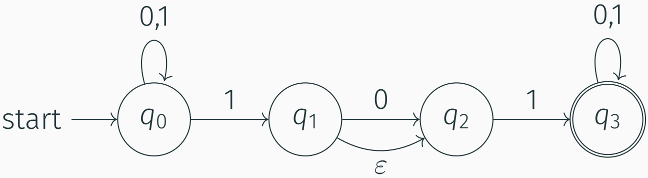
NFA acceptance: Example



- Is 010110 accepted? **Yes**
- Is 010 accepted? **No**
- Is 101 accepted? **Yes**
- Is 10011 accepted? **Yes**
- What is the language accepted by N? **All strings with 101 or 11 as a sub string.**

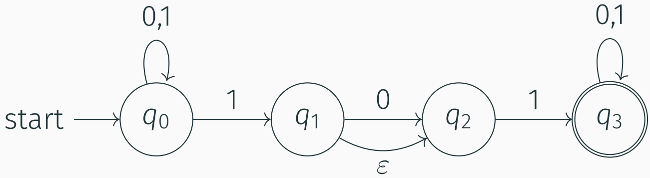
→ Think about it!

NFA acceptance: Example



- Is 010110 accepted? **Yes**
- Is 010 accepted? **No**
- Is 101 accepted? **Yes**
- Is 10011 accepted? **Yes**
- What is the language accepted by N? **All strings with 101 or 11 as a sub string.**

NFA acceptance: Example



- Is 010110 accepted? **Yes**
- Is 010 accepted? **No**
- Is 101 accepted? **Yes**
- Is 10011 accepted? **Yes**
- What is the language accepted by N? **All strings with 101 or 11 as a sub string.**

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) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

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

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,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \to \mathcal{P}(Q)$ is the transition function (here $\mathcal{P}(Q)$ is the power set of Q),

$$\delta : Q \times \Sigma \cup \{\epsilon\} \longrightarrow P(Q)$$

Power set: The set of all possible subsets of a set

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,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \to \mathcal{P}(Q)$ 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
$Q = \{1, 2, 3, 4\}$

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

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,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \to \mathcal{P}(Q)$ 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,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \to \mathcal{P}(Q)$ is the transition function (here $\mathcal{P}(Q)$ is the power set of Q),
- $s \in Q$ is the start state,

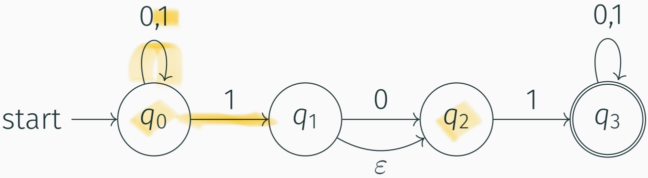
Formal Tuple Notation

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,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \to \mathcal{P}(Q)$ 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



- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $\delta =$

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

(?) ✓

- $s = q_0$
- $A = \{q_3\}$

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 $a \in \Sigma \cup \{\varepsilon\}$.

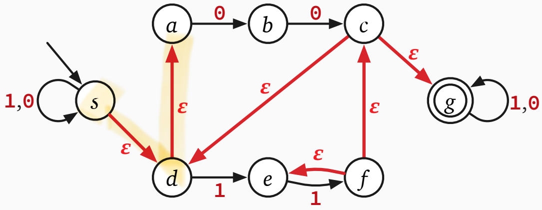
Extending the transition function to strings

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

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 $a \in \Sigma \cup \{\varepsilon\}$.
- Want transition function $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$
- $\delta^*(q, w)$: set of states reachable on input w starting in state q.

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

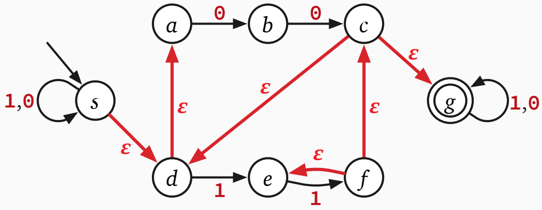


ε-reach(q)

ε-reach(s) = { d, a } ∪ {s}

Extending the transition function to strings

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



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

$X = \{s, d\}$ ϵreach$(X) =$ ϵreach$(s) \cup \epsilon$reach(d)

Extending the transition function to strings

ϵreach(q): set of all states that q can reach using only ε-transitions.

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

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

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 $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$:

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

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

$\delta : Q \times \Sigma \to P(Q)$

Extending the transition function to strings

ϵreach(q): set of all states that q can reach using only ε-transitions.

Power set of Q

$\delta : Q \times \Sigma \to P(Q)$

Definition

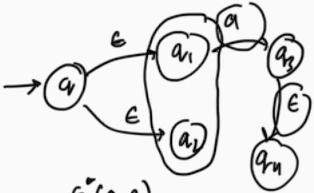
Inductive definition of $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$:

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

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

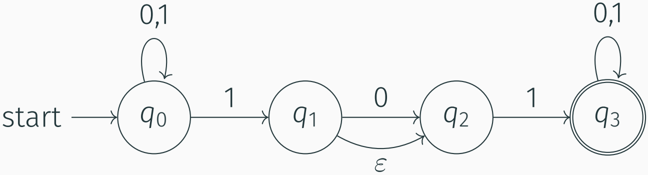
- if $w = ax$:

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



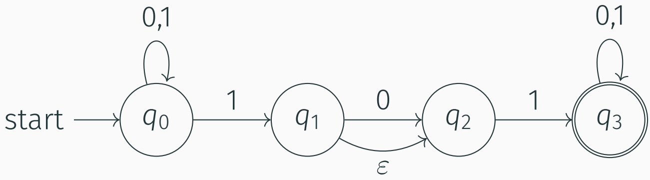
$\delta^*(q, a)$

Example of extended transition function



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

Example of extended transition function



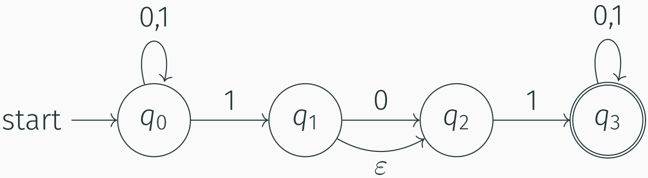
$w = ax \Rightarrow a=1,\ x=1$

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

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

$q \to q_0$, $w \to 11$, $q \to q_0$, $a \to 1$, $x \to 1$

Example of extended transition function

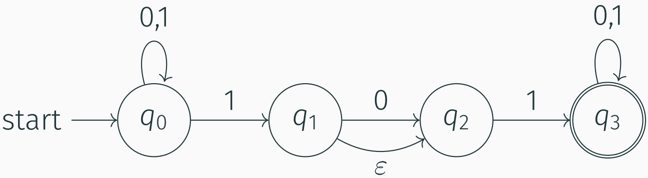


We know $w = 11 = ax$ so $a = 1$ and $x = 1$

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

$\{q_0\}$

Example of extended transition function

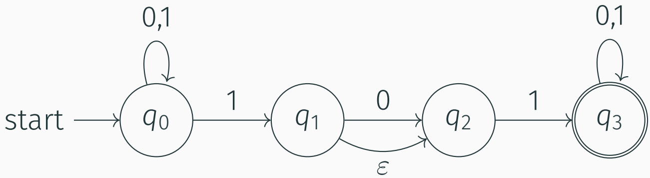


$$\epsilon\text{reach}(q_0) = \{q_0\}$$

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

q_0

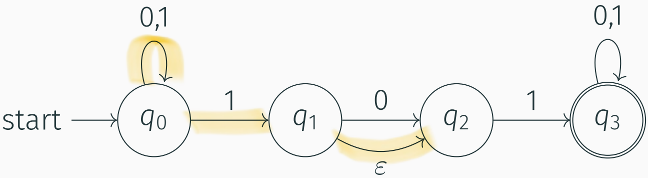
Example of extended transition function



Simplify:

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

Example of extended transition function

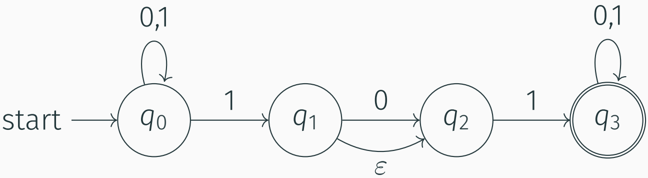


Need $\delta^*(q_0, 1) = \epsilon\text{reach}\left(\bigcup_{p\in\epsilon\text{reach}(q)} \delta(p, a)\right) = \epsilon\text{reach}(\delta\,(q_0, 1))$:

$= \epsilon\text{reach}(\{q_0, q_1\}) = \{q_0, q_1, q_2\}$

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

Example of extended transition function



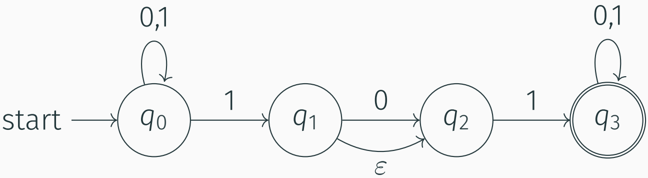
Need

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

$= \epsilon\text{reach}(\{q_0, q_1\}) = \{q_0, q_1, q_2\}$

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

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

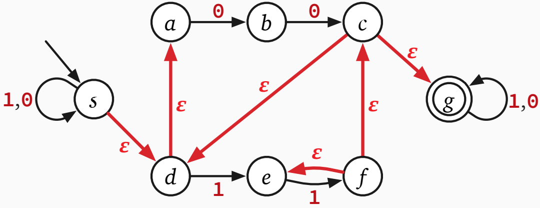
Definition
A string w is accepted by NFA N if $\delta_N^*(s, w) \cap A \neq \emptyset$.

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.

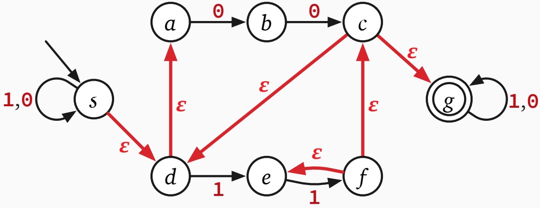
Example



What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$

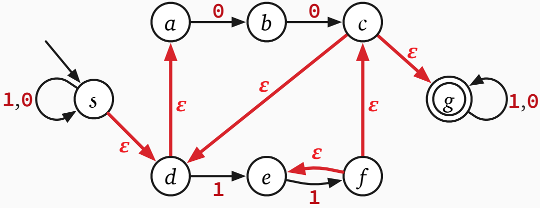
Example



What is:

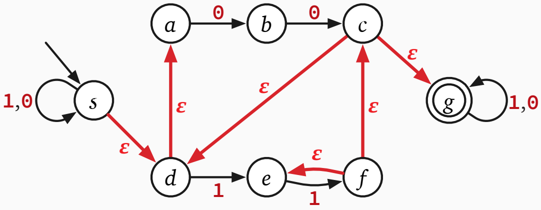
- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) = \{s, d, a, b\}$

Example



What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) = \{s, d, a, b\}$
- $\delta^*(b, 0) = \{d, a, c, g\}$



What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) = \{s, d, a, b\}$
- $\delta^*(b, 0) = \{d, a, c, g\}$
- $\delta^*(b, 00) = \{b, g\}$

Constructing generalized NFAs

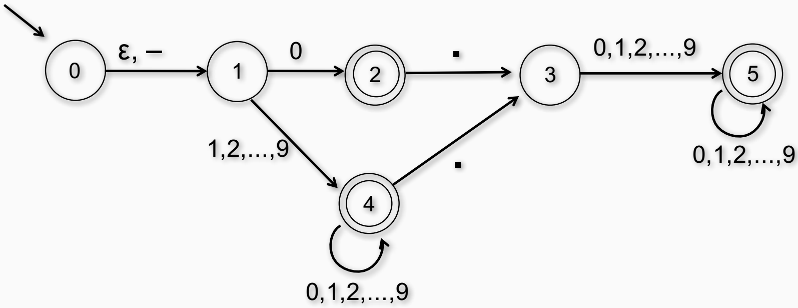
DFAs and NFAs

- 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

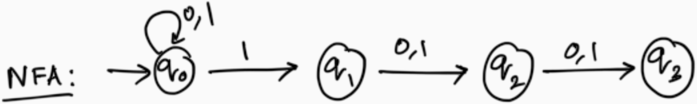
Strings that represent decimal numbers.

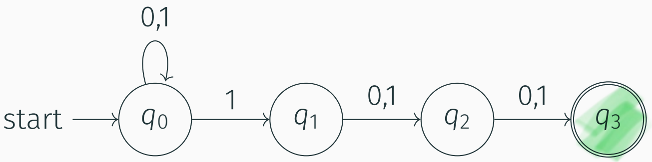
Examples: 154, 345.75332, 534677567.1



Example

$L = \{$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*

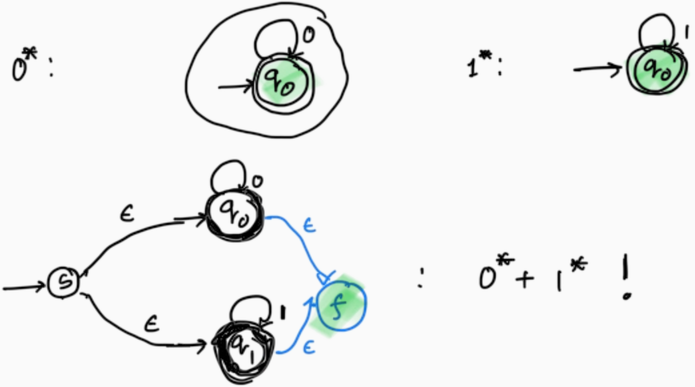
Theorem
For every NFA N there is another NFA N' such that $L(N) = L(N')$ and such that N' has the following two properties:

- *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 = $0^* + 1^*$.



{q0, q1} : non-accepting state

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

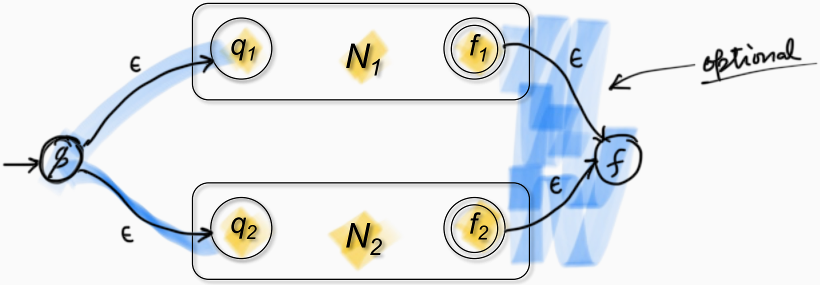
Theorem
For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \cup L(N_2)$.

$N_1 \rightarrow L(N_1) \qquad N_2 \longrightarrow L(N_2)$

$L(N_1) \cup L(N_2) \overset{?}{\longrightarrow} N$

Closure under union

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



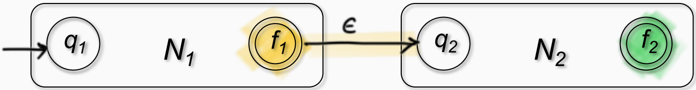
True. Single start state with ε-transition to q_1 and q_2.

Closure under concatenation

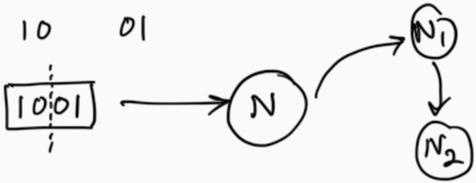
Theorem
For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \bullet L(N_2)$.

Closure under concatenation

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

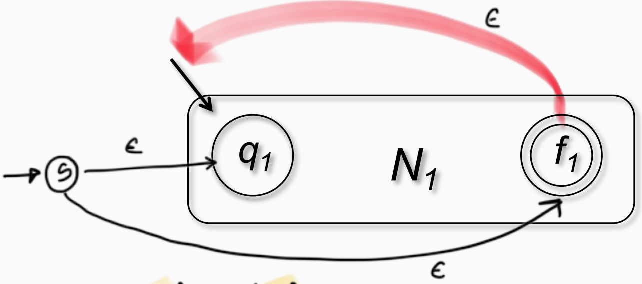


True. f_1 connected to q_2 and f_2 as accept state.



Closure under Kleene star

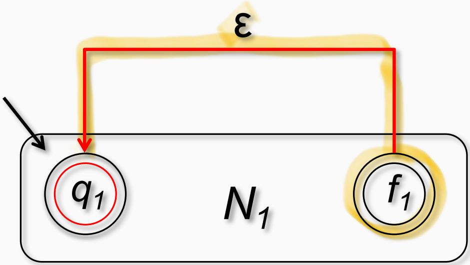
Theorem
For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.



Eg. $L(N) = \{01\}$

$(L(N))^* = \{\epsilon, 01, 0101, 010101, \ldots\}$

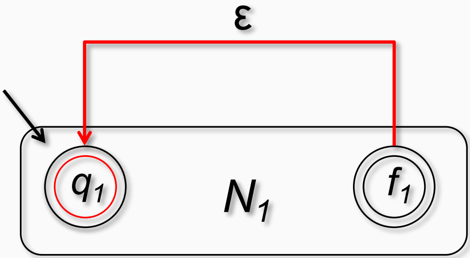
Theorem
For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.



Why does this not work?

Closure under Kleene star

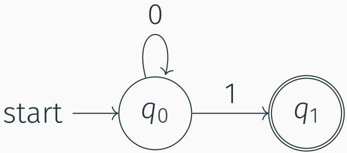
Theorem
For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.



Does not work! Why? Think about it!

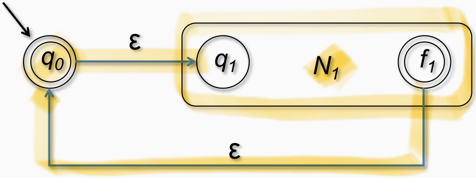
Closure under Kleene star instructors note

Because Kleene star must include εbut if we turn the initial state into an accept state, we are inserting a εinto $L(N)$ where there might not be one. imagine:



Closure under Kleene star

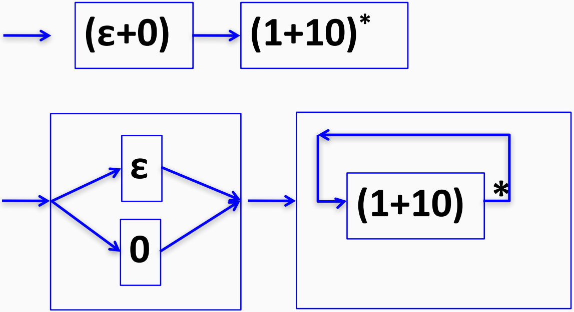
Theorem
For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.



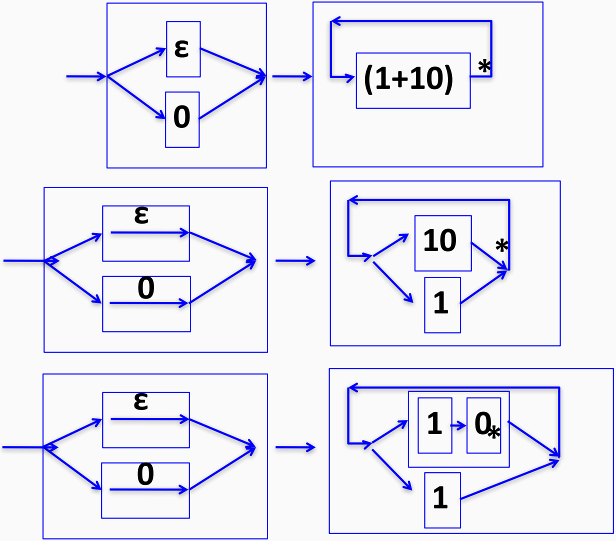
NFAs capture Regular Languages

Example

$(\varepsilon+0)(1+10)^*$

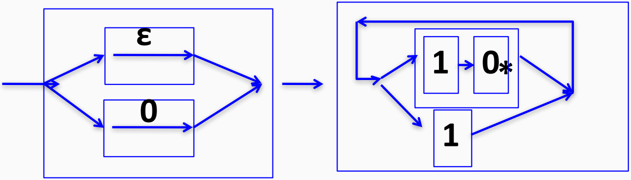


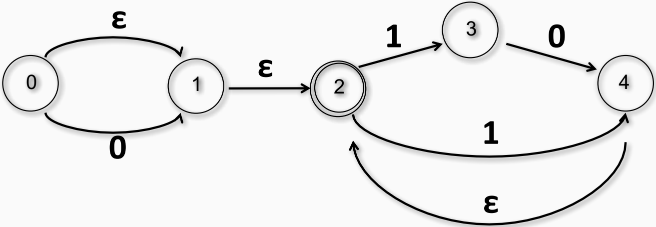
Example



Example

Final NFA simplified slightly to reduce states

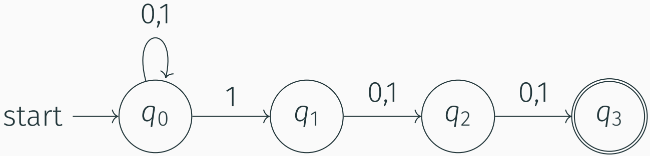




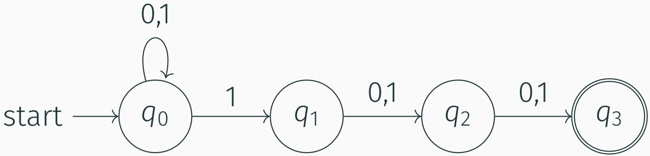
Last thought

Equivalence

Do all NFAs have a corresponding DFA?



Do all NFAs have a corresponding DFA?



Yes but it likely won't be pretty.